A Taxonomy of Complexity Classes of Functions

Alan L. Selman *
Department of Computer Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, New York 14260

June 5, 1992

Abstract

This paper comprises a systematic comparison of several complexity classes of functions that are computed nondeterministically in polynomial time or with an oracle in NP. There are three components to this work.

- A taxonomy is presented that demonstrates all known inclusion relations of these classes. For (nearly) each inclusion that is not shown to hold, evidence is presented to indicate that the inclusion is false. As an example, consider FewPF, the class of multivalued functions that are nondeterministically computable in polynomial time such that for each x, there is a polynomial bound on the number of distinct output values of $f(x)$. We show that FewPF $\subseteq \mathrm{PF}^{\mathrm{NP}}_{tt}$. However, we show $\mathrm{PF}^{\mathrm{NP}}_{tt} \subseteq$ FewPF if and only if NP = co-NP, and thus $\mathrm{PF}^{\mathrm{NP}}_{tt} \subseteq$ FewPF is likely to be false.
- Whereas it is known that $\mathrm{P}^{\mathrm{NP}}(O(\log\ n)) = \mathrm{P}^{\mathrm{NP}}_{tt} \subseteq \mathrm{P}^{\mathrm{NP}}$ [Hem87, Wagb, BH88], we show that $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) = \mathrm{PF}^{\mathrm{NP}}_{tt}$ implies P = FewP and R = NP. Also, we show that $\mathrm{PF}^{\mathrm{NP}}_{tt} = \mathrm{PF}^{\mathrm{NP}}$ if and only if $\mathrm{P}^{\mathrm{NP}}_{tt} = \mathrm{P}^{\mathrm{NP}}$.
- We show that if every nondeterministic polynomial-time multivalued function has a single-valued nondeterministic refinement (equivalently, if every honest function that is computable in polynomial-time can be inverted by

*The author acknowledges support by the National Science Foundation under grant CCR-9002292.

a single-valued nondeterministic function), then there exists a disjoint pair of NP-compete sets such that every separator is NP-hard. The latter is a previously studied open problem that is closely related to investigations on promise problems. This result motivates a study of reductions between partial multivalued functions.

1 Introduction

In this paper we study complexity classes of functions that correspond to well-known complexity classes of sets (decision problems). For the classes to be discussed, all possible inclusions are shown. For inclusions that are not shown, we provide evidence as to why they probably do not hold. For each inclusion that is shown, we prove results about whether the inclusion is proper. Usually, two function classes collapse if and only if the corresponding set classes collapse. However, surprises will occur. For example, whereas it is well known that $\mathrm{P}^{\mathrm{NP}}(O(\log\ n)) = \mathrm{P}^{\mathrm{NP}}_{tt}$ ([Hem87, Wagb, BH88]), we will show that the corresponding classes of functions, $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$and $\mathrm{PF}^{\mathrm{NP}}_{tt}$, are equal only if P = FewP and P = R. (FewP is defined in [AR88].) Another surprise is a connection with the study of promise problems and with a conjecture about promise problems of Even, Selman, and Yacobi ([ESY84]).

This paper is an attempt at a systematic study of the complexity of computing functions. It is certainly the case that *partial functions* are the fundamental objects studied in recursive function theory, so it is somewhat surprising that complexity theory has developed largely as a theory of the classification of languages only, and has ignored a classification of function classes. This is unfortunate, for some of the exciting recent developments in complexity theory directly concern complexity of function classes. This is true for example of Toda's result that $\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}}$ [Tod89]. Krentel's work that relates the complexity of optimization problems to set recognition problems in the Boolean Hierarchy is concerned not only with the complexity of function classes [Kre88], but is directly concerned with relating the complexity of function classes with those of certain set recognition problems.

The problems we traditionally think of as set recognition problems are more naturally thought of as functional computational problems. Presumably, one does not care to know only whether a graph has a hamiltonian circuit, but one wants a hamiltonian circuit to be output, if in fact one exists. (Certainly, a chemist wishes to find an isomorphism between two graphs, if one exists, and is not content with merely an answer to the existence question.) A given graph might have no hamiltonian circuit, or might be several, or even exponentially many. Thus, it is natural to think of the hamiltonian circuit problem as a partial multivalued function, moreover, this function has an obvious nondeterministic algorithm.

Showing that a problem is complete or hard for a class has in practice been sufficient for showing that no efficient algorithm exists for computing witnesses to the problem. This is because the combinatorial problems we usually study are self-reducible. It is not known whether all problems in NP are self-reducible (cf. [Sel88]), so it is not known whether the familiar approach can work in all cases. This is a primary motivation for studying function classes directly, and for directly inquiring about the complexity of multivalued functions that can be solved by nondeterministic algorithms.

We might have subtitled this paper, "On the complexity of inverting one-way functions," but a meaningful explanation of this comment must wait until several technical definitions are given. For now, let us note that the inverse of any honest polynomial-time computable function is itself a function, albeit a multivalued function, and that this inverse is solvable by an obvious nondeterministic algorithm. Thus, for this reason also, it makes sense to inquire directly about the complexity of multivalued functions that can be solved by nondeterministic algorithms.

1.1 Definitions of some complexity classes of functions

Several of the definitions we consider originate in [BLS84].

A transducer T is a nondeterministic Turing machine with a read-only input tape, a write-only output tape, and accepting states in the usual manner. T computes a value y on an input string x if there is an accepting computation of T on x for which y is the final contents of T's output tape. (In this case, we will write $T(x) \mapsto y$.) Such transducers compute partial, multivalued functions.

- NPMV is the set of all partial, multivalued functions computed by nondeterministic polynomial time-bounded transducers;

- NPSV is the set of all $f \in$ NPMV that are single-valued;
- PF is the set of all partial functions computed by deterministic polynomial time-bounded transducers.

Given partial multivalued functions f and g, define g to be a *refinement* of f if $dom(g) = dom(f)$ and for all $x \in dom(g)$ and all y, if $g(x) \mapsto y$, then $f(x) \mapsto y$.

Let $\mathcal{F}$ and $\mathcal{G}$ be classes of partial multivalued functions. Purely as a convention, if f is a partial multivalued function, we define $f \in_c \mathcal{G}$ if $\mathcal{G}$ contains a refinement g of f, and we define $\mathcal{F} \subseteq_c \mathcal{G}$ if for every $f \in \mathcal{F}$, $f \in_c \mathcal{G}$. Thus, "NPMV $\subseteq_c$ PF" would mean that every partial multivalued function in NPMV can be computed efficiently by some deterministic polynomial time transducer.

The following proposition is known:

Proposition 1 *1.* [Sel90] NPMV $\subseteq_c$ PF *if and only if* P = NP.

2. [SXB83] NPSV $\subseteq$ PF *if and only if* P = NP.

Following Valiant [Val76], given a class of partial functions $\mathcal{F}$, let $\mathcal{F}_g$ denote the class of all functions $f \in \mathcal{F}$ such that $graph(f) \in$ P. [1] Valiant noticed that ordinary search problems associated with NP decision problems are functions in NPMV$_g$. That is, let $R(x,y)$ be an arbitrary relation in P, and let p be a polynomial, so that $A = \{x \mid \exists y_{|y| \leq p(|x|)} R(x,y)\}$ is an arbitrary set in NP. Define

$$f_A(x) \mapsto \begin{cases} \text{any } y \text{ such that } |y| \leq p(|x|) \wedge R(x,y), & \text{if } x \in A, \\ \uparrow, & \text{otherwise.} \end{cases}$$

These naturally occurring functions are in NPMV$_g$.

A function $f \in$ PF is *honest* if there is a polynomial q such that for every y in $range(f)$ there exists x in $dom(f)$ such that $f(x) = y$ and $|x| \leq q(|y|)$. The inverse of every honest function $f \in$ PF belongs to NPMV$_g$, and the inverse of every honest one-one function

[1] $graph(f) = \{\langle x, y\rangle \mid f(x) \mapsto y\}$.

$f \in$ PF belongs to NPSV_g. (These are easy exercises; the main point to keep in mind is that the inverse of a single-valued function is a multivalued function.)

Given a partial, multivalued function f, for all x, we define

$$set\text{-}f(x) = \{y \mid f(x) \mapsto y\}.$$

- FewPF is the set of all functions f in NMPV such that for some polynomial p and all x, $\|set\text{-}f(x)\| \leq p(|x|)$.

