Theoretical Computer Science 1 (1975) 103–123.

A COMPARISON OF POLYNOMIAL TIME REDUCIBILITIES

R. E. LADNER[1]

University of Washington, Seattle, Wash., USA

N. A. LYNCH

University of Southern California, Los Angeles, Calif., USA

and

A. L. SELMAN

Florida State University, Tallahassee, Fla., USA

Communicated by A. Meyer
Received 15 September 1974

Abstract. Various forms of polynomial time reducibility are compared. Among the forms examined are many-one, bounded truth table, truth table and Turing reducibility. The effect of introducing nondeterminism into reduction procedures is also examined.

1. Introduction

Computation bounded reducibilities play a role in the theory of computational complexity which is analogous to, and perhaps as important as, the role of the various kinds of effective reducibilities used in recursive function theory. Most fruitful thus far have been the polynomial time bounded reducibilities of Cook [2] and Karp [5], respectively corresponding to Turing and many-one reducibilities in recursive function theory. Other computation bounded reducibilities have been defined as well by Meyer–Stockmeyer [10], Jones [4], Ladner [6], Lynch [7], but these differ only in the bound on time or space allowed for the reduction, rather than in the form of the reduction procedure. The form of a reduction procedure refers to the kinds of reduction procedures found in Rogers [12] such as Turing, one-one, many-one, truth table, bounded truth table, positive, conjunctive and disjunctive reducibilities. Each such form of reduction procedure has a polynomial time bounded version. Our aim here is to compare these different forms of polynomial time reducibility.

Intuitively speaking, a set A is *polynomial time reducible* to a set B if there is an

[1] Research supported by NSF Grant Number GJ-34745X.

algorithm for computing whether or not a word x is in A that runs in time bounded by a polynomial in $|x|$ (the length of x) and with the added power of asking individual membership *questions* of B (at a cost of 1 move per question). This algorithm used for computing A is called a *reduction procedure* while the set B is sometimes called the *oracle* of the reduction procedure. What makes polynomial time reducibility an important tool in the study of computational complexity is the fact that *if A is polynomial time reducible to B and A is not computable in polynomial time then neither is B.*

The most commonly applied form of polynomial time reducibility is the polynomial time bounded many-one reducibility introduced by Karp [5]. A set A *is many-one reducible to B in polynomial time* ($A \leqslant_m^{\mathcal{P}} B$) if there is a function f computable in polynomial time such that $x \in A$ if and only if $f(x) \in B$. Hence a many-one reduction procedure entails exactly one membership question of the oracle. Moreover, the procedure is "positive" in the sense that the answer to the membership question of the set A is affirmative if the answer to the membership question of B is affirmative. As we shall show (in Section 2), this positiveness is not common to all polynomial time reducibilities, for any set is polynomial time reducible to its complement, yet there are infinite, coinfinite computable sets that are not many-one reducible to their complements in polynomial time.

Probably the most general form of polynomial time reducibility that can be devised is Turing reducibility in polynomial time introduced by Cook [2]. A set A *is Turing reducible to a set B in polynomial time* ($A \leqslant_T^{\mathcal{P}} B$) if there is an oracle Turing machine M and a polynomial function p such that $x \in A$ iff M accepts x with B as its oracle within $p(n)$ steps. (An oracle Turing machine is a multitape Turing machine with a special oracle tape and special states Q, YES, NO. Should the machine enter state Q then the next state is YES or NO depending on whether or not the string currently written on the oracle tape is in the oracle set.) If one can believe a modified version of Church's Thesis, then we can remove the word "probably" as the first word of this paragraph. We refer to an oracle Turing machine M with the property that there is a polynomial p such that for every oracle, M halts within $p(n)$ moves on each input of length n as a *polynomial time bounded Turing reduction procedure* or sometimes simply as a *polynomial time bounded reduction procedure.* The omission of "Turing" reflects our faith in an appropriate modification of Church's Thesis.

One interesting feature of Turing reduction procedures is that a question of the oracle may depend on the answers to previous questions. Hence even if a reduction procedure is polynomial time bounded there may be 2^n or more potential questions of the oracle asked. The particular questions asked on an input depend on the oracle that is used. For example, imagine an oracle Turing machine implementing the algorithm shown in Fig. 1.

Such an oracle Turing machine can be constructed that runs in time bounded by cn^2 for some constant c no matter what the set X is. Furthermore, given any input x

```
begin
    read x ∈ {0, 1}*;
z ← x;
    while |z| < 2 |x| do
        if z ∈ X then z ← z1 else z ← z0;
  if z ∈ X then ACCEPT else REJECT;
end
```

Fig. 1. Algorithm with 2^n potential questions. (Note: ACCEPT and REJECT are HALT statements.)

