Efficient Language Instance Generation

Laura A. Sanchis¹  Mark A. Fulk²
Computer Science Department
The University of Rochester
Rochester, N.Y. 14627

TR 235
January 1988

Abstract

For many familiar languages in P or NP, such as the set of connected graphs or the set of satisfiable CNF boolean formulas, it appears easy to determine whether there exist any instances in the language having a certain size, and even to construct an instance of the desired size on demand. With a little more thought one can also come up with polynomial time nondeterministic or probabilistic procedures which can output all instances of the language of a given size, albeit with repetitions and not necessarily in a uniform manner. In this paper we investigate whether such efficient procedures exist for all languages in P or NP. We exhibit relativizations which show that this question cannot be easily answered. We also relate construction, generation, and categorical generation (generation with unique computation path for each string generated, i.e. no repetitions), to the existence of sparse languages in NP-P and in $D^P-P$. We give a characterization of the languages in NP which can be efficiently constructed from which it can be deduced that all languages in NP can be efficiently constructed if and only if all languages in P can be efficiently constructed. We also deal with parameter-based generation, with construction in the polynomial time hierarchy, and with the existence of certain types of generators viewed as NP machines.

¹AT&T Bell Labs Scholar
²Supported in part by NSF CER grant no. DCR-8320136
1 Introduction

For many familiar languages in P or NP, such as the set of connected graphs or the set of satisfiable CNF boolean formulas, it appears easy to determine whether there exist any instances in the language having a certain size, and even to construct an instance of the desired size on demand. With a little more thought one can also come up with polynomial time nondeterministic or probabilistic procedures which can output all instances of the language of a given size, albeit with repetitions and not necessarily in a uniform manner. In this paper we investigate whether such efficient procedures exist for all languages in P or NP. Among the questions we will look at are: are there efficient existence, construction, and/or generation algorithms for all languages in P or NP; if not, what characterizes languages that can be efficiently generated, and what are the differences among the complexities of existence, construction, generation, and categorical generation (generation with unique computation path for each string generated, i.e. no repetitions).

The need to exhibit or construct particular instances of a language comes up among other places in empirical testing of algorithms, where it is desired to obtain instances having certain characteristics to see how the algorithm operates on such instances [San87]. Another situation where we may want to obtain instances of a language of varying sizes or characteristics is in order to gain intuition when designing algorithms to work on such instances.

Previous work on the generation problem has been done by Jerrum, Valiant, and Vazirani [JVV86], who examined the complexity of uniform generation for relations which can be checked in polynomial time (p-relations), and related this complexity to the complexity of counting and construction for such relations. Other papers have dealt with specific generation problems, such as the generation of certain kinds of graphs, most often in a uniform or at least exhaustive manner ([Wor84],[Ti79],[Bac83],[DW83]). Our viewpoint differs from that in [JVV86] in the following ways. While [JVV86] deals with all p-relations and considers such problems as generating cycles from a given graph, we consider the generation of instances of a language of a given size. In other words, we are dealing with relations of a restricted type, where the first element of a pair is of the form $1^n$ and the second is a string of length $n$ from the language we wish to generate. While it can be easily shown that if construction can be done in polynomial time for all p-relations then P=NP, our relativization results suggest that the same conclusion cannot be reached from the assertion that all P languages can be generated or constructed in polynomial time. We will also deal with generation and construction for both P and NP languages and other language classes in the polynomial hierarchy. Finally, we will consider the complexity of construction and generation without imposing the uniformity constraint. It will be seen that some of the complexity questions for these types of generation are still open and apparently difficult to establish, even for languages in P.

Following is a summary of the main results in this paper. Polynomial time construction and (noncategorical) generation turn out to be equivalent for languages in NP. We will show that if P=NP categorical generation can be done efficiently for all languages in NP, while if there are no sparse languages in $D^P - P$ then construction and generation can be done efficiently for all languages in NP. The converse of the last statement appears difficult to obtain, as we can exhibit an oracle
A relative to which all languages in $\text{NP}^A$ can be efficiently generated, but such that there is a sparse language in $(\text{DP})^A - \text{P}^A$. We also exhibit another oracle for which there exists a language in $\text{P}$ which cannot be efficiently generated. In addition efficient construction or generation for all languages in $\text{P}$ implies that there are no sparse languages in $\text{NP}-\text{P}$. Another relativization provides evidence that obtaining categorical generators for languages in $\text{NP}$ may be harder than obtaining plain constructors or generators.

We give a characterization of the languages in $\text{NP}$ which can be efficiently generated from which it can be deduced that all languages in $\text{NP}$ can be efficiently generated if and only if all languages in $\text{P}$ can be efficiently generated. Although as will be seen shortly only languages in $\text{NP}$ can be efficiently generated, in section 4 we generalize some of our results about polynomial time construction for $\text{NP}$ languages to other levels of the polynomial time hierarchy. Section 5 relates parameter-based generation to the length-restricted generation which is dealt with in the rest of the paper. Finally, the last section shows that nondeterministic polynomial time generators are ubiquitous, in the sense that if $\text{P} \neq \text{NP}$ then there exists a language in $\text{NP}-\text{P}$ with a polynomial time generator, if $\text{P} \neq \text{UP}$ then there exists a language in $\text{UP}-\text{P}$ with a categorical polynomial time generator, and if $\text{P} \neq \text{UP} \cap \text{co-UP}$ then there exists a language in $\text{P}$ which has a polynomial time generator for which one cannot efficiently find a generating computation given a particular string in the language.