If $f \in$ FewPF, then it makes sense to seek *all* of these values. For this reason, we redefine our conventions for $f \in$ FewPF and for $\mathcal{F} \subseteq$ FewPF. For a finite set $\{y_1, \cdots, y_n\}$, where the elements are listed in lexicographic order, $c(\{y_1, \cdots, y_n\}) = \%y_1 \cdots \%y_n\%$, where % is a symbol not in Σ. If $\|set\text{-}f(x)\|$ is finite for each x, then the function $c(set\text{-}f)$ is defined by $c(set\text{-}f)(x) = c(set\text{-}f(x))$. $c(set\text{-}f)$ is a single-valued total function. Given $f \in$ FewPF and a class of single-valued functions $\mathcal{G}$, define $f \in_c \mathcal{G}$ to mean that $c(set\text{-}f) \in \mathcal{G}$, and given a class of functions $\mathcal{F} \subseteq$ FewPF, let "$\mathcal{F} \subseteq_c \mathcal{G}$" denote the assertion that "for every $f \in \mathcal{F}$, $c(set\text{-}f) \in \mathcal{G}$." The intended meaning will always be clear from the context.

Finally, we mention the class of functions computed in polynomial time with oracles in NP, namely $\mathrm{PF}^{\mathrm{NP}}$, and the class of functions computed in polynomial time with at most $O(\log\ n)$ queries to an oracle in NP, namely $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$. Krentel [Kre88] studied these classes and demonstrated that if $\mathrm{PF}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$, then P = NP. Kadin [Kad87] showed that if NP has sparse Turing-complete sets, then the entire polynomial hierarchy collapses to $\mathrm{P}^{\mathrm{NP}}(O(\log\ n))$. The class $\mathrm{P}^{\mathrm{NP}}(O(\log\ n))$ has emerged in recent years as a natural and robust complexity class. It is equal to the class of sets that are polynomial time truth-table reducible to SAT [Hem87, Wagb, BH88], namely $\mathrm{P}^{\mathrm{NP}}_{tt}$, and has natural complete sets [Kre88, KSW86, Kad87, Waga].

Thus, the relationships

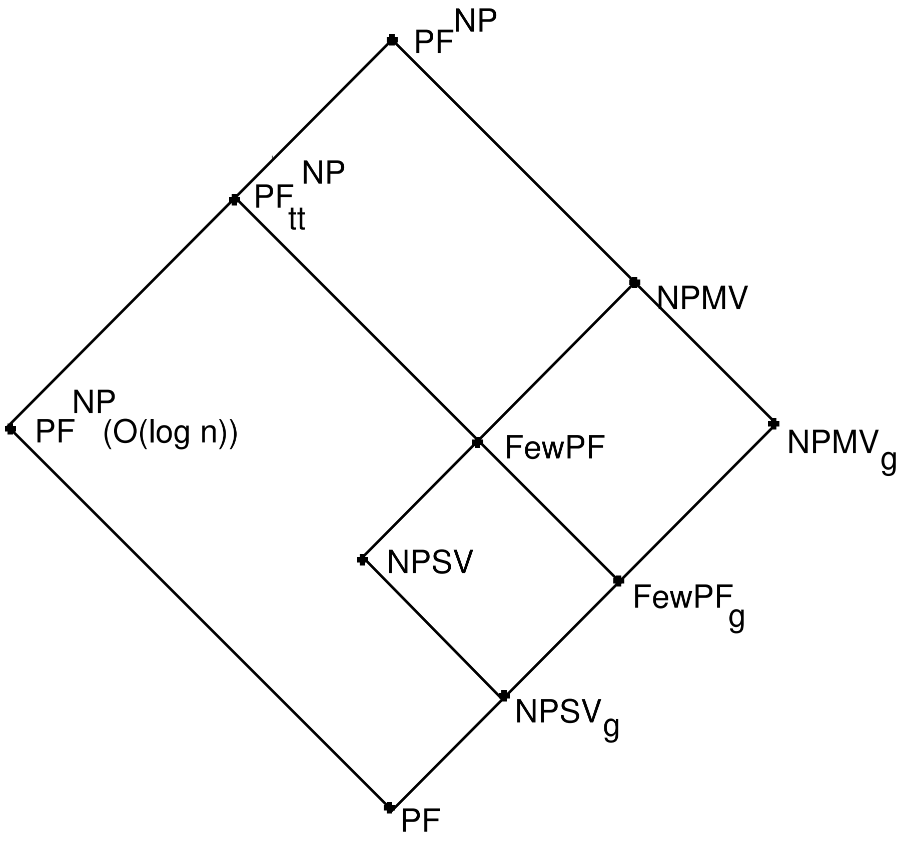
$$\mathrm{P}^{\mathrm{NP}}(O(\log\ n)) = \mathrm{P}^{\mathrm{NP}}_{tt} \subseteq \mathrm{P}^{\mathrm{NP}} \tag{1}$$

are known.

Here we will consider the corresponding class of functions that can be computed nonadaptively with oracles in NP, viz. $\mathrm{PF}^{\mathrm{NP}}_{tt}$.

1.2 Summary of results

Figure 1, "Taxonomy of Function Classes," summarizes all the known inclusions among the classes just described. An inclusion of a class of multivalued functions in a class of single-valued functions is in accordance with the appropriate convention. All of the inclusions indicated are obvious except for FewPF $\subseteq_c$ $\mathrm{PF}_{tt}^{\mathrm{NP}}$.



TAXONOMY OF FUNCTION CLASSES

FIGURE 1

Analogous to the relations in (1), we note the following inclusions:

$$\mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) \subseteq \mathrm{PF}^{\mathrm{NP}}_{tt} \subseteq \mathrm{PF}^{\mathrm{NP}}. \tag{2}$$

Krentel demonstrated that if the outer two function classes are equal, then P = NP. We will show the following concerning the two inner inclusions.

We will show that that $\mathrm{P}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP}}_{tt}$ if and only if $\mathrm{PF}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP}}_{tt}$. I.e., the right two function classes in (2) are equal if and only if all three set classes in (1) are equal. Beigel, Hemachandra, and Wechsung [BHW89] showed that $\mathrm{P}^{\mathrm{NP}}(O(\log\ n)) \subseteq \mathrm{PP}$. Thus, $\mathrm{PF}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP}}_{tt}$ implies $\mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{PP}$. We take this to be evidence that the classes $\mathrm{PF}^{\mathrm{NP}}$ and $\mathrm{PF}^{\mathrm{NP}}_{tt}$ are not identical.

We will show that if $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) = \mathrm{PF}^{\mathrm{NP}}_{tt}$, then FewP = P and R = NP. Thus, the left two function classes in (2) are equal only if complexity classes collapse in an unexpected way. This result stands in direct contrast to the fact that $\mathrm{P}^{\mathrm{NP}}(O(\log\ n)) = \mathrm{P}^{\mathrm{NP}}_{tt}$.

Are the inclusions given in Figure 1 the only ones, and are they strict? Each logical possibility will be examined. Many (such as that demonstrated by Krentel) hold only if P = NP. One of the more interesting results to follow is that $\mathrm{FewPF}_g \subseteq_c \mathrm{PF}$ if and only if FewP $\subseteq$ P. In fact, the result $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) = \mathrm{PF}^{\mathrm{NP}}_{tt}$ implies FewP = P will follow as a corollary of this theorem.

Now let us understand these results in terms of measuring the complexity of inverting one-way functions. For every single-valued class of functions $\mathcal{G}$, it is easy to see that $\mathrm{NPMV}_g \subseteq_c \mathcal{G}$ if and only if every honest polynomial-time computable function can be inverted in $\mathcal{G}$ (or, more precisely, if and only if the inverse of every honest $f \in \mathrm{PF}$ has a refinement $g \in \mathcal{G}$). Furthermore, we will see that for most classes $\mathcal{G}$, $\mathrm{NPMV}_g \subseteq_c \mathcal{G}$ if and only if $\mathrm{NPMV} \subseteq_c \mathcal{G}$. Although we already know that arbitrary honest polynomial-time computable functions can be inverted deterministically in polynomial time (i.e. $\mathrm{NPMV}_g \subseteq_c \mathrm{PF}$) only if P = NP, perhaps every such function can be inverted in some interesting weaker class such as $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$or NPSV? Thus, we see that such questions can be formulated precisely as questions about inclusions between function classes. As an example of this principle, the inverse of every honest $f \in \mathrm{PF}$ has a refinement $g \in \mathrm{NPSV}$ if and only if $\mathrm{NPMV}_g \subseteq_c \mathrm{NPSV}$.