$\in \{0, 1\}^*$ and any string $y \in \{0, 1\}^*$ with length $\leqslant |x|$ there is an oracle Y such that the procedure asks the membership question "$xy \in Y$?". This amounts to $\geqslant 2^n$ potential questions of the oracle. This anomaly prompts one to ask about polynomial time bounded reduction procedures that have the property that questions asked of the oracle are independent of each other. This leads to a study of *polynomial time bounded truth table reducibility*. In Section 3 we give several equivalent definitions of polynomial time bounded truth table reducibility and show that the relation is strictly stronger than polynomial time bounded Turing reducibility. Also in Section 3 we survey a variety of strengthened forms of polynomial time bounded truth table reducibility as to their relative strength.

In Section 4 we examine nondeterministic versions of polynomial time bounded reducibilities. In general, nondeterministic reducibilities are not transitive, so that some interesting features are lost. In contrast to polynomial time bounded algorithms without oracles it can be *proven* that nondeterministic polynomial time bounded reduction procedures are more powerful than their deterministic counterparts. Indeed, a result of Baker, Gill and Solovay [1] implies that Turing reducibility in polynomial time is properly stronger than Turing reducibility in nondeterministic polynomial time.

When we speak of sets we refer to computable sets of words in a finite alphabet. Since words in any alphabet can be encoded into and decoded from the two letter alphabet $\{0, 1\}$ in polynomial time, it does not hurt to assume we are working with the fixed alphabet $\{0, 1\}$.

In general, our reducibility relations are written as "$\leqslant_r^{\mathcal{R}}$" where the superscript represents the set of allowable time bounds and subscripts represent the form of the reduction procedures defining the reducibility relation. If we allow nondeterminism, then we write "$\leqslant_r^{\mathcal{NR}}$". In most instances the set of allowable time bounds is the polynomials. If $\leqslant_r^{\mathcal{R}}$ is a reducibility then we write $A <_r^{\mathcal{R}} B$ just in case $A \leqslant_r^{\mathcal{R}} B$ and $B \not\leqslant_r^{\mathcal{R}} A$, $A \equiv_r^{\mathcal{R}} B$ just in case $A \leqslant_r^{\mathcal{R}} B$ and $B \leqslant_r^{\mathcal{R}} A$, and $A|_r^{\mathcal{R}} B$ just in case $A \not\leqslant_r^{\mathcal{R}} B$ and $B \not\leqslant_r^{\mathcal{R}} A$. If $\leqslant_r^{\mathcal{R}}$ is reflexive and transitive, then $\equiv_r^{\mathcal{R}}$ is an equivalence relation. We sometimes say A and B are $\leqslant_r^{\mathcal{R}}$-*incomparable* if $A|_r^{\mathcal{R}} B$. We say that $\leqslant_r^{\mathcal{R}}$ is *stronger* (*weaker*) than $\leqslant_s^{\mathcal{S}}$ if for all computable A and B, $A \leqslant_r^{\mathcal{R}} B$ implies $A \leqslant_s^{\mathcal{S}} B$ ($A \leqslant_s^{\mathcal{S}} B$

implies $A \leqslant_r^{\mathcal{R}} B$). One reducibility is *properly stronger* (properly weaker) than another if the first is stronger (weaker) than the second, but not *vice versa*. Two reducibilities are *incomparable* if neither is stronger than the other. We say that $\leqslant_r^{\mathcal{R}}$ *stratifies* $\leqslant_s^{\mathcal{S}}$ if there exist computable sets A and B such that $A \equiv_s^{\mathcal{S}} B$ and $A|_r^{\mathcal{R}} B$. It is impossible for the weaker of two comparable reducibilities to stratify the stronger, but not impossible for two incomparable reducibilities to stratify each other. If one reducibility is stronger than and stratifies another, then it is properly stronger.

Our comparison of polynomial time bounded reducibilities mainly deals with the classification of the forms of polynomial time bounded reducibility as to their relative strength and stratification. The basic tools we use are *encoding* and *diagonalizing*. For instance, to show $\leqslant_r^{\mathcal{R}}$ is not stronger than $\leqslant_s^{\mathcal{S}}$ one simultaneously defines computable sets A and B in such a way that (i) A is encoded effectively into B so that A can be reduced to B via an $\mathcal{R}$ time bounded r-reduction procedure (this ensures $A \leqslant_r^{\mathcal{R}} B$) and (ii) we diagonalize over all $\mathcal{S}$ time bounded s-reduction procedures to yield $A \not\leqslant_s^{\mathcal{S}} B$.