2 Construction and Generation for $\text{P}$ and $\text{NP}$ languages

Definition 2.1

1. A polynomial time tallier (PTT) for a language $L$ is a polynomial time deterministic machine which on input $1^n$ outputs 1 if there exist strings of length $n$ in $L$ and outputs 0 otherwise.

2. A polynomial time constructor (PTC) for a language $L$ is a polynomial time deterministic machine which on input $1^n$ outputs a string in $L$ of length $n$, if such a string exists, and outputs $\Lambda$ otherwise.

3. A polynomial time generator (PTG) for a language $L$ is a polynomial time nondeterministic machine which on input $1^n$ outputs a string in $L$ of length $n$, if such a string exists, and outputs $\Lambda$ otherwise. Moreover, for every string $x$ in $L$ of length $n$ there exists some computation of the generator on input $1^n$ which outputs $x$.

4. A categorical PTG for a language $L$ is a PTG for $L$ such that, for each string $x \in L$ of length $n$ there is exactly one computation of the generator on input $1^n$ which outputs $x$.

It is clear that any language that has a PTG has a PTC, and any language that has a PTC has a PTT. In addition any language that has a PTG must be in $\text{NP}$, while any language in $\text{NP}$ which has a PTC actually has a PTG. If a language $L$ in $\text{NP}$ has a PTC $C_L$, we can define a PTG $G_L$ for
$L$ as follows. On input $1^n$, $G_L$ randomly constructs a string $x$ of length $n$ and runs a polynomial time NDTM for $L$ on $x$. If the machine accepts, $G_L$ outputs $x$, otherwise it runs $C_L$ on $1^n$ and outputs its output.

**Proposition 2.1** The following are equivalent:

(a) All languages in NP have PTT's.
(b) All languages in P have PTT's.
(c) There are no tally languages in NP-P.

**Proof:** (sketch) It is not hard to see that $(a) \Rightarrow (b) \Rightarrow (c) \Rightarrow (a)$. For $(b) \Rightarrow (c)$, let $L$ be any tally language in NP with polynomial time NDTM $M$ which accepts $L$ and which runs in time $p(n)$, where $M$ makes exactly $p(n)$ nondeterministic branches on input $1^n$ and $p(n)$ is a strictly increasing polynomial. Define the language $S$ to consist of all strings $x$ of length $p(n)$ such that $M$ accepts on input $1^n$ if it follows the nondeterministic branches coded in $x$. Clearly $S$ is in P, so by assumption $S$ has a PTT. Also $1^n$ is in $L$ if and only if $S$ has a string of length $p(n)$. Hence $L$ is in P. ■

[Boo74] shows that there are no tally languages in NP-P iff E=NE, while in [Har83] it is shown that there are no sparse languages in NP-P if and only if E=NE. We therefore have the following corollary:

**Corollary 2.2** If all languages in NP have PTC's then there are no sparse languages in NP-P.

The above corollary can also be derived from a more direct argument (bypassing the E=NE connection) as follows. Assume all languages in NP have PTC's and let $L$ be any sparse language in NP. There exists a polynomial $p$ such that for each integer $n > 0$ there are at most $p(n)$ strings of length $n$ in $L$. We will construct another language $LC$ as follows. Let $g(n, m) = (n + m - 1)(n + m - 2)/2 + n$. $g : N^+ \times N^+ \rightarrow N^+$ is injective and $g(n, m) \geq nm$ for all $n, m \in N^+$. For each pair of positive integers $n, m$ we will define a set of strings $LC_{n,m}$ belonging to $LC$. Let $q(n)$ be the number of strings in $L$ of length $n$, for each $n > 0$. If $m > q(n)$, then $LC_{n,m}$ is empty. Otherwise, for each ordered sequence $s_1, s_2, \ldots, s_m$ of $m$ distinct strings of length $n$ from $L$, construct a string in $LC_{n,m}$ consisting of the concatenation of $s_1, s_2, \ldots, s_m$ followed by $g(n, m) - nm$ extra 'x' symbols: this produces a string of length $g(n, m)$ belonging to $LC_{n,m}$. Define $LC = \bigcup_{n,m \in N^+} LC_{n,m}$. It is not hard to see that $LC$ belongs to NP, so by assumption $LC$ has a PTC $C_{LC}$. We can define the following polynomial time procedure $A$ for recognizing $L$. Given input $x$ of size $n$, $A$ will repeatedly simulate $C_{LC}$ on inputs $g(n, p(n)), g(n, p(n) - 1)$, etc., until $C_{LC}$ returns something other than $A$. $C_{LC}$ will then have outputted a string of length $g(n, q(n))$ which must contain all $q(n)$ strings in $L$ of length $n$. $A$ need only check this string to see if $x$ appears in it, and accept only if it does. ■
It is an open question whether the converse of the last corollary is true, or equivalently, whether if all languages in NP have PTT's then they all must have PTC's. However, we can show that if there are no sparse languages in $D^P$ then all NP languages have PTC's. The class $D^P$ [PW85] consists of all languages formed from the difference of two NP languages, i.e. $D^P = \{ L_1 - L_2 | L_1, L_2 \in NP \}$.