The question of whether $\mathrm{NPMV}_g \subseteq_c \mathrm{NPSV}$ (which is equivalent to $\mathrm{NPMV} \subseteq_c \mathrm{NPSV}$) is particularly interesting. We will show that this relation holds only if there exist disjoint NP-complete sets such that every set that separates them is NP- hard, which contradicts the conjecture studied in [ESY84] and [GS88]. This question is taken up in Section 3.

1.3 Promise problems

Section 3 uses previous work on promise problems. Readers who are unfamiliar with promise problems are referred to [ESY84] and [GS88].

Given multivalued partial functions f and g, define g to be an *extension* of f if $domain(f) \subseteq domain(g)$ and for all $x \in dom(f)$ and all y, if $g(x) \mapsto y$, then $f(x) \mapsto y$. A function g is a *single-valued total extension* of f if g is a single-valued total function and g is an extension of f.

The question of whether a function f has an extension g in a smaller class distinguishes the computational problem of computing values from the domain recognition problem. This approach might be useful in cryptographic applications: Suppose a cryptanalyst has such a $g \in \mathrm{PF}$ which outputs a secret for "good" inputs. The function g is not actually computing f because it is not capable of recognizing "good" inputs, i.e., it cannot determine membership in $dom(f)$. For the cryptanalyst though, this is good enough as this g still breaks the code.

For any single-valued partial function f, (Q_f, R_f) is the promise problem with promise $Q_f = domain(f) \times \Sigma^*$, and property $R_f = \{\langle x, y\rangle \mid x \in domain(f) \text{ and } y \leq f(x)\}$. (Q_f, R_f) is called the promise problem *associated with* f. For convenience, we repeat here the following proposition from [GS88]. This Proposition shows that finding an algorithm to compute a partial function f is equivalent to finding an algorithm to solve (Q_f, R_f)

Proposition 2 [2]

(i) *If f is a single-valued partial function and A is a solution of (Q_f, R_f), then there is a total function h that extends f such that $h \leq_{\mathrm{T}}^{\mathrm{P}} A$.*

[2]Both statements of this proposition in [GS88] include as hypothesis that $f \in \mathrm{NPSV}$, but this is not required.

(ii) *If f is a single-valued partial function and h is a total extension of f, then there is a solution A of (Q_f, R_f) such that $A \leq_{\mathrm{T}}^{\mathrm{P}} h$.*

Proof.

(i) On an input string x, apply a binary search algorithm to find the largest y (whose length is within the right polynomial length of x) such that $(x, y) \in A$. Let $h(x) = y$. If no such y exists, then define $h(x)$ to be λ. Since A is a solution, $y = h(x) = f(x)$ for $x \in dom(f)$.

(ii) Given h, define $A = R_h$, the only solution of (Q_h, R_h). It is easy to see that A is a solution of (Q_f, R_f) and $A \leq_{\mathrm{T}}^{\mathrm{P}} h$. □

2 Inclusion Relations

Given a nondeterministic polynomial-time Turing machine M, let f_M denote the multi-valued function that on input $x \in L(M)$, outputs an accepting computation of M, and that is undefined otherwise. Observe that $f_M \in \mathrm{NPMV}_g$.

Given a set L, let S_L denote the semi-characteristic function of L, ($S_L(x) = 1$, if $x \in L$, and $S_L(x)$ is undefined otherwise), and let C_L denote the characteristic function of L ($C_L(x) = 1$, if $x \in L$, and $C_L(x) = 0$, if $x \in \overline{L}$).

Let $\tau = \langle\ ,\ \rangle$ denote a polynomial time computable pairing function with polynomial time computable inverses σ_1 and σ_2.

2.1 The inclusions indicated in Figure 1 all hold ...

For $f \in NPMV$, let $R_f = \{\langle x, z\rangle \mid \exists y(f(x) \mapsto y \textit{ and } z \leq y)\}$. R_f belongs to NP and it can be used as an oracle in a binary search procedure to find the largest value y such that $f(x) \mapsto y$. Thus, $\mathrm{NPMV} \subseteq \mathrm{PF}^{\mathrm{NP}}$.

It is well-known that $\mathrm{P}^{\mathrm{NP}}(O(\log\ n)) \subseteq \mathrm{P}_{tt}^{\mathrm{NP}}$, and the same proof demonstrates that $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) \subseteq \mathrm{PF}_{tt}^{\mathrm{NP}}$.

Our next goal is to prove that $\mathrm{FewPF} \subseteq \mathrm{PF}_{tt}^{\mathrm{NP}}$. The idea is to encode all information in a FewPF function into an NP oracle. There is a simple intuition that we illustrate first with single-valued functions. This intuition states that a function f is nonadaptively computed (relative to some oracle) if and only if each output bit of the answer $f(x)$ can be determined independently, and nonadaptively (relative to the oracle).

For any single-valued function f, *code*(f) is defined in [Sel78] to be the set of all triples $\langle \sigma, x, k\rangle$, where $\sigma \in \{0,1\}$, $x \in \Sigma^*$ and $k \in \{0,1\}^*$ is the binary representation of a number $n(k)$, such that the following properties hold: $\langle 0, x, k\rangle \in code(f) \leftrightarrow f(x)$ has an $n(k)^{th}$ bit (i.e. $x \in dom(f)$ and $f(x)$ has length $\geq n(k)$), and $\langle 1, x, k\rangle \in code(f) \leftrightarrow$ the $n(k)^{th}$ bit of $f(x)$ is 1. A number of properties of *code*(f) are summarized in Lemma 2.2 of [SXB83], and we repeat them here in the following proposition.

A function f is *polynomial-bounded* if there is a polynomial p such that for all x in the domain of f, $|f(x)| \leq p(|x|)$.

Proposition 3 *Let f be a single-valued function.*

(i) *If f is polynomial-bounded, then $f \leq^{\mathrm{P}}_{tt} code(f)$.*

(ii) *If f is polynomial-bounded and for some A, $code(f) \in \mathrm{P}^A$, then $f \in \mathrm{PF}^A$.*

(iii) *If $f \in \mathrm{PF}^A$, then $code(f) \in \mathrm{P}^A$.*

We illustrate the power of this coding by showing that $\mathrm{P}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP}}_{tt}$ if and only if $\mathrm{PF}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP}}_{tt}$.

Proposition 4 **(i)** *$f \in \mathrm{PF}^{\mathrm{NP}}$ if and only if $code(f) \in \mathrm{P}^{\mathrm{NP}}$.*

(ii) *$f \in \mathrm{PF}^{\mathrm{NP}}_{tt}$ if and only if $code(f) \in \mathrm{P}^{\mathrm{NP}}_{tt}$.*

Proof. Statement (i) from left to right follows from Proposition 3, part (iii). The converse follows from Proposition 3, part (ii). This proves the first assertion.

By Proposition 3, part (i), $f \leq^{\mathrm{P}}_{tt} code(f)$, so $code(f) \leq^{\mathrm{P}}_{tt} SAT$ implies $f \in \mathrm{PF}^{\mathrm{NP}}_{tt}$. Conversely, if $f \in \mathrm{PF}^{\mathrm{NP}}_{tt}$, then $code(f) \in \mathrm{P}^{\mathrm{NP}}_{tt}$ by the obvious algorithm: Given a triple $\langle \sigma, x, k\rangle$ compute $f(x)$ and read the answer. This proves the second assertion. □

Theorem 1 $\mathrm{P}^{\mathrm{NP}} = \mathrm{P}_{tt}^{\mathrm{NP}}$ *if and only if* $\mathrm{PF}^{\mathrm{NP}} = \mathrm{PF}_{tt}^{\mathrm{NP}}$.