A *function f is computable in time $t(n)$* if there is a multitape Turing machine with a distinguished output tape such that if x is any input of length n then the Turing machine halts on input x within $t(n)$ moves with $f(x)$ written on the output tape. Turing machines that compute functions are often called Turing machine *transducers*. A set A *is computable in (nondeterministic) time $t(n)$* if there is a (nondeterministic) multitape Turing machine that recognizes A and on each input of length n the machine halts within $t(n)$ moves (on each computation sequence). A set has *time complexity $t(n)$* if it is computable in time $t(n)$. We define $\mathcal{P}$ to be the class of all subsets of $\{0, 1\}^*$ computable in time bounded by a polynomial and $\mathcal{NP}$ to be the class of all subsets of $\{0, 1\}^*$ computable in nondeterministic time bounded by a polynomial. We say a set A is *polynomial r-complete* if $A \in \mathcal{NP}$ and for all $B \in \mathcal{NP}$, $B \leqslant_r^{\mathcal{P}} A$.

For $m, n \in \mathcal{N}$ define $\langle m, n\rangle = \frac{1}{2}[(m+n)(m+n+1)]+m$ and inductively $\langle m_1, \ldots, m_{k+1}\rangle = \langle\langle m_1, \ldots, m_k\rangle, m_{k+1}\rangle$. Given $m_1, \ldots, m_k, \langle m_1, \ldots, m_k\rangle$ can be determined in polynomial time as a function of the sum of the lengths of $m_1, \ldots, m_k$ written in binary. For a fixed k there is a polynomial time bounded algorithm (as a function of the length of m written in binary) for determining $m_1, \ldots, m_k$ such that $m = \langle m_1, \ldots, m_k\rangle$. If A is a set we define C_A to be the characteristic function of A; $C_A(x) = 1$ if $x \in A$ and $C_A(x) = 0$ if $x \notin A$. We let $\bar{A}$ denote the complement of A in $\{0, 1\}^*$.

2. The reducibilities of Cook and Karp

The reducibilities $\leqslant_T^{\mathcal{P}}$ and $\leqslant_m^{\mathcal{P}}$ (polynomial time bounded Turing reducibility and polynomial time bounded many-one reducibility) are those defined by Cook [2] and Karp [5] respectively. (Karp used the notation "$\propto$" rather than "$\leqslant_m^{\mathcal{P}}$".)

Below we list important yet easily proved properties of these reducibility notions.

Proposition 2.1.

(i) $\leqslant_m^{\mathcal{P}}$ *and* $\leqslant_T^{\mathcal{P}}$ *are reflexive and transitive relations.*

(ii) $A \leqslant_m^{\mathcal{P}} B \Rightarrow A \leqslant_T^{\mathcal{P}} B$.

(iii) $A \leqslant_m^{\mathcal{P}} B \Rightarrow \bar{A} \leqslant_m^{\mathcal{P}} \bar{B}$.

(iv) $A \leqslant_T^{\mathcal{P}} B$, $A \leqslant_T^{\mathcal{P}} \bar{B}$, $\bar{A} \leqslant_T^{\mathcal{P}} B$ *and* $\bar{A} \leqslant_T^{\mathcal{P}} \bar{B}$ *are equivalent.*

(v) *If* $A \leqslant_T^{\mathcal{P}} B$ *and* B *is computable in* $t(n)$ *time then* A *is computable in* $p(n)+p(n)\max\{t(m): m \leqslant p(n)\}$ *time for some polynomial* p.

(vi) *If* $A \leqslant_m^{\mathcal{P}} B$ *and* B *is computable (nondeterministically) in* $t(n)$ *time then* A *is computable (nondeterministically) in* $p(n)+\max\{t(m): m \leqslant p(n)\}$ *time for some polynomial* p.

The proof of Proposition 2.1 is routine.

The following proposition is an immediate consequence of the above.

Proposition 2.2.

(i) $A \leqslant_T^{\mathcal{P}} B$ *and* $B \in \mathcal{P}$ *implies* $A \in \mathcal{P}$.

(ii) $A \leqslant_m^{\mathcal{P}} B$ *and* $B \in \mathcal{NP}$ *implies* $A \in \mathcal{NP}$.

(iii) $\mathcal{NP}$ *is closed under complement if and only if some polynomial m-complete set has its complement in* $\mathcal{NP}$.