**Definition 2.2** The prefix closure of a language $L$ is defined as the set of strings $\{ x \# n | n \geq 0, \exists y \text{ such that } |y| = n \text{ and } xy \in L \}$.

**Theorem 2.3** If there are no sparse languages in $D^P - P$, then all languages in NP have PTC's.

**Proof:** Suppose there are no sparse languages in $D^P - P$. Let $L$ be a language in NP and consider the language $L'$ consisting for each $n$ of the smallest string of length $n$ in $L$, if such a string exists. This language is in $D^P$ and is sparse. It's prefix closure is also sparse and in $D^P$. By assumption $L'$ is in $P$ and hence $L$ has a PTC.

The converse of the above theorem cannot be easily shown, as can be seen by the relativization in the next theorem.

**Definition 2.3** Let $A$ be an oracle set. A language $L$ has a PTC relative to $A$ if there exists a polynomial time deterministic oracle machine $M$ which on input $1^n$ and querying oracle $A$ outputs some string $x \in L$ of length $n$, if such a string exists; otherwise $M$ outputs $A$.

We use relativization techniques that have been used among other places in [BGS75], [Kur85], and [CH85]. In particular, the method of coding a sparse language into $(D^P)^A - P^A$ is similar to that used in [CH85].

**Theorem 2.4** There exists an oracle $A$ such that there exists a sparse language in $(D^P)^A - P^A$ and such that all languages in $NP^A$ have PTC's relative to $A$.

**Proof:** Let $P_1, P_2, \ldots$ be an enumeration of all of the polynomial time oracle machines and let $N_1, N_2, \ldots$ be an enumeration of the nondeterministic polynomial time oracle machines. Assume without loss of generality that the running time of both $P_j$ and $N_j$ is bounded by strictly increasing polynomial $p_j$. Strings of odd length put into $A$ will be used to ensure that all languages in $NP^A$ have PTC's relative to $A$. Strings of even length will be used to ensure that there is a sparse language $L_A$ in $(D^P)^A - P^A$, where $L_A = \{ w | (\exists z \in A \text{ such that } |z| = 2|w|) \}$ and $\neg (\exists y \text{ such that } wy \in A \text{ and } |w| = |y|)$. Construction of $A$ will proceed in stages. At the beginning $A$ is empty.

The following will be done at each stage $i \geq 1$. Choose even $m$ such that $2^{m/2} > p_i(m) + p_i(m/2)$ and such that no strings of length greater than or equal to $m$ have been put into $A$ or queried in
computations at previous stages. At stage \( i \) we will determine which strings of length \( m \) go into \( A \) and also which strings of length \( 2n + 1 \) go into \( A \) for all \( n \) such that strings of length \( 2n + 1 \) have not been processed in a previous stage and such that \( 2n + 1 \leq p_i(m) \). (\( A \) will contain no strings of even length except for the strings of length \( m \) considered at each stage \( i \)). Before deciding which strings of length \( m \) go into \( A \) we must have decided which strings of length \( 2n + 1 \) are in \( A \) for all \( n \) such that strings of length \( 2n + 1 \) have not been processed in a previous stage and such that \( 2n + 1 \neq \pi_i(m) \). (\( A \) will contain no strings of even length except for the strings of length \( m \) considered at each stage \( i \)).

Before deciding which strings of length \( m \) go into \( A \) we must have decided which strings of length \( 2n + 1 \) are in \( A \) for all \( n \) such that no processing has been done yet for strings of length \( 2n + 1 \) and such that \( 2n + 1 \neq \pi_i(m) \). This is done as follows. For each \( n \) such that no processing has been done yet for strings of length \( 2n + 1 \) and such that \( 2n + 1 \neq \pi_i(m) \), in order from smallest such \( n \) to largest such \( n \), do the following. Determine all \( j, k \) such that \( j \leq \log n \) and \( j + 1 + k + 1 + \pi_j(k) = n \). For each such pair see if there exists some string of length \( k \) accepted by \( N_j \) using as oracle the strings put into \( A \) so far. If so, choose the smallest such string \( x \) and put the string \( 0^j 1^j 0^j \pi_j(k) 1^j n \) and all its prefixes into \( A \). (The prefixes of a string \( w10^n \) are all strings of the form \( y10^{2n-m} \) where \( y \) is a prefix of \( w \) and \( |y| = m \)). Fix all strings of length \( m \) queried by one of \( N_j \)'s accepting computations on input \( x \) and not yet put in \( A \) to be out of \( A \). If no string of length \( k \) is found to be accepted by \( N_j \), then check whether adding any strings of length \( m \) to \( A \) (other than those already fixed to be out of \( A \) would cause \( N_j \) to accept some string \( x \) of length \( k \). If so, put these strings in \( A \), put \( 0^j 1^j 0^j \pi_j(k) 1^j n \) and all its prefixes into \( A \), and also fix any other strings of length \( m \) queried by this computation to not be in \( A \).