Proof. The implication from right to left is obvious. Assume $\mathrm{P}^{\mathrm{NP}} = \mathrm{P}_{tt}^{\mathrm{NP}}$. Let $f \in \mathrm{PF}^{\mathrm{NP}}$. Then, $code(f) \in \mathrm{P}^{\mathrm{NP}}$, by Proposition 3(i). So, $code(f) \in \mathrm{P}_{tt}^{\mathrm{NP}}$, and therefore, by Proposition 3(ii), $f \in \mathrm{PF}_{tt}^{\mathrm{NP}}$. $\square$

Next we extend the definition of $code(f)$ to multivalued functions. For each multivalued function f, define $code'(f)$ to be the set of all tuples $\langle i, j, \sigma, x, k\rangle$, where i, j and k are binary representations of numbers $n(i)$, $n(j)$ and $n(k)$, respectively, and $n(j) \leq n(i)$, such that $\langle i, j, 0, x, k\rangle \in code'(f)$ if there are at least $n(i)$ distinct values of f on x such that the $n(j)$-th value in lexicographic order has an $n(k)$-th bit. And, $\langle i, j, 1, x, k\rangle \in code'(f)$ if there are at least $n(i)$ distinct values of f on x such that the $n(k)$-th bit of the $n(j)$-th value in lexicographic order is a 1. The following proposition follows directly.

Proposition 5 *For each* $f \in \mathrm{FewPF}$, $code'(f) \in \mathrm{NP}$.

Theorem 2 $\mathrm{FewPF} \subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}}$

Proof. Let $f \in \mathrm{FewPF}$. It suffices to show that *all* the values of f on an input word x can be nonadaptively computed in polynomial time from $code'(f)$. First, for each $n(i) \leq p(|x|)$, ask the query $\langle i, i, 0, x, 1\rangle$. The largest value $n(i)$ for which the oracle $code'(f)$ answers “yes” is the exact number of strings in $set\text{-}f(x)$. Once this value is known, then make all queries $\langle i, j, 0, x, k\rangle$ and $\langle i, j, 1, x, k\rangle$ for this value of i and all possible j and k in order to compute $set\text{-}f(x)$. (Note that this is a nonadaptive procedure, since all possible queries can be asked in advance for both stages of the procedure.) $\square$

All other inclusion relations exhibited in Figure 1 are obvious.

2.2 ...and no other inclusions hold

Proposition 6 *If* $\mathcal{G}$ *is a class of functions such that* $h \in \mathcal{G} \to \sigma_2(h) \in \mathcal{G}$, *then* $\mathrm{NPMV}_g \subseteq_c \mathcal{G}$ *implies* $\mathrm{NPMV} \subseteq_c \mathcal{G}$.

Proof. Assume that $\mathcal{G}$ satisfies the hypothesis, and assume that $\mathrm{NPMV}_g \subseteq \mathcal{G}$. Let $f \in \mathrm{NPMV}$, and let M be a nondeterministic transducer that computes f. Define a nondeterministic transducer M' as follows: On input x, M' guesses a string y. If y is an accepting computation of M on input x, then M' outputs the pair $\langle y, z\rangle$, where z is the final contents of M's output tape on computation y.

Let g be the function computed by M', and observe that the graph of g,

$$\begin{aligned}\{\langle x, \langle y, z\rangle\rangle \quad | \quad & y \text{ is an accepting computation of M on input } x \\ & \text{and } z \text{ is the final contents of } M\text{'s output tape on computation } y\},\end{aligned}$$

belongs to P. Thus, $g \in \mathrm{NPMV}_g$, so, by assumption, g has a refinement h that is in $\mathcal{G}$. (For all x, $g(x) \mapsto h(x)$.)

Define $h'(x) = \sigma_2(h(x))$, for all x. Then, h' is a refinement of f, and $h' \in \mathcal{G}$. $\square$

All the classes we consider satisfy the hypothesis of this proposition, so, for example, the following corollary holds. However, this result does not seem to hold for restricted single-valued classes of the form $\mathcal{G}_g$. In particular, $h \in \mathcal{G}_g$ does not necessarily imply $\sigma_2(h) \in \mathcal{G}_g$.

Corollary 1 $\mathrm{NPMV} \subseteq_c \mathrm{NPSV}$ *if and only if* $\mathrm{NPMV}_g \subseteq_c \mathrm{NPSV}$.

Proposition 7 (Krentel) $f \in \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$ *and* $graph(f) \in \mathrm{P}$ *implies* $f \in \mathrm{PF}$.

Though this proposition is not stated in [Kre88], the proof is embedded in the proof of his Theorem 4.1. Namely, there is a P^{SAT} machine M that computes f and that makes at most $O(log\ n)$ queries. Simulate M on input x for all possible oracle answers. This gives a polynomial number of possible output values. A value $y = f(x)$ if and only if $y \in graph(f)$. Since $graph(f) \in \mathrm{P}$, $f \in \mathrm{PF}$.

The following theorem shows that a very large number of possible inclusions hold if and only if P = NP.

Theorem 3 *The following statements are equivalent.*

1. P = NP

2. NPMV $\subseteq_c$ PF

3. NPMV $\subseteq$ NPMV$_g$

4. NPMV $\subseteq_c$ FewPF$_g$

5. NPMV $\subseteq_c$ NPSV$_g$

6. NPMV $\subseteq_c$ PF$^{\rm NP}(O(\log\ n))$

7. FewPF $\subseteq_c$ PF

8. FewPF $\subseteq$ NPMV$_g$

9. FewPF $\subseteq$ FewPF$_g$

10. FewPF $\subseteq_c$ NPSV$_g$

11. NPSV $\subseteq$ PF

12. NPSV $\subseteq$ NPMV$_g$

13. NPSV $\subseteq$ FewPF$_g$

14. NPSV $\subseteq$ NPSV$_g$

15. NPMV$_g$ $\subseteq_c$ PF

16. NPMV$_g$ $\subseteq_c$ PF$^{\rm NP}(O(\log\ n))$

17. PF$^{\rm NP}$ $\subseteq$ PF

18. PF$^{\rm NP}$ $\subseteq$ NPMV$_g$

19. PF$^{\rm NP}$ $\subseteq$ FewPF$_g$

20. PF$^{\rm NP}$ $\subseteq$ NPSV$_g$

21. PF$^{\rm NP}$ $\subseteq$ PF$^{\rm NP}(O(\log\ n))$

22. $\mathrm{PF}_{tt}^{\mathrm{NP}} \subseteq \mathrm{PF}$

23. $\mathrm{PF}_{tt}^{\mathrm{NP}} \subseteq \mathrm{NPMV}_g$

24. $\mathrm{PF}_{tt}^{\mathrm{NP}} \subseteq \mathrm{FewPF}_g$

25. $\mathrm{PF}_{tt}^{\mathrm{NP}} \subseteq \mathrm{NPSV}_g$

26. $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) \subseteq \mathrm{PF}$

27. $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) \subseteq \mathrm{NPMV}_g$

28. $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) \subseteq \mathrm{FewPF}_g$

29. $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) \subseteq \mathrm{NPSV}_g$

Proof sketch. Most items are proved readily from Propositions 1, 6, and 7. We will illustrate the proof of key exceptions and leave remaining cases to the reader.

Item 5: To see that NPMV $\subseteq_c$ NPSV_g implies P = NP, observe that if the graph of S_L, $\{\langle x, 1\rangle \mid x \in L\}$, belongs to P, then obviously $L \in \mathrm{P}$ as well.

Item 15: To see that $\mathrm{NPMV}_g \subseteq_c$ PF implies P = NP, let L be an NP-complete set and let M be an NP-acceptor for L. Then, by hypothesis, the function $f_L \in \mathrm{PF}$. So, $L \in \mathrm{P}$ follows immediately. □

Another sizeable number of possible inclusions hold if and only if NP = co-NP.

Theorem 4 *The following statements are equivalent.*

1. NP = co-NP

2. $\mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{NP}$

3. $\mathrm{P}_{tt}^{\mathrm{NP}} \subseteq \mathrm{NP}$

4. $\mathrm{PF}^{\mathrm{NP}} \subseteq \mathrm{NPSV}$

5. $\mathrm{PF}^{\mathrm{NP}} \subseteq \mathrm{FewPF}$

6. $\mathrm{PF}^{\mathrm{NP}} \subseteq \mathrm{NPMV}$

7. $\mathrm{PF}^{\mathrm{NP}}_{tt} \subseteq \mathrm{NPSV}$

8. $\mathrm{PF}^{\mathrm{NP}}_{tt} \subseteq \mathrm{FewPF}$

9. $\mathrm{PF}^{\mathrm{NP}}_{tt} \subseteq \mathrm{NPMV}$

10. $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) \subseteq \mathrm{NPSV}$

11. $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) \subseteq \mathrm{FewPF}$

12. $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) \subseteq \mathrm{NPMV}$

Proof. It is well-known that NP = co-NP implies $\mathrm{P}^{\mathrm{NP}} \subseteq \mathrm{NP}$. (If NP = co-NP, then the NP oracle can be replaced by NP-acceptors for the oracle and for its complement.) Thus, items 1 - 3 are seen to be equivalent. Similarly, NP = co-NP implies $\mathrm{PF}^{\mathrm{NP}} \subseteq$ NPSV, which in turn implies items 5 and 6. To see that item 6 implies NP = co-NP, consider the characteristic function of any NP-complete set. The remaining cases are proved similarly. □

2.2.1 UP, FewP, and FEW

Some of the remaining possibilities are equivalent to either P = UP or P = FewP. It is proved in [GS88] that P = UP if an only if $\mathrm{NPSV}_g \subseteq \mathrm{PF}$. Thus, using Proposition 7 as well, we have the following theorem.