If A and B are in $\mathcal{P}$ then $A \equiv_T^{\mathcal{P}} B$. It is pathological that this is not true for $\equiv_m^{\mathcal{P}}$, because $\emptyset$ and $\{0,1\}^*$ are $\leqslant_m^{\mathcal{P}}$ incomparable. However, we do have $A \equiv_m^{\mathcal{P}} B$ for all $A, B \in \mathcal{P} - \{\emptyset, \{0,1\}^*\}$.

It is quite trivial to reduce a set to its complement in polynomial time. Indeed the reduction procedure requires only one question of the oracle, namely the question whether or not the input itself is in the oracle set. Even though many-one reduction procedures require only one question of the oracle the "positive" requirement of these reduction procedures does not in general allow sets to be m-reducible to their complements in polynomial time.

Theorem 2.3. *There exists an infinite, coinfinite set* A *such that* $A \not\leqslant_m^{\mathcal{P}} \bar{A}$ *and* A *is computable in* 2^n *time.*

Proof. The idea is to diagonalize over all polynomial time bounded many-one reduction procedures. To eliminate M as a Turing machine transducer that many-one reduces A to $\bar{A}$ in polynomial time choose an as yet undecided input x and run M on input x. Should M halt with output y within $2^{|x|}$ moves then put both x and y into $\bar{A}$ if y is not already decided. If y is decided put x into A if and only if $y \in A$.

As the diagonalization proceeds, keep returning to M every so often, so that if M is running in polynomial time then M is caught running in 2^n moves. Also assign infinitely many words into each of A and $\bar{A}$ as the diagonalization goes on. In order to program this algorithm on a 2^n time bounded Turing machine, we employ the function h (defined below) to "spread out" the diagonalization. The idea of "spreading out" the diagonalization was inspired by a technique of M. Machtey.

Without loss of generality we assume that our Turing machine transducers are three tape Turing machines. Each has a read only input tape, a right moving only write only output tape, and a storage tape. The input and output tapes have alphabet $\{0, 1\}$ while the storage tape has alphabet $\{0, 1, B\}$. If the encoding is suitably chosen, there is an encoding of these machines into words in the alphabet $\{0, 1\}$. We have a universal Turing machine U that can simulate $d \in \{0, 1\}^*$ in such a way that there is a constant c_U with the property that if the machine described by d runs in time $T(n)$ then U may simulate the machine in time $c_U|d|\,T(n)$. Let $d_0 = \lambda, d_1 = 0, d_2 = 1, d_3 = 00, \ldots$ be the natural ordering of the words in $\{0, 1\}^*$.

Define $h : N \to N$ by $h(0) = 1$ and $h(n+1) = 2^{h(n)}$. There is a polynomial time bounded Turing machine H that on input x writes m in binary where m is the greatest number such that $h(m) \leqslant |x|$ and determines if $h(m) = |x|$. We leave the construction of H to the reader.

We now give a recursive definition of A. To begin with λ is in A. Let $x \in \{0, 1\}^*$ with $|x| = n > 0$.

(1) If $x \notin 0^*$ or $n \neq h(m)$ for some m then $x \notin A$.

(2) If $x = 0^{h(m)}$ then let $\langle i, j\rangle = m$. For $2^{h(m)}$ moves of U simulate d_i on the input x. If the machine doesn't halt then $x \notin A$. If it does halt and outputs y then:

(3) If $|y| > h(m-1)$ then $x \notin A$. If $|y| \leqslant h(m-1)$ then $x \in A$ if and only if $y \in \bar{A}$. [recursive call]

To show $A \not\leqslant_m^{\mathcal{P}} \bar{A}$, we suppose not by presuming that $A \leqslant_m^{\mathcal{P}} \bar{A}$ via a polynomial time bounded Turing machine transducer D with description d. Suppose $d = d_i$ and D is time bounded by kn^k .Choose j such that if $m = \langle i, j\rangle$ and $n = h(m)$ then $2^n > c_U|d_i|\,kn^k$. On input 0^n the algorithm for A must proceed to step 3. Let y be the output of D on input 0^n. Since d_i was simulated for only $2^n - 1$ moves of U then $|y| < 2^n = h(m+1)$. If $h(m-1) < |y|$ then 0^n is not in A by definition. With $h(m-1) < |y| < h(m+1)$ then y could be in A if and only if y is actually 0^n. If $|y| \leqslant h(m-1)$ then $0^n \in A$ if and only if $y \in A$. In any case D cannot many-one reduce A to $\bar{A}$.