Note that after processing all \( n \) such that \( 2n + 1 \leq p_i(m) \), no more than \( \pi_i(m) \) strings of length \( m \) will have been fixed to be in or out of \( A \). Since \( 2^{m/2} < \pi_i(m) \), there exists at least one string \( w \) of length \( m/2 \) such that no string \( w^y \) of length \( m \) has been fixed to be in or out of \( A \). Run \( P_i \) on string \( w \) using as oracle the strings put into \( A \) so far. If \( P_i \) rejects do nothing. Else choose some string \( y \) such that \( w^y \) was not queried in \( P_i \)'s computation on \( w \) (this can be done since \( p_i(m/2) < 2^{m/2} \) and add \( w^y \) to \( A \). Also in either case for every string \( w' \neq w \) of length \( m/2 \) find a string \( y' \) such that \( w'y' \) has not been fixed to be in or out of \( A \) and such that the string was not queried in \( P_i \)'s computation on \( w \), and put \( w'y' \) in \( A \) (thus ensuring \( L_A \) will be sparse). This completes the processing at stage \( i \).

Note \( L_A \in (D^p)^A \), by the above construction \( L_A \) is not in \( PA \), and \( L_A \) is sparse. Also it is not hard to see that if \( L \) is equal to the language accepted by any \( N_j^A \), \( L \) has a PTC with respect to \( A \).

Note that if in the above construction we omit the steps used to ensure that there is a sparse language in \( (D^p)^A - PA \) (i.e. only put odd strings into \( A \), then we have a sparse oracle \( A \) relative to which all languages in \( NP^A \) will have PTC's.

We can also exhibit an oracle relative to which there exists a language in \( P \) which does not have a PTC.

**Theorem 2.5** There exists an oracle \( B \) and a language \( L \in P^B \) such that \( L \) does not have a PTC relative to \( B \).

**Proof:** The proof is a straightforward diagonalization. \( B \) is constructed in stages. Let \( PO_1, PO_2, \ldots \) be an enumeration of the polynomial time oracle machines with output tape and assume that the
computation time of $PO_j$ is bounded by polynomial $p_j$. At the $i$th stage, $i \geq 1$, choose $m$ such that $2^m > p_i(m)$ and such that no string of length greater than or equal to $m$ has been put into $B$ or queried in computations at previous stages. Run $PO_i$ on input $1^m$ using as oracle the strings put into $B$ so far. If $PO_i$ outputs $\Lambda$ then choose a string $x$ of length $m$ not queried by $PO_i$'s computation and add this string to $A$. Otherwise do not add any string to $B$ at this stage. Let $L = B$. Clearly $L \in P^B$ and no $PO_j^B$ can be a PTC for $L$ relative to $B$. 

If the construction in the last proof is combined with the generic oracle construction used in [BI87] used to ensure that no infinite NP language is P-immune, we get part (c) of the following proposition.

**Proposition 2.6**

(a) Any infinite set that has a PTC has an infinite subset in $P$.

(b) If all NP languages have PTC’s then NP has no infinite P-immune sets.

(c) There exists an oracle $C$ such that $NP^C$ has no infinite P-immune sets but such that there exists a language in $P^C$ which does not have a PTC relative to $C$.

The above relativizations show that it is difficult to establish whether or not all languages in NP have PTC’s. We can however characterize the NP languages that have PTC’s (or PTG’s). The following definition will be useful.

**Definition 2.4** Prefix-P consists of the class of languages $L$ such that the prefix closure of $L$ is in $P$.

Clearly Prefix-P $\subset$ P. Also P=NP iff Prefix-P = P. Moreover if a language is in Prefix-P it has a categorical PTG which constructs a string of length $n$ in $L$ by successively adding a bit and checking whether the string constructed so far is a valid prefix for a string in $L$ of length $n$. We therefore have the following.

**Proposition 2.7** If P=NP then all languages in NP have categorical PTG’s.

Even if P $\neq$ NP, the class of languages that have PTG’s is not restricted to those in Prefix-P. In fact the NP complete language 3-SAT has a PTG, described informally as follows. Given inputs $n$, $m$ denoting respectively the number of variables $x_1, x_2, \ldots, x_n$, and the number of clauses, randomly assign a truth value T or F to each of $x_1, x_2, \ldots, x_n$. Let $u_i = x_i$ if $x_i$ was assigned T, $u_i = \bar{x}_i$ otherwise. To form each of the $m$ clauses first randomly choose some $u_i$, thus assuring that the clause will be true, and then randomly choose any 2 more variables for the clause from
among \( x_1, x_2, \ldots, x_n, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n \). Clearly each satisfiable formula with \( n \) variables and \( m \) clauses is generated in this manner. Moreover the number of times a particular satisfiable formula is generated is proportional to the number of different assignments that satisfy it. (The input \( n, m \) to the above procedure is not actually the length of the string that will be generated, although it is polynomially related to it. It is however not hard to construct from this informally described procedure a formal PTG for some encoding of the 3-SAT language. See also section 5.)

**Definition 2.5** A language \( L \) is prefixable if there exists a language \( L' \) in Prefix-P, a polynomial \( f \) which is one-to-one, and a polynomially computable onto function \( g : L' \to L \) such that \( |x| = f(|g(x)|) \) for all \( x \in L' \).

**Theorem 2.8** A language \( L \) has a PTG iff it is prefixable.

**Proof:** (sketch) Suppose \( L \) is prefixable. Then there exist \( L', f, g \) with the required properties as stated in the definition. Since \( L' \) is in Prefix-P it has a PTG \( G_{L'} \). Define PTG \( G_L \) for \( L \) as follows. On input \( n \), \( G_L \) computes \( f(n) \) and simulates \( G_{L'} \) on input \( f(n) \). Then \( G_L \) outputs \( y = g(x) \) where \( x \) is the output of \( G_{L'} \).