Theorem 5 *The following statements are equivalent.*

1. $\mathrm{P} = \mathrm{UP}$

2. $\mathrm{NPSV}_g \subseteq \mathrm{PF}$

3. $\mathrm{NPSV}_g \subseteq \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$

Corollary 2 NPSV $\subseteq$ PF$^{\mathrm{NP}}(O(\log\ n))$ *implies* P = UP.

Theorem 6 NPMV$_g \subseteq_c$ NPSV$_g$ *implies* NP = UP.

Proof. It is proved in [GS88] that NP = UP if and only if there is a function in NPSV$_g$ whose domain is NP-complete. Let L be any NP-complete set and let M be an NP-acceptor for L. The function f_M belongs to NPMV$_g$, so, by hypothesis, there is a refinement h of f_M that belongs to NPSV$_g$. Furthermore, $dom(h) = L$ is NP-complete. $\Box$

We should not inquire whether NPMV $\subseteq$ FewPF. For one thing, the set-theoretic inclusion is false, for it is well-known that formulas of propositional logic can have exponentially many satisfying assignments. Thus, NPMV $\not\subseteq$ FewPF, NPMV$_g \not\subseteq$ FewPF, and NPMV$_g \not\subseteq$ FewPF$_g$. Furthermore, the inquiry is meaningless because Figure 1 collapses to a point if and only if P = NP. Thus, these classes cannot separate trivially. The correct questions for us to raise are whether NPMV $\subseteq_c$ FewPF, whether NPMV$_g \subseteq_c$ FewPF, and whether NPMV$_g \subseteq_c$ FewPF$_g$. The following theorem addresses one of these questions.

Theorem 7 NPMV$_g \subseteq_c$ FewPF$_g$ *implies* NP = FewP.

Proof. Let L be an NP-complete set and let M be an NP-acceptor for L. The function f_M belongs to NPMV$_g$, so, by hypothesis, there is a refinement h of f_M that belongs to FewPF$_g$. Furthermore, $dom(h) = L$ is NP-complete. Since $x \in L \leftrightarrow \exists y \langle x, y\rangle \in graph(h)$ and $graph(h) \in$ P, $L \in$ FewP follows. $\Box$

Similar to Theorem 5, we equate the function class collapse of FewPF$_g$ to PF with the set class collapse of FewP to P.

Theorem 8 FewPF$_g \subseteq_c$ PF *if and only if* FewP = P.

Proof. The proof from left to right is straightforward. Assume FewP = P. Let $f \in$ FewPF$_g$, let M be a nondeterministic transducer that computes f in time q, and let p bound $\|set\text{-}f\|$.

Consider the language

$$
\begin{aligned}
L &= \{(x, c(F), u) \mid F \text{ is a finite set and there exists} \\
&\qquad w \notin F \text{ such that } u \text{ is a prefix of } w \text{ and } f(x) \mapsto w\}.
\end{aligned}
$$

We claim that $L \in$ FewP: Given x, F, and u, guess a string w and check whether $w \notin F$, u is a prefix of w, and $(x, w) \in \mathit{graph}(f)$. The number of correct guesses is at most $p(|x|)$.

Since FewP $=$ P is assumed, $L \in$ P. For each x, the following algorithm uses L to compute c(set-$f(x)$) in polynomial time. The basic idea is to maintain F as a subset of c(set-$f(x)$). Use L to determine whether there exists a value of $f(x)$ that does not belong to F; if so, use L to find such a value w by implementing a typical prefix search, and then increment F to contain w.

```
begin
input x;
F := ∅;
while (x, c(F), λ) ∈ L do
    begin { F is a proper subset of set-f(x) }
    u := λ;
    while ((x, u) ∉ graph(f) ∨ u ∈ F) do
          if (x, c(F), u0) ∈ L
             then u := u0
             else if (x, c(F), u1) ∈ L
                then u := u1;
    { f(x) ↦ u ∧ u ∉ F }
    F := F ∪ [u];
    end
halt in an accepting state with c(F) on the output tape;
end.
```

When execution of the outer while-loop terminates, $F = \mathit{set}$-$f(x)$. To see this, note that the inner while-loop is reached only if there is a string $y \in \mathit{set}$-$f(x)$ that has not yet been found and that the inner while-loop preserves this property. In particular, the inner while-loop terminates only when a string u is found such that $f(x) \mapsto u$ and $u \notin F$. This condition enures that if $f(x) \mapsto w_1$ and $f(x) \mapsto w_2$, where w_1 is a prefix of w_2, then both w_1 and w_2 are eventually placed into F.

Let us observe that the procedure runs in polynomial time. Since $L \in \mathrm{P}$ and $graph(f) \in \mathrm{P}$, each test takes polynomial time. The outer while-loop is executed at most $p(|x|)$ times, and, for each execution of the outer loop, the inner while-loop executes at most $q(|x|)$ times. Thus, we conclude that $\mathrm{c}(set\text{-}f) \in \mathrm{PF}$. $\square$

Theorem 9 $\mathrm{FewPF}_g \subseteq_c \mathrm{NPSV}_g$ *implies* $\mathrm{FewP} = \mathrm{UP}$.

Proof. Let $L \in \mathrm{FewP}$, and let M be a FewP-acceptor for L. The function $f_M \in \mathrm{FewPF}_g$, so, by hypothesis, $c(set\text{-}f) \in \mathrm{NPSV}_g$. Then, $L \in \mathrm{UP}$ by the following algorithm: On input x, guess a string y and check whether $\langle x, y\rangle \in graph(c(set\text{-}f))$. If so, then accept, for in this case $x \in L$. $\square$

For each nondeterministic polynomial time-bounded Turing machine M, define $count_M$ so that for all strings x, $count_M(x)$ is the binary representation of the number of accepting computations of M on input x.

Lemma 1 *If there is a polynomial q such that for all x, $count_M(x) \leq q(|x|)$, then $count_M \in \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$.*

Proof. Define $CS_M = \{(x, k) \mid k \leq count_M(x)\}$. Clearly, $CS_M \in \mathrm{NP}$. Then, a binary search algorithm shows that $count_M \in \mathrm{P}^{\mathrm{NP}}(O(\log\ n))$. $\square$

The class FEW is defined by Cai and Hemachandra [CH90] to be an extension of UP and FewP. FEW is the class of languages L such that there is a nondeterministic polynomial time Turing machine M, a polynomial time predicate Q, and a polynomial q such that for all x,

$$count_M(x) \leq q(|x|)$$

and

$$x \in L \leftrightarrow Q(x, count_M(x)).$$

Theorem 10 FEW $\subseteq \mathrm{P}^{\mathrm{NP}}(O(\log\ n))$.

The proof follows immediately from the lemma.

Theorem 11 $\mathrm{FewPF}_g \subseteq_c \mathrm{PF}$ *implies* FEW = P.

The proof is straightforward. If $L \in$ FEW, and M is the machine given in the definition, then the multivalued function f_M that on an input x outputs any accepting computation of M on x is in FewPF_g. Thus, by assumption, all accepting computations can be found, and therefore $count_M$ can be found.

Corollary 3 *The following statements are equivalent.*

1. P = FewP
2. P = FEW
3. $\mathrm{FewPF}_g \subseteq_c \mathrm{PF}$
4. $\mathrm{FewPF}_g \subseteq_c \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$

The equivalence of statement (4) with the other assertions follows from Proposition 7. It is only necessary to observe that $graph(f) \in \mathrm{P}$ implies $graph(set\text{-}f) \in \mathrm{P}$.