Since $\lambda \in A$ and $0^3 \notin A$ then A is neither empty nor $\{0, 1\}^*$. We conclude that A is infinite and coinfinite for otherwise $A \equiv_m^{\mathcal{P}} \bar{A}$.

We now show that A can be computed in 2^n time. Imagine the algorithm for A being executed on a Turing machine. Let x be an input of length n. Using the polynomial time bounded Turing machine H, step 1 and the determination of d_i in step 2 can be done in polynomial time. The simulation of d_i by U is time bounded

by $c2^n$ for some constant c. Assuming $h(m) = n$ the test in step 3 can be accomplished in polynomial time by first comparing $|x|$ and $|y|$. If $|y| \geqslant |x|$ then certainly $|y| > h(m-1)$. If $|y| < |x|$ then using H we can find the greatest p such that $h(p) < |y|$. If $p = m-1$ then $|y| > h(m-1)$ and if $p < m-1$ then $|y| \leqslant h(m-1)$. Finally we must count the cost of the recursive call, on an input of length $\leqslant h(m-1) = \log h(m)$. On input of length not in the set $\{h(m): m \geqslant 0\}$ the algorithm for A runs in polynomial time (since no simulation or recursive call occurs). Summarizing, on an input of length n, A can be computed in time $T(n)$ where:

$$T(n) \leqslant T(\log n) + p(n) + c2^n \tag{1}$$

for some polynomial p and constant c. The inequality (1) implies $T(n) \leqslant a2^n$ for some constant a. By appealing to universal linear speed-up for Turing machines (cf. Hopcroft and Ullman [3, page 138]) A can be computed in 2^n time. □

Since $A \not\leqslant_m^{\mathcal{P}} \bar{A}$ then by Proposition 2.1(iii) $\bar{A} \not\leqslant_m^{\mathcal{P}} A$. Considering Proposition 2.1 (iv), $A \not\equiv_T^{\mathcal{P}} \bar{A}$. Hence we have:

Corollary 2.4. *$\leqslant_m^{\mathcal{P}}$ stratifies $\leqslant_T^{\mathcal{P}}$.*

Proof above.

The kind of diagonalization used in Theorem 2.3 can be applied to show that given any time bound $t(n)$ there is a computable set A such that A is not many-one reducible to $\bar{A}$ in time $t(n)$. We conclude that many-one reducibility with any recursive time bound always stratifies polynomial time bounded Turing reducibility.

3. Truth table reducibilities

The notion of truth table reducibility was originally defined by Post [11]. A set *A is truth table reducible to B* ($A \leqslant_{tt} B$) if, given x, one can effectively compute a finite set of words, say $y_1, \ldots, y_k$, and a Boolean function α such that $x \in A$ if and only if $\alpha(C_B(y_1), \ldots, C_B(y_k)) = 1$. As Post presented the notion, α was represented by its table which in many cases would be an extremely inefficient way of presenting a Boolean function. Another alternative is to allow the Boolean function to be presented as a Boolean formula with atoms of the form $y \in X$ (where y is a word in the alphabet $\{0, 1\}$ and X is a set variable). We could define $A \leqslant_{tt} B$ if and only if, given x, one can effectively compute such a formula Φ with the property that $x \in A$ if and only if Φ is satisfied with B substituted for X. A problem that arises with this approach is that given a fixed space bound, seemingly more Boolean functions can be presented using the Boolean connectives $\wedge$, $\vee$, $\neg$, and $\equiv$ than can be presented using just $\wedge$, $\vee$, and $\neg$. Rather than tie ourselves down to a particular representation of Boolean functions, we adopt an abstract approach.

Let Δ be a fixed finite alphabet for encoding Boolean functions and let $c \notin \Delta \cup \{0, 1\}$.

Definition. A tt-*condition* is a member of $\Delta^* c (c \{0, 1\}^*)^*$.

A tt-*condition generator* is a recursive mapping of $\{0, 1\}^*$ into the set of tt-conditions.

A tt-*condition evaluator* is a recursive mapping of $\Delta^* c \{0, 1\}^*$ into $\{0, 1\}$.

Let e be a tt-condition evaluator. A tt-condition $\alpha c c y_1 c y_2 c \ldots c y_k$ is *e-satisfied by* $B \subseteq \{0, 1\}^*$ if and only if $e(\alpha c c C_B(y_1) \ldots C_B(y_k)) = 1$. ($\alpha \in \Delta^*$ and $y_1, \ldots, y_k \in \{0, 1\}^*$.)