Suppose conversely that \( L \) has a PTG \( G_L \) time-bounded by polynomial \( p(n) \). Let \( L' \) consist of all strings \( x#y \) where for some \( n \geq 1 \) \( |y| = n, |x| = p(n) \), and if \( G_L \) follows the branches coded in \( x \), \( G_L \) outputs \( y \). It can be easily seen that \( L' \) is in Prefix-P. Define \( f(n) = n + 1 + p(n) \) and define \( g : L' \to L \) by \( g(x#y) = y \).

**Corollary 2.9** All languages in NP have PTG's iff all languages in \( P \) have PTG's.

**Proof:** (idea) This follows from the fact that any NP language can be prefixed by the language consisting of the accepting computations for the strings in the language.

Finally, another fact which has bearing on the question of whether or not all languages in NP have PTG's is the existence of a language in NP which is complete in this regard.

**Proposition 2.10** There exists a language \( L \) in NP such that: \( L \) has a (categorical) PTG iff all languages in NP have (categorical) PTG's.

**Proof:** (sketch) Define \( g(n, m) = (n + m - 1)(n + m - 2)/2 + n \). \( g \) is injective and \( g(n, m) \geq nm \) for all \( n, m \in \mathbb{N}^+ \). We may assume the \( i \)th NP machine has time bound \( p_i \) where \( p_i \) is a one-to-one polynomial. For each pair of positive integers \( i, n \), we will code each string of length \( n \) accepted by the \( i \)th NP machine into a string of length \( g(i, p_i(n)) \), using padding. The language \( L \) will consist of all strings formed in this way from all such pairs. Clearly \( L \) is in NP and if it has a (categorical) PTG each NP language has a (categorical) PTG.
3 Categorical PTG’s

UP consists of all NP languages that are accepted by nondeterministic polynomial time machines with unique accepting computation paths [Val76]. Such machines are called categorical. Categorical language generators, which were defined in the previous section, are generators with unique generation paths for each string generated. Categorical generators are desirable since they may be “closer” to uniform generators than general PTG’s. This may be seen as follows. Suppose a language \( L \) has a categorical PTG \( G_L \) and assume without loss of generality that at each nondeterministic branch in the computation there are two possible paths to follow. We may assign probability 1/2 to each of these paths, thus making \( G_L \) a probabilistic generator. Let \( p \) be a polynomial bounding the length of each computation. Then whenever a string \( x \) of length \( n \) is generated by \( G_L \), we can compute (a posteriori) the probability of \( x \) being generated on input \( n \). This probability is equal to \( 1/(2^q) \) where \( q \) is the number of probabilistic choices made by \( G_L \) when outputting \( x \). We also know a lower bound on the probability of any single string \( x \in L \) being generated by \( G_L \) on input \( 1^n \); this lower bound is \( 1/(2^{p(n)}) \). We may then construct a new probabilistic machine \( G'_L \). On input \( 1^n \), \( G'_L \) first runs \( G_L \) on input \( n \). If \( G_L \) outputs \( x \) with probability \( 1/(2^q) \), then with probability \( 1/(2^{p(n)-q}) \) \( G'_L \) outputs \( x \), otherwise it outputs nothing. Thus on input \( 1^n \), every string \( x \in L \) of length \( n \) has the same probability \( 1/(2^{p(n)}) \) of being generated. However, \( G'_L \) may also fail to output anything. It is not hard to see that the probability of failure is \( (2^{p(n)} - L_n)/(2^{p(n)}) \), where \( L_n \) is the number of strings of length \( n \) in \( L \). Whether \( G'_L \) is a practical generator depends on how big the probability of failure is as a function of \( n \). (Similar ideas involving using knowledge of a posteriori probabilities may be found in [Bac83] and [JVV86]).

Definition 3.1 A language \( L \) is categorically prefixable if there exists a language \( L' \) in Prefix-P, a one-to-one polynomial \( f \), and a polynomial time computable function \( g : L' \to L \) such that \( g \) is onto and one-to-one and \( |x| = f(|g(x)|) \) for all \( x \in L' \).

Proposition 3.1

(a) If a language has a categorical PTG it is in UP.
(b) A language has a categorical PTG iff it is categorically prefixable.
(c) All languages in \( P \) have categorical PTG’s iff all languages in \( UP \) have categorical PTG’s.
(d) If a sparse language has a categorical PTG then it is \( P \)-printable.
(e) All sparse languages in \( NP \) have categorical PTG’s iff there are no sparse languages in \( NP-P \).

Proof: (sketch) (d) The number of nodes in the tree of all possible computations of a categorical PTG on input \( 1^n \) is bounded by a polynomial in \( n \). Hence it can be totally explored in polynomial time.
(e) (\( \Leftarrow \)) Suppose that there are no sparse languages in \( NP-P \) and let \( L \) be a sparse language in \( NP \).
The prefix closure of \( L \) is also a sparse language in NP so by assumption it is in P. Hence \( L \) is in Prefix-P, implying that it has a categorical PTG.

The following relativization shows that Part (e) above cannot easily be extended to say that if there are no sparse languages in NP-P then all NP languages have categorical PTG's.

**Theorem 3.2** There exists an oracle \( D \) such that there are no sparse languages in \( NP^D - P^D \) but there exists a language \( L \in P^D \) which does not have a categorical PTG with respect to \( D \).