Corollary 4 **(i)** $\mathrm{PF}^{\mathrm{NP}}_{tt} \subseteq \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$ *implies* P = FewP.

(ii) $\mathrm{FewPF} \subseteq \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$ *implies* P = FewP.

Proof. By Theorem 2, $\mathrm{FewPF} \subseteq \mathrm{PF}^{\mathrm{NP}}_{tt}$. Thus, $\mathrm{PF}^{\mathrm{NP}}_{tt} \subseteq \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$ implies P = FewP, by the previous corollary. □

Let $SAT1$ denote the set of formulas of propositional logic that contain at most one satisfying assignment. Valiant and Vazirani [VV85] showed that R = NP if the promise problem $(SAT1, SAT)$ has a solution in P. We use their result in order to prove the following theorem. The proof of this theorem is inspired by, but easier than, a result in [Tod91].

Theorem 12 $\mathrm{PF}_{tt}^{\mathrm{NP}} = \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$ *implies* $\mathrm{R} = \mathrm{NP}$.

Proof. Define

$$SAT' = \{(\phi, i) \mid \phi \text{ has } n \text{ variables, } n \geq i, \text{ and there is a satisfying assignment } w \text{ of } \phi \text{ in which the } i\text{-th variable is true}\}.$$

Clearly, $SAT' \in \mathrm{NP}$.

Define $cand(\phi) = SAT'((\phi, 1))SAT'((\phi, 2)) \cdots SAT'((\phi, n))$. (One might think of $cand$ as a candidate for a satisfying assignment of ϕ. Of course, in general it is unlikely.) Clearly, $cand \in \mathrm{PF}_{tt}^{\mathrm{NP}}$. Thus, by hypothesis, $cand \in \mathrm{PF}^{\mathrm{NP}}(O(\log\ n))$. Let M be a P^{SAT} machine that computes $cand$ and that makes at most $O(log\ n)$ queries.

Let $L = \{\phi \mid$ some output of M on input ϕ is a satisfying assignment$\}$. Clearly, $L \in$ P. Also, L is a solution of $(SAT1, SAT)$. Thus, NP $=$ R follows from the result of Valiant and Vazirani. $\square$

Corollary 4 and Theorem 12 are particularly interesting in light of the fact that the corresponding classes of sets $\mathrm{P}_{tt}^{\mathrm{NP}}$ and $\mathrm{P}^{\mathrm{NP}}(O(\log\ n))$ are known to be equal.

2.2.2 Taking stock—and the difficulty of inverting functions

A large number of possible inclusions have now been shown to be equivalent to or to imply complexity class collapses that are considered unlikely.

Only the following possibilities have not been considered yet:

1. $\mathrm{NPMV}_g \subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}}$ (equivalent to $\mathrm{NPMV} \subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}}$)

2. $\mathrm{NPMV}_g \subseteq_c \mathrm{FewPF}$ (equivalent to $\mathrm{NPMV} \subseteq_c \mathrm{FewPF}$)

3. $\mathrm{NPMV}_g \subseteq_c \mathrm{NPSV}$ (equivalent to $\mathrm{NPMV} \subseteq_c \mathrm{NPSV}$)

4. FewPF $\subseteq_c$ NPSV

5. $\mathrm{FewPF}_g \subseteq_c$ NPSV

The next proposition helps to explain why the questions on this list are interesting. Recall that the inverse f^{-1} of every honest function $f \in PF$ is a multivalued function in NPMV_g. Thus, the difficulty of inverting f is the complexity of the single-valued refinements of f^{-1}. We say that a function f *is invertible in class* $\mathcal{C}$ if f^{-1} has a single-valued refinement in $\mathcal{C}$. For example, f is invertible in polynomial time if f^{-1} has a single-valued refinement in PF.

Every honest function in PF is invertible in $\mathrm{PF}^{\mathrm{NP}}$. Here we question whether every honest function in PF is invertible in some class that is smaller than $\mathrm{PF}^{\mathrm{NP}}$.

Define a single-valued function f to be *few-one* if there is a polynomial p such that for each $y \in range(f)$, $||\{x \in domain(f) \mid f(x) = y\}|| \leq p(|y|)$.

Proposition 8 *Let $\mathcal{C}$ be any class of single-valued functions such that $f \in \mathcal{C}$ implies $\sigma_2(f) \in \mathcal{C}$. Then, every honest (one-one, few-one) polynomial time computable function is invertible in class $\mathcal{C}$ if and only if* $\mathrm{NPMV}_g \subseteq \mathcal{C}$ *(*$\mathrm{NPSV}_g \subseteq \mathcal{C}$, $\mathrm{FewPF}_g \subseteq \mathcal{C}$, *respectively).*

Proof. Let f be an honest function in PF. Then $f^{-1} \in \mathrm{NPMV}_g$. If, f is one-one, then $f^{-1} \in \mathrm{NPSV}_g$, and if f is few-one, then $f^{-1} \in \mathrm{FewPF}_g$. Thus, if f^{-1} does not have a refinement in $\mathcal{C}$, then neither does the respective class.

Assume that every honest polynomial time computable function is invertible in class $\mathcal{C}$. Let $f \in \mathrm{NPMV}_g$. Define a function h by $h(x,y) = y$, if $f(x) \mapsto y$, and $h(x,y)$ is undefined, otherwise. $h(x,y) = y \leftrightarrow \langle x,y\rangle \in graph(f)$. So, h is an honest function in PF. Then, h is invertible in the class $\mathcal{C}$. That is, there is a function $s \in \mathcal{C}$ such that for all $y \in range(h)$, $s(y) = \langle x,y\rangle$, where $f(x) \mapsto y$. Define $\sigma'(x) = \sigma_2(s(y)) = \sigma_2(\langle x,y\rangle) = y$. Then, σ' is a refinement of f which belongs to $\mathcal{C}$. Furthermore, if $f \in \mathrm{NPSV}_g$, then σ' is one-one, and if $f \in \mathrm{FewPF}_g$, then σ' is few-one. □

The following examples illustrate application of this Proposition:

Example 1 *Every honest polynomial time computable function is invertible in the class* PF

$$\begin{aligned} &\leftrightarrow \quad \mathrm{NPMV}_g \subseteq_c \mathrm{PF} \\ &\leftrightarrow \quad \mathrm{NPMV} \subseteq_c \mathrm{PF} \\ &\leftrightarrow \quad \mathrm{P} = \mathrm{NP}. \end{aligned}$$

Example 2 ([GS88]) *Every honest one-one polynomial time computable function is invertible in the class* PF

$$\begin{aligned} &\leftrightarrow \quad \mathrm{NPSV}_g \subseteq_c \mathrm{PF} \\ &\leftrightarrow \quad \mathrm{P} = \mathrm{UP}. \end{aligned}$$

Example 3 *Every honest few-one polynomial time computable function is invertible in the class* PF

$$\begin{aligned} &\leftrightarrow \quad \mathrm{FewPF}_g \subseteq_c \mathrm{PF} \\ &\leftrightarrow \quad \mathrm{P} = \mathrm{FewP}. \qquad (\textit{by Theorem 8}). \end{aligned}$$

Example 4 *Every honest polynomial time computable function is invertible in the class* $\mathrm{PF}_{tt}^{\mathrm{NP}}$

$$\begin{aligned} &\leftrightarrow \quad \mathrm{NPMV}_g \subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}} \\ &\leftrightarrow \quad \mathrm{NPMV} \subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}}. \end{aligned}$$

Example 5 *Every honest polynomial time computable function is invertible in the class* NPSV

$$\begin{aligned} &\leftrightarrow \quad \mathrm{NPMV}_g \subseteq_c \mathrm{NPSV} \\ &\leftrightarrow \quad \mathrm{NPSV} \subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}}. \end{aligned}$$

The question of whether $\mathrm{NPMV}_g \subseteq_c \mathrm{PF}_{tt}^{\mathrm{NP}}$ is studied in [WT], where it is shown that this assertion holds relative to a set of oracles of Lebesgue measure one.

The question of whether $\mathrm{NPMV}_g \subseteq_c \mathrm{NPSV}$ is studied in greater detail in the next section.