$A \leqslant^{\mathcal{P}}_{\mathrm{tt}} B$ if and only if there exist a polynomial time computable generator g and polynomial time computable evaluator e such that $x \in A$ iff $g(x)$ is e-satisfied by B. If $A \leqslant^{\mathcal{P}}_{\mathrm{tt}} B$ we say that A is *polynomial time truth table reducible to* B or A *is truth table reducible to* B *in polynomial time.*

It is not immediately apparent that this definition captures the most general notion of truth table reducibility in polynomial time. At the end of this section, we argue that it does.

To obtain various strengthenings of polynomial time bounded truth table reducibility, we place appropriate restrictions on the tt-generator and tt-evaluator in the definition.

$A \leqslant^{\mathcal{P}}_{k\text{-tt}} B$ (A is *polynomial time k-question truth table reducible to* B) provided $A \leqslant^{\mathcal{P}}_{\mathrm{tt}} B$ via a generator g and evaluator e, where g has range $\Delta^* c (c \{0, 1\}^*)^k$.

$A \leqslant^{\mathcal{P}}_{\mathrm{btt}} B$ (A is *polynomial time bounded truth table reducible to* B) if $A \leqslant^{\mathcal{P}}_{k\text{-tt}} B$ for some k.

$A \leqslant^{\mathcal{P}}_{\mathrm{p}} B$ (A is *polynomial time positive reducible to* B) if the evaluator e has the property that if $e(\alpha c \sigma_1 \ldots \sigma_k) = 1$ and $\sigma_i = 1$ implies $\tau_i = 1$ for $1 \leqslant i \leqslant k$ then $e(\alpha c \tau_1 \ldots \tau_k) = 1$.

$A \leqslant^{\mathcal{P}}_{\mathrm{c}} B$ (A is *polynomial time conjuctive reducible to* B) if the evaluator e has the property that $e(\alpha c \sigma_1 \ldots \sigma_k) = 1$ iff $\sigma_i = 1$ for $1 \leqslant i \leqslant k$.

$A \leqslant^{\mathcal{P}}_{\mathrm{d}} B$ (A is *polynomial time disjunctive reducible to* B) if $e(\alpha c \sigma_1 \ldots \sigma_k) = 0$ iff $\sigma_i = 0$ for $1 \leqslant i \leqslant k$.

It is possible to define bounded question versions of p, c, and d reducibility, by combining some of the restrictions above. For instance, $A \leqslant^{\mathcal{P}}_{2\text{-c}} B$ if $A \leqslant^{\mathcal{P}}_{2\text{-tt}} B$ via a generator g and evaluator e with e having the property described in the clause for $\leqslant^{\mathcal{P}}_{\mathrm{c}}$.

The fundamental relationships between these reducibilities are outlined below.

Proposition 3.1.

(i) *For any* $k \geqslant 1$ *we have the following*:

$$A \leqslant^{\mathcal{P}}_{m} B \Rightarrow A \leqslant^{\mathcal{P}}_{k\text{-tt}} B \Rightarrow A \leqslant^{\mathcal{P}}_{k+1\text{-tt}} B \Rightarrow A \leqslant^{\mathcal{P}}_{\mathrm{btt}} B \Rightarrow A \leqslant^{\mathcal{P}}_{\mathrm{tt}} B \Rightarrow A \leqslant^{\mathcal{P}}_{\mathrm{T}} B,$$

(ii) $A \leqslant_m^{\mathcal{P}} B \Rightarrow \begin{matrix} A \leqslant_c^{\mathcal{P}} B \\ A \leqslant_d^{\mathcal{P}} B \end{matrix} \Rightarrow A \leqslant_p^{\mathcal{P}} B \Rightarrow A \leqslant_{tt}^{\mathcal{P}} B$,

(iii) $\leqslant_m^{\mathcal{P}}, \leqslant_{btt}^{\mathcal{P}}, \leqslant_c^{\mathcal{P}}, \leqslant_d^{\mathcal{P}}, \leqslant_p^{\mathcal{P}}$ *and* $\leqslant_{tt}^{\mathcal{P}}$ *are all reflexive and transitive relations,*

(iv) $A \leqslant_{tt}^{\mathcal{P}} B$, $\bar{A} \leqslant_{tt}^{\mathcal{P}} B$, $A \leqslant_{tt}^{\mathcal{P}} \bar{B}$, $\bar{A} \leqslant_{tt}^{\mathcal{P}} \bar{B}$ *are equivalent (also for* btt *and* k-tt),

(v) $A \leqslant_p^{\mathcal{P}} B \Leftrightarrow \bar{A} \leqslant_p^{\mathcal{P}} \bar{B}$,

(vi) $A \leqslant_c^{\mathcal{P}} B \Leftrightarrow \bar{A} \leqslant_d^{\mathcal{P}} \bar{B}$,

(vii) $A \leqslant_p^{\mathcal{P}} B$ *and* $B \in \mathcal{NP} \Rightarrow A \in \mathcal{NP}$.