**Proof:** (sketch) Let \( N_0, N_1, \ldots \) be an enumeration of all of the nondeterministic polynomial time oracle machines with output tape. Let \( N_1, N_2, \ldots \) be an enumeration of all of the nondeterministic polynomial time oracle machines. Assume that the computation times of \( N_0 \) and \( N_j \) are both bounded by polynomial \( p_j \) which is strictly increasing. \( C \) will be built up in stages. There are two types of strings that will be put into \( D \). Strings of odd length will be used to ensure that all tally languages in \( NP^D \) are in \( P^D \). The way this is done is similar to the way PTC's were coded into \( A \) in the proof of Theorem 2.4. Strings of even length will be used to ensure that \( L = D \) will not have a categorical PTG relative to \( D \). Briefly, at the \( i \)th stage we choose even \( m \) such that \( 2^m > p_i(m) + p_j(m) \) and such that no strings of length greater than or equal to \( m \) have been put into \( D \) or queried in computations at previous stages. We then determine which strings of length \( 2n + 1 \) go into \( D \) for all \( n \) such that strings of length \( 2n + 1 \) have not been processed in a previous stage and such that \( 2n + 1 \leq p_i(m) \). In the process of doing this we will fix no more than \( p_i(m) \) strings of length \( m \) to be in or out of \( D \). We then run \( NO_i \) with oracle \( D \) through all its possible computations on input \( 1^m \). \( NO_i \) may or may not act as a categorical PTG on input \( 1^m \). It does not, i.e. if it outputs some strings of lengths other than \( m \) or outputs the same string twice, then no more processing need be done. Otherwise check whether the strings outputted are exactly the strings of length \( m \) that have been put into \( D \) so far. If not, then again \( NO_i \) cannot generate \( D \). Otherwise note that no more than \( p_i(m) \) strings are outputted by \( NO_i \), and since each computation must output a distinct string, \( NO_i \) has at most \( p_i(m) \) computations on input \( 1^m \), each querying at most \( p_i(m) \) strings of length \( m \). Hence we may find a string of length \( m \) not outputted or queried by \( NO_i \) on input \( 1^m \) in any of its computations, and not fixed to be in or out of \( D \), and add this string to \( D \).

A technique similar to the one used in the proof of the last theorem can be used to prove the following.

**Theorem 3.3** There exists an oracle \( E \) such that every language in \( NP^E \) has a PTC relative to \( E \) but there exists a language in \( P^E \) which does not have a categorical PTG relative to \( E \).

It appears that many natural languages in NP have PTG's with the characteristic that the number of times a string is generated is equal to the number of accepting computations for the string in some NP machine for the language. An example of this is the PTG for 3-SAT described in the previous section; for this PTG the number of times each string is generated equals the number
of satisfying assignments it has. It is not hard to see that all NP languages will have such PTG's iff all P languages have categorical PTG's.

**Theorem 3.4** All languages in P have categorical PTG's iff the following holds: if L is any language in NP and M is a polynomial time NDTM accepting L, then L has a PTG $G_L$ for which given any string $x$ of length $n$ in $L$ the number of computations in which $x$ is outputted by $G_L$ on input $1^n$ equals the number of accepting computations for $x$ in $M$.

**Proof:** Suppose all P languages have categorical PTG's. Let $L$ be a language in NP and M be a polynomial time NDTM accepting $L$. Let $p(n)$ be a bound on the number of nondeterministic choices made by $M$ on inputs of length $n$, where $p(n)$ is a one-to-one polynomial. We may modify $M$ so that it makes exactly $p(n)$ nondeterministic choices on each input of length $n$ and so that it still has the same number of accepting computations for each input. Define $L'$ to consist of all strings of the form $x#y$ where $|x| = p(n)$, $|y| = n$, and $M$ on input $y$ and following the nondeterministic choices coded in $x$, accepts. $L'$ is in P so by assumption it has a categorical PTG. This PTG in turn yields a PTG for $L$ having the required properties. ■

4 Construction in the Polynomial Time Hierarchy

Since any language that has a PTG is in NP, unless the polynomial time hierarchy collapses language classes above NP in the polynomial time hierarchy cannot have PTG's. We can however ask whether they have PTC's. The following propositions are generalizations of Corollary 2.9, Corollary 2.2, and Theorem 2.3, respectively, and have similar proofs. (See [Sto76] for definition of the polynomial time hierarchy).

**Proposition 4.1** All languages in $\Delta^P_k$ have PTC's iff all languages in $\Sigma^P_k$ have PTC's.

**Proposition 4.2**

1. If all languages in $\Sigma^P_k$ have PTC's then all sparse languages in $\Sigma^P_k$ are P-printable.
2. If all languages in $\Pi^P_k$ have PTC's then all sparse languages in $\Pi^P_k$ are P-printable.

**Definition 4.1** For $k \geq 1$, $Diff_k = \{L_1 - L_2 | L_1, L_2 \in \Sigma^P_k\}$. Note $Diff_1 = D^P$.

**Proposition 4.3** If there are no sparse languages in $Diff_k - P$, then all languages in $\Sigma^P_k$ have PTC's.
5 Parameter Based Generation

Up to now we have considered construction and generation of strings required to have a specified length. In practical situations it is usually desired to generate a string having certain parameter values related to length. For example, it may be required to obtain a CNF satisfiable formula having \( n \) variables and \( m \) clauses, or a graph possessing a certain property having a specified number of vertices and edges. This section presents a connection between this kind of parametrized generation and the length-restricted generation discussed in previous sections.