We leave items 2, 4 and 5, whether $\mathrm{NPMV}_g \subseteq_c \mathrm{FewPF}$, whether $\mathrm{FewPF} \subseteq_c \mathrm{NPSV}$, and whether $\mathrm{FewPF}_g \subseteq_c \mathrm{NPSV}$, as open questions. It might be worth observing that $\mathrm{FewPF} \subseteq_c \mathrm{NPSV}$ implies $\mathrm{FEW} \subseteq \mathrm{NP}$, and the latter is probably false.

Let $\mathcal{C}$ be a class of single-valued functions and let $f \in \mathrm{FewPF}$. Before completing this summary, let us reconsider the meaning of $f \in_c \mathcal{C}$. We defined this to mean that $c(\textit{set-}f) \in \mathcal{C}$—all the values of $f(x)$ can be found within C. The definition for multivalued functions f in general is that some refinement of f belongs to $\mathcal{C}$—one value of $f(x)$ can be found within C. An advantage of the stronger form is that stronger results are obtainable; eg., Theorem 2 is a stronger statement than it would be otherwise. Are the two versions equivalent? In fact, if for all $f \in \mathrm{FewPF}$, some single-valued refinement of f belongs to PF, then $c(\textit{set-}f) \in \mathrm{PF}$ follows by an easy iteration. An iterative procedure along the lines of "Oracle procedure 3.1" in [BLS84][3] accomplishes this. Briefly, at each step of the iteration, values of $f(x)$ that the refinement has already found are deleted, until the range of $f(x)$ becomes empty. Then, the list of values that has been found is output. However, if we replace PF by NPSV, then this algorithm does not seem to work because we cannot detect termination within NP (unless NP = co-NP). Thus, we can formulate two versions of the open problem $\mathrm{FewPF} \subseteq_c \mathrm{NPSV}$—one in accordance to the convention established for FewPF, and the other in accordance to the convention established for multivalued functions in general.

3 Can nondeterminism be guided?

Now we take up the question $\mathrm{NPMV} \subseteq_c \mathrm{NPSV}$ in greater detail. The first theorem shows that it is sufficient to raise the question for satisfiability. Let SAT denote the NP-complete satisfiability problem. Let sat be the multivalued function whose value on input a formula of propositional logic x is any satisfying assignment of x, if $x \in \mathrm{SAT}$,

[3]The class FewPF is called NPMVPB in [BLS84].

and which is undefined, otherwise.

Theorem 13 NPMV $\subseteq_c$ NPSV *if and only if there is a refinement h in* NPSV *of sat.*

Proof. The theorem only requires proof in one direction. Assume h is a refinement in NPSV of *sat*. Let $f \in$ NPMV and let T be a nondeterministic polynomial time transducer that computes f. By Cook's theorem [Coo71], for each string x, there is a formula ϕ_x, obtainable in polynomial time, such that the satisfying assignments of ϕ_x are encodings of accepting computations of T on input x. Let $\mathcal{O}$ be defined so that if y is an encoding of an accepting computation, then $\mathcal{O}(y)$ is the final value of the output tape. Obviously, $\mathcal{O} \in$ PF. Define $H(\phi) = \mathcal{O}(h(\phi))$, for each formula ϕ. Then, define $f_1(x) = H(\phi_x)$, for each x. f_1 is a refinement in NPSV of f. $\square$

Define a partial function f to be NP-*hard* if for every single-valued total extension g of f, SAT $\leq^{\mathrm{P}}_{\mathrm{T}} g$.

Theorem 14 *If* NPMV $\subseteq_c$ NPSV*, then* NPSV *contains an* NP-*hard partial function.*

The proof is straightforward. Let h be a refinement of *sat* in NPSV. To see that every single-valued total extension g of h is NP- hard, observe that

$x \in \mathrm{SAT} \leftrightarrow g(x)$ is a satisfying assignment for x.

A conjecture is raised in [ESY84] which states that there exists no promise problem (Q, R) such that

(i) $Q \in$ NP,

(ii) $(Q, R) \in$ NPP $\cap$ co-NPP, and

(iii) (O, R) is NP-hard.

This conjecture implies that NP-hard public key cryptosystems do not exist, that NP $\neq$ co-NP, and, as is proved in [ESY84], that UP $\neq$ NP.

Theorem 15 *The following statements are equivalent.*

1. NPSV *contains an* NP*-hard partial function.*

2. *There is a function* $f \in$ NPSV *such that* (Q_f, R_f) *is* NP*-hard (i.e. every solution is* NP*-hard).*

3. *There is a promise problem* (Q, R) *such that* $Q \in$ NP*,* $(Q, R) \in$ NPP $\cap$ co-NPP*, and* (Q, R) *is* NP*-hard.*

4. *There exist disjoint* $\leq_{\mathrm{T}}^{\mathrm{P}}$*-complete sets* A *and* B *in* NP *such that every set that separates them is* NP*-hard.*

Proof. To see that statement 1 implies statement 2, let f be an NP-hard function in NPSV, and let A be a solution of (Q_f, R_f). By Proposition 2, there is a total extension h of f such that $h \leq_{\mathrm{T}}^{\mathrm{P}} A$. Thus, A is NP-hard.

That statement 2 implies 3, and that 3 implies 4, are demonstrated in [GS88].

To see that statement 4 implies 1, let A and B be disjoint $\leq_{\mathrm{T}}^{\mathrm{P}}$-complete sets such that every set that separates them is NP- hard. Define

$$f(x) = \begin{cases} 0 & x \in A \\ 1 & x \in B \\ \uparrow & \text{otherwise.} \end{cases}$$

Clearly, $f \in$ NPSV. Let h be any total extension of f, and define $L = \{x \mid h(x) = 0\}$. Then, $A \subseteq L$ and $B \subseteq \overline{L}$, so L separates A and B. Therefore, by assumption, L is NP-hard. However, $L \leq_{\mathrm{T}}^{\mathrm{P}} h$. Therefore, h is NP-hard. Since h is an arbitrary total extension of f, it follows that f is NP- hard. □

Corollary 5 *If the conjecture stated above is true, then* NPMV $\not\subseteq_c$ NPSV*.*

3.1 Reductions between partial functions

We have been considering NP-hard partial functions. Here we define the reducibility, polynomial time Turing reductions between partial functions, that underlies this hardness notion. The definition is not obvious, for one must decide what to do in case a query is made to the oracle function when the query is not in the domain of the oracle function. We will start from first principles; then, for single-valued functions, we will prove that a function f is reducible to a function g if and only if there is a reduction from (Q_f, R_f), the promise problems associated with f, to (Q_g, R_g), the promise problem associated with g.

Let M be a deterministic oracle transducer. (That is, M is a deterministic multitape Turing machine with a read-only input tape, a write-only output tape, a write-only oracle query tape, and a read- only oracle answer tape.) We assume that M has designated accepting states in the usual manner and that M is polynomial time-bounded.

If g is a single-valued total function, then we define $M[g]$, the single-valued partial function computed by M with oracle g as follows: $x \in \mathit{domain}(M[g])$ if and only if M reaches an accepting state on input x. In this case, $M[g](x)$ is the final value of M's output tape.

Definition 1 *Let f and g be multivalued partial functions. f* is Turing reducible to *g* in polynomial time*, $f \leq_{\mathrm{T}}^{\mathrm{P}} g$, if for some deterministic oracle transducer M, for every single-valued total extension g' of g, $M[g']$ is an extension of f.*

Note that we do not require the oracle to "know" when a query x is not in the domain of g. In this case, our approach allows the oracle to provide an arbitrary answer. We require only that the result of the computation does not depend on the value of such answers. Note also that M is not required to "know" the domain of f, for M is only required to compute extensions of f.

Define a partial function g to be *hard* for a class of partial functions $\mathcal{C}$ if $f \in \mathcal{C}$ implies $f \leq_{\mathrm{T}}^{\mathrm{P}} g$. The following theorem demonstrates that the reducibility we have been defined yields the desired definition of NPMV-hard.

Lemma 2 *The function sat is hard for* NPMV*.*

Proof. Let h be any single-valued total extension of sat. The proof is identical to the proof of Theorem 13. Let $f \in \mathrm{NPMV}$ and let T be a nondeterministic polynomial time transducer that computes f. This time notice that the function f_1 that is defined in the proof of Theorem 13 is an extension of f and that $f_1 = M[h]$, where M is the transducer specified by the algorithm given in the proof of theorem 12. □

Theorem 16 *The following statements are equivalent.*

1. *f is* NP*-hard.*

2. *$sat \leq^{\mathrm{P}}_{\mathrm{T}} f$.*

3. *f is hard for* NPMV*.*

Proof. Suppose that f is NP-hard and let f' be any single-valued total extension of f. It is well-known that self-reducibility of SAT can be used to find a satisfying assignment of any satisfiable formula. In order to prove that 1 implies 2, implement this reduction but replace SAT with a simulation of the reduction from SAT to the single-valued total extension f'. The result is a reduction from an extension of sat to f'.