The proof is straightforward.

The next theorem shows that each implication in (i) and (ii) of Proposition 3.1 is proper ($\leqslant_{tt}^{\mathcal{P}}$ properly stronger that $\leqslant_T^{\mathcal{P}}$ is shown in Theorem 3.3).

Theorem 3.2.

(i) $\leqslant_m^{\mathcal{P}}$ *stratifies* $\leqslant_{1\text{-tt}}^{\mathcal{P}}$,

(ii) *for any* k, $\leqslant_{k\text{-tt}}^{\mathcal{P}}$ *stratifies both* $\leqslant_{k+1\text{-c}}^{\mathcal{P}}$ *and* $\leqslant_{k+1\text{-d}}^{\mathcal{P}}$,

(iii) $\leqslant_d^{\mathcal{P}}$ *stratifies* $\leqslant_{2\text{-c}}^{\mathcal{P}}$ *and* $\leqslant_c^{\mathcal{P}}$ *stratifies* $\leqslant_{2\text{-d}}^{\mathcal{P}}$,

(iv) $\leqslant_c^{\mathcal{P}}$ *and* $\leqslant_d^{\mathcal{P}}$ *both stratify* $\leqslant_{4\text{-p}}^{\mathcal{P}}$ *via the same pair of sets,*

(v) $\leqslant_p^{\mathcal{P}}$ *stratifies* $\leqslant_{1\text{-tt}}^{\mathcal{P}}$,

(vi) $\leqslant_{btt}^{\mathcal{P}}$ *stratifies both* $\leqslant_d^{\mathcal{P}}$ *and* $\leqslant_c^{\mathcal{P}}$.

In each case the verifying sets can be found with time complexity 2^n.

Proof. (i) actually directly follows from Theorem 2.3. In all the other cases we construct sets A and B with time complexity 2^n satisfying the appropriate stratification property. For (ii) we may just construct A and B satisfying $A \equiv_{k+1\text{-c}}^{\mathcal{P}} B$ with $A|_{k\text{-tt}}^{\mathcal{P}} B$. We then conclude that $\bar{A} \equiv_{k+1\text{-d}}^{\mathcal{P}} \bar{B}$ with $\bar{A}|_{k\text{-tt}}^{\mathcal{P}} \bar{B}$ to get the other result. In (iii) we construct sets A and B satisfying $A \equiv_{2\text{-c}}^{\mathcal{P}} B$ and $A|_d^{\mathcal{P}} B$ obtaining the other result through their complements, and in (vi) we construct sets A and B satisfying $A \equiv_d^{\mathcal{P}} B$ and $A|_{btt}^{\mathcal{P}} B$ obtaining the other result through their complements.

In each case we need to encode each set into the other in a way to get equivalence, and also diagonalize to obtain incomparability. We list below for each case conditions which exhibit the encodings. (Below we only mention strings that can possibly be in A or B. Those strings not specifically mentioned are not in the appropriate set.)

(ii) $A \equiv_{k+1\text{-c}} B$. For all strings z:
$z11 \in A \Leftrightarrow z110^j \in B$ for all j such that $1 \leqslant j \leqslant k+1$,
$z01 \in B \Leftrightarrow z010^j \in A$ for all j such that $1 \leqslant j \leqslant k+1$,
$z110^j \in A \Leftrightarrow z110^j \in B$ for $1 \leqslant j \leqslant k+1$,
$z010^j \in A \Leftrightarrow z010^j \in B$ for $1 \leqslant j \leqslant k+1$.

(iii) $A \equiv^{\mathcal{P}}_{2\text{-c}} B$. For all strings z:
$z11 \in A \Leftrightarrow z110 \in B$ and $z1100 \in B$,
$z01 \in B \Leftrightarrow z010 \in A$ and $z0100 \in A$,
$z110^j \in A \Leftrightarrow z110^j \in B$ for $1 \leqslant j \leqslant 2$,
$z010^j \in A \Leftrightarrow z010^j \in B$ for $1 \leqslant j \leqslant 2$.