Definition 5.1 Let \( l_1, \ldots, l_k \) be polynomial time computable functions, \( l_j : \sum^* \rightarrow N \) for \( 1 \leq j \leq k \), and \( q, r \) be polynomials such that \( l_j(x) \leq q(|x|) \) for \( 1 \leq j \leq k \) and \( |x| \leq r(l_1(x), \ldots, l_k(x)) \) for all \( x \). An \((l_1, \ldots, l_k)\)-PTG for a language \( L \) is a polynomial time NDTM \( M \) which on input \( 1^m \# \ldots \# 1^n \), either outputs a string \( x \in L \) such that for \( 1 \leq j \leq k \), \( l_j(x) = v_j \), or outputs the symbol \( \Lambda \) indicating that no such string exists. Furthermore for every string \( x \in L \) such that \( l_j(x) = v_j \) for \( 1 \leq j \leq k \) there exists some computation of \( M \) on input \( 1^m \# \ldots \# 1^n \) which outputs \( x \).

Let \( L \) be a language in NP. The following relationships are easily checked. Suppose there exists a polynomial \( p \) such that for all \( x \in L \), \( p(l_1(x), \ldots, l_k(x)) = |x| \). Then if \( L \) has an \((l_1, \ldots, l_k)\)-PTG it has a length restricted PTG. Moreover if \( p \) is one-to-one then if \( L \) has a length restricted PTG it has an \((l_1, \ldots, l_k)\)-PTG. Suppose there exist polynomials \( p_1, \ldots, p_k \) such that for all \( x \in L \), \( p_j(|x|) = l_j(x) \) for \( 1 \leq j \leq k \). Then if \( L \) has a length-restricted PTG it has an \((l_1, \ldots, l_k)\)-PTG. Also if the function \( P : N \rightarrow N^k \) defined by \( P(m) = (p_1(m), \ldots, p_k(m)) \) is one-to-one, then if \( L \) has an \((l_1, \ldots, l_k)\)-PTG it has a length-restricted PTG. The following theorem gives a more general kind of result.

Theorem 5.1 Let \( (l_1, \ldots, l_k), q, r \) be as in the above definition. Then if all languages in NP have (length-restricted) PTG's, then all languages in NP have \((l_1, \ldots, l_k)\)-PTG's.

Proof: Let \( L \) be a language in NP and suppose all NP languages have length-restricted PTG's. Recall \( g : N^+ \times N^+ \rightarrow N^+ \) was defined to be a one-to-one polynomial such that \( g(n, m) \geq nm \) for all \( n, m \in N^+ \). By repeated compositions of \( g \) one can define a polynomial function \( g_{k+1} : (N^+)^{k+1} \rightarrow N^+ \) on \( k + 1 \) variables which is injective and for which \( g_{k+1}(n_1, \ldots, n_{k+1}) \geq n_1 \ldots n_{k+1} \) for all \( n_1, \ldots, n_{k+1} \in N^+ \). We will define a language \( S \) as follows. For \( x \in L \), let \( f(x) = g_{k+1}(l_1(x) + 1, \ldots, l_k(x) + 1, r(l_1(x), \ldots, l_k(x)) + 1) \) and let \( s(x) \) be the string of length \( f(x) \) consisting of the concatenation of \( x \) and \( f(x) - |x| \) "*" symbols (where * is not in \( \sum \)). For each \( x \in L \), put the corresponding \( s(x) \) in \( S \). \( S \) is clearly in NP, so if all languages in NP have (length-restricted) PTG's, \( S \) has a PTG \( G_S \). We can then define an \((l_1, \ldots, l_k)\)-PTG \( G_L \) for \( L \) as follows. On input \( 1^m \# \ldots \# 1^n \), \( G_L \) computes \( m = g_{k+1}(v_1 + 1, \ldots, v_k + 1, r(v_1, \ldots, v_k) + 1) \) and runs \( G_S \) on input \( m \). \( G_S \) will then output a string \( y \) of length \( m \) in \( S \) (if such a string exists); by construction of \( S \), \( y \) will consist of a string \( x \in L \) with parameter values \( v_1, \ldots, v_k \) followed by '*s. \( G_L \) should then output \( x \). □
6 PTG's as NP Machines and Traceability

If we have a PTG for a language $L$ in NP, then as has been pointed out before, we can use this PTG as a polynomial time NDTM for recognizing $L$. In other words, all languages that have PTG's have NP machines. We have seen that it is hard to determine whether the converse is true. However, for various types of NP machines (e.g. categorical machines), if a machine of this type exists for a language $L$, it is possible to find another language which is "close" to $L$ and which has a PTG which viewed as an NP machine has the same characteristics as the NP machine for $L$. Thus PTG's as NP machines are rather diversified. This section presents some of these kinds of results.

Proposition 6.1

(a) If $P \neq NP$ then there exists a PTG for a language in NP-P
(b) If $P \neq UP$ then there exists a categorical PTG for a language in UP-P.