The Lemma show that 2 implies 3. In fact, 3 obviously implies 2 since $sat \in \mathrm{NPMV}$.

Assume $sat \leq^{\mathrm{P}}_{\mathrm{T}} f$ is witnessed by oracle transducer M. If f' is any single-valued total extension of f, then $M[f']$ is an extension of sat. Thus, $x \in \mathrm{SAT} \leftrightarrow M[f'](x)$ is an satisfying assignment of x. Thus, 3 implies 1. □

Theorem 17 *If f and g are single-valued partial functions, then the following statements are equivalent.*

1. *$f \leq^{\mathrm{P}}_{\mathrm{T}} g$.*

2. *For every total extension g' of g, there is a total extension f' of f such that $f' \leq^{\mathrm{P}}_{\mathrm{T}} g'$.*

3. *$(Q_f, R_f) \leq^{\mathrm{PP}}_{\mathrm{T}} (Q_g, R_g)$.*

4. $(Q_f, R_f) \leq^{\mathrm{PP}}_{\mathrm{UT}} (Q_g, R_g)$*.*

5. There is polynomial time oracle transducer M such that for every total extension g' of g, $M[g']$ is a total extension of f.

Proof. Assume $f \leq^{\mathrm{P}}_{\mathrm{T}} g$ and let g' be a total extension of g. Then, define M' so that M' simulates M with oracle g', except that if M halts without accepting its input, then M' accepts and outputs λ. In this way, M' computes a total extension f' of f such that $f' \leq^{\mathrm{P}}_{\mathrm{T}} g'$.

To see that statement 2 implies statement 3, let L_1 be a solution of (Q_g, R_g). By Proposition 2, there is a total extension g_1 of g such that $g_1 \leq^{\mathrm{P}}_{\mathrm{T}} L_1$. Assuming statement 2, there is a total extension f_1 of f such that $f' \leq^{\mathrm{P}}_{\mathrm{T}} g'$. Again by Proposition 2, there is a solution A of (Q_f, R_f) such that $A \leq^{\mathrm{P}}_{\mathrm{T}} f_1$. By transitivity, $A \leq^{\mathrm{P}}_{\mathrm{T}} L_1$ follows, and this is what we needed to prove.

$(Q_f, R_f) \leq^{\mathrm{PP}}_{\mathrm{T}} (Q_g, R_g)$ implies $(Q_f, R_f) \leq^{\mathrm{PP}}_{\mathrm{UT}} (Q_g, R_g)$ is one of the main results proved in [GS88].

Assume $(Q_f, R_f) \leq^{\mathrm{PP}}_{\mathrm{UT}} (Q_g, R_g)$, and let M be a Turing machine that witnesses the uniform reduction. The general idea is this: Let g' be a total extension of g. By part (ii) of Proposition 2, there is a solution L_1 of (Q_g, R_g) such that $L_1 \leq^{\mathrm{P}}_{\mathrm{T}} g'$. $L[M]$ is a solution of (Q_f, R_f), and, by part (i) of Proposition (1), there is a total extension f_1 of f such that $f_1 \leq^{\mathrm{P}}_{\mathrm{T}} L[M]$. Thus, $f_1 \leq^{\mathrm{P}}_{\mathrm{T}} g_1$. We need to show that the resulting reduction is uniform. This is easy to do by including the routines given in the proof of Propositions 2. Namely, define M' to be an oracle transducer that when operated with oracle function g' behaves as follows: On an input word x, M' simulates the algorithm given in the proof of part (i) of Proposition 2, but when this algorithm queries its oracle, M' simulates M instead. Furthermore, when M' is simulating M, M' uses the oracle g' to simulate queries to $R_{g'}$, as in the proof of part (ii) of Proposition 2.

It is obvious that statement 4 implies statement 1. $\square$

4 Acknowledgements

The author originally proved that $\mathrm{PF}^{\mathrm{NP}}(O(\log\ n)) = \mathrm{PF}_{tt}^{\mathrm{NP}}$ implies UP = P. Richard Beigel identified a similarity between this result and results in [Bei] and [Tod91], and for this reason suggested that R = NP might be obtainable also (cf. Theorem 11.) The extension from UP = P to FEW = P (Corollary 3 (i)) is due to Lane Hemachandra.

The author benefited from many conversations about this program with Steve Homer, Andy Klapper, Luc Longpré, and Ken Regan.

References

[AR88] E. Allender and R. Rubinstein. P-printable sets. *SIAM J. Comput.*, 17(6), 1988.

[Bei] R. Beigel. NP-hard sets are P-superterse unless R = NP. Technical Report 88-04, Department of Computer Science, The Johns Hopkins University.

[BH88] S. Buss and L. Hay. On truth table reducibility to sat and the difference hierarchy over NP. In *Procs. of the 3rd Annual Conference on Structure in Complexity Theory*, pages 224–233. IEEE Computer Society Press, 1988.

[BHW89] R. Beigel, L. Hemachandra, and G. Wechsung. On the power of probabilistic polynomial time. In *Proc. Fourth Annual IEEE Structure in Complexity Theory Conference*, pages 225–227, 1989.

[BLS84] R. Book, T. Long, and A. Selman. Quantitative relativizations of complexity classes. *SIAM J. Comput.*, 13(3):461–487, August 1984.

[CH90] J. Cai and L. Hemachandra. On the power of parity polynomial time. *Math. Systems Theory*, 1990. In press.

[Coo71] S. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symp. Theory of Computing*, pages 151–158, 1971.

[ESY84] S. Even, A. Selman, and Y. Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information And Control*, 61(2):159–173, May 1984.

[GS88] J. Grollmann and A. Selman. Complexity measures for public-key cryptosystems. *SIAM J. Comput.*, 17(2), April 1988.

[Hem87] L. Hemachandra. *Counting in Structural Complexity Theory*. PhD thesis, Cornell University, 1987.

[Kad87] J. Kadin. $\mathrm{p}^{\mathrm{np}}[\log n]$ and sparse Turing-complete sets for NP. In *Proc. Struct. in Complexity Second Annual Conference*, pages 33–40, 1987.

[Kre88] M. Krentel. The complexity of optimization problems. *J. Computer Systems Sci.*, 36:490–509, 1988.

[KSW86] J. Köbler, U. Schöning, and K. Wagner. The difference and truth-table hierarchies for NP. In *Theoretical Informatics and Applications*, 1986.

[Sel78] A. Selman. Polynomial time enumeration reducibility. *SIAM J. Comput.*, 7(4):440–457, November 1978.

[Sel88] A. Selman. Natural self-reducible sets. *SIAM J. Comput.*, 17:989–996, 1988.

[Sel90] A. Selman. One-way functions in complexity theory. In *Proc. Symp. on Math. Foundations of Computer Science, Lecture Notes in Computer Science*, volume 452, pages 88–104, Berlin, 1990. Springer-Verlag.

[SXB83] A. Selman, Xu M.-R., and R. Book. Positive relativizations of complexity classes. *SIAM J. Comput.*, 12:465–479, 1983.

[Tod89] S. Toda. On the computational power of PP and $\oplus$P. In *Proc. 30th IEEE Symp. on Foundations of Computer Science*, pages 514–519, 1989.

[Tod91] S. Toda. On polynomial-time truth-table reducibilities of intractable sets to P-selective sets. *Mathematical Systems Theory*, 24(2):69–82, 1991.

[Val76] L. Valiant. Relative complexity of checking and evaluating. *Information Processing Letters*, 5(1):20–23, May 1976.

[VV85] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. In *Proc. 17th ACM Symp. Theory of Computing*, pages 458–463, 1985.

[Waga] K. Wagner. Bounded query classes. Technical Report TR-157, Universitat Ausburg, Institut fur Mathematik, 1987.

[Wagb] K. Wagner. Log query classes. Technical Report TR-145, Universitat Ausburg, Institut fur Mathematik, 1987.

[WT] O. Watanabe and S. Toda. Some structural analyses on the complexity of inverse functions. manuscript, 1991.