Proof: (a) This follows from the fact that many NP-complete languages, e.g. 3-SAT have PTG's.
(b) Let $L$ be a language in UP-P. $L$ has a categorical machine $M$ which recognizes $L$ and which makes at most $p(n)$ moves on inputs of length $n$, where $p(n)$ is a one-to-one polynomial. We may modify $M$ so that it makes exactly $p(n)$ nondeterministic branches on inputs of length $n$ and remains categorical. Define the language $L' = L1 \cup L2$. $L1$ consists of all strings of the form $x \# y$ where $|x| = n$, $|y| = p(n)$, and if $M$ is run on input $x$ following the branches coded in $y$, $M$ rejects. $L2$ consists of all strings $x*p(n)+1$ where $|x| = n$ and $x \in L$. Clearly $L'$ is in UP but not in P. Also $L'$ has a categorical PTG: on input $1^m$ where $m = n + 1 + p(n)$ this PTG randomly constructs strings $x$ and $y$ of lengths $n$ and $p(n)$ respectively. It then runs $M$ on input $x$ following the branches coded in $y$. If $M$ rejects, the PTG outputs $x \# y$. Otherwise it outputs $x*p(n)+1$.

Definition 6.1 An NP machine for a language $L$ is traceable if there exists a polynomial time procedure which on input $x \in L$ outputs an accepting computation of the machine on input $x$.

In terms of PTG's viewed as NP machines, this concept has the following meaning: A PTG for a language $L$ is traceable if there exists a polynomial time procedure which on input $x \in L$ outputs some computation of the PTG which on input $1^n$ outputs $x$. Clearly any language that has a traceable PTG must be in $P$, but it is not clear that all PTG's for languages in $P$ must be traceable.

Lemma 6.2 If a language in $P$ has a PTG it has a traceable PTG.

In [BD76] it was shown that if $P \neq NP \cap \text{co-NP}$ there is an NP machine for $\Sigma^*$ which is not traceable. Similarly if $P \neq UP \cap \text{co-UP}$ there is a UP machine for a language in $P$ which is not traceable [GS84]. These hypotheses also imply the existence of non-traceable PTG's.
Proposition 6.3

(a) If $P \neq UP \cap \text{co-UP}$ then there exists a categorical PTG for a language in $P$ which is not traceable.

(b) If $P \neq NP \cap \text{co-NP}$ then there exists a PTG for a language in $P$ which is not traceable.

Proof: (a) Let $L$ be a language in $(UP \cap \text{co-UP}) - P$. Let $M_1, M_2$ be categorical machines accepting $L$ and $\overline{L}$, respectively. Let $p$ be a one-to-one polynomial such that $p(n)$ bounds the number of nondeterministic choices made by both $M_1$ and $M_2$ on inputs of length $n$. We may modify $M_1$ and $M_2$ so that they always make exactly $p(n)$ nondeterministic choices, while remaining categorical. Define languages $L_1$ and $L_2$ as follows. $L_1 = U_1 \cup V_1$. $U_1$ consists of all strings of the form $x \# y$ where $|x| = n$, $|y| = p(n)$, and $M_1$ on input $x$ and following branches coded in $y$ rejects. $V_1$ consists of all strings of the form $x \# (p(n)+1)$ where $|x| = n$ and $x \in L$. $L_2 = U_2 \cup V_2$. $U_2$ consists of all strings of the form $x \% y$ where $|x| = n$, $|y| = p(n)$, and $M_2$ on input $x$ and following computation coded in $y$ rejects. $V_2$ consists of all strings of the form $x \% (p(n)+1)$ where $|x| = n$ and $x \in \overline{L}$. $L_1, L_2$ have categorical PTGs $G_{L_1}$ and $G_{L_2}$ respectively, as shown in the proof of Proposition 6.1. $L_1 \cap L_2$ is empty, hence a categorical PTG $G$ can be constructed for $L_1 \cup L_2$ which first nondeterministically chooses to generate a string from $L_1$ or from $L_2$ and then runs $G_{L_1}$ or $G_{L_2}$. Also it is evident that $L_1 \cup L_2$ is in $P$. However, $G$ cannot be traceable for then $L$ would be in $P$.

(b) Let $L$ be a language in $(NP \cap \text{co-NP}) - P$. Define the language $L'$ as follows:

$$L' = \{1x|x \in L\} \cup \{00y|y \in \Sigma^*\}$$

So

$$\overline{L'} = \{1x|x \in \overline{L}\} \cup \{01y|y \in \Sigma^*\}$$

$L'$, $\overline{L'}$ are both in $NP$ and have PTG's. Hence we may define a PTG for $\Sigma^*$ whose first branch point consists of deciding whether to output a string in $L'$ or $\overline{L'}$. If this PTG were traceable, $L'$ would be in $P$, which would imply that $L$ is in $P$, contradiction.

Finally, the converses for some of the statements in the above propositions are also true, so we have the following corollary.

Corollary 6.4

(a) $P=NP$ iff all PTG's are traceable.

(b) $P=UP$ iff all categorical PTG's are traceable.

(c) $P = UP \cap \text{co-UP}$ iff all categorical PTG's for languages in $P$ are traceable.
7 Acknowledgements

We'd like to thank Joel Seiferas and Mandayam Srinivas for useful comments and in particular J. Seiferas for suggesting the equivalence between PTG's and PTC's for languages in P. We thank the Xerox Corporation University Grants Program for providing equipment used in the preparation of this paper.

References