(iv) $A \equiv^{\mathcal{P}}_{4\text{-p}} B$. For all strings z:
$z11 \in A \Leftrightarrow (z110 \in B$ and $z110^2 \in B)$ or $(z110^3 \in B$ and $z110^4 \in B)$,
$z01 \in B \Leftrightarrow (z010 \in A$ and $z010^2 \in A)$ or $(z010^3 \in A$ and $z010^4 \in A)$,
$z110^j \in A \Leftrightarrow z110^j \in B$ for $1 \leqslant j \leqslant 4$,
$z010^j \in A \Leftrightarrow z010^j \in B$ for $1 \leqslant j \leqslant 4$.

(v) $A \equiv^{\mathcal{P}}_{1\text{-tt}} B$ because $B = \bar{A}$.

(vi) $A \equiv^{\mathcal{P}}_{\text{d}} B$. For all strings z and numbers $k > 0$:
$z10^k11 \in A \Leftrightarrow z10^k110^j \in B$ for some j where $1 \leqslant j \leqslant k+1$,
$z10^k01 \in B \Leftrightarrow z10^k010^j \in A$ for some j where $1 \leqslant j \leqslant k+1$,
$z10^k110^j \in A \Leftrightarrow z10^k110^j \in B$ for $1 \leqslant j \leqslant k+1$,
$z10^k010^j \in A \Leftrightarrow z10^k010^j \in B$ for $1 \leqslant j \leqslant k+1$.

Rather than give five separate diagonalizations, we shall give the details of just (vi), the A and B satisfying $A \equiv^{\mathcal{P}}_{\text{tt}} B$ but $A \not\equiv^{\mathcal{P}}_{\text{btt}} B$. The others follow similar lines.

Let h be defined as in Theorem 2.3 ($h(0) = 1$ and $h(n+1) = 2^{h(n)}$) and let H be the polynomial time bounded Turing machine that on input x determines the least m such that $|x| \geqslant h(m)$. We construct A and B in stages. At stage m we determine for each x with $h(m) \leqslant |x| < h(m+1)$ whether or not x is in A and whether or not x is in B.

Let members of $\{0, 1\}^*$ represent descriptions of tt-condition generators and tt-condition evaluators. As in Theorem 2.3, we can assume that our generators and evaluators take the form of three tape Turing machines. (For convenience, we let $g_0, g_1, \ldots$ denote generators and $e_0, e_1, \ldots$ denote evaluators.) We have universal machines G and E which can simulate tt-condition generators and tt-condition evaluators. There are simulation constants c_{G} and c_{E} for G and E with the same properties as the constant c_{U} in the universal machine U of Theorem 2.3.

We now define A and B. To begin with, we have $\lambda \notin A$ and $\lambda \notin B$.

Stage m: We have $m = 2\langle i, j, k, l\rangle + \sigma$ for some $i, j, k, l \in N$ and $\sigma = 0$ or 1. If $h(m+1) \leqslant h(m) + 2k + 4$ go to the next stage. Otherwise set $n = h(m)$.

1. Simulate g_i on input $0^n10^k\sigma1$ for $2^n - 1$ moves of G. If g_i doesn't halt or halts with an output of the form $\alpha ccy_1\, c \ldots cy_p$ with $p > k$ then go to the next stage. Suppose then that g_i outputs $\alpha ccy_1\, c \ldots cy_p$ with $y_1, \ldots, y_p \in \{0, 1\}^*$, $\alpha \in \Delta^*$ and $p \leqslant k$.

2. If $\sigma = 0$ then set $w = C_A(y_1) \ldots C_A(y_p)$ and if $\sigma = 1$ set $w = C_B(y_1) \ldots C_B(y_p)$. We assume that if $|y_q| \geqslant n$ then $C_A(y_q) = C_B(y_q) = 0$ so that this step is a recursive call.

3. Simulate e_j on input αcw for 2^n-1 moves of E. If e_j doesn't halt or outputs 1, go to the next stage. If e_j outputs 0 and $\sigma = 0$ put 0^n10^k01 into B and $0^n10^k010^r$ into both A and B where r is the least number > 0 such that $0^n10^k010^r \notin \{y_1, ..., y_p\}$. If e_j outputs 0 and $\sigma = 1$ put 0^n10^k11 into A and $0^n10^k110^r$ into both A and B where r is the least number > 0 such that $0^n10^k110^r \notin \{y_1, ..., y_p\}$.

End of stage m.

It is not difficult to verify that the condition guaranteeing $A \equiv^{\mathcal{P}}_{\text{tt}} B$ is forced by the construction. We now argue that $A \not\leqslant^{\mathcal{P}}_{\text{btt}} B$. The argument that $B \not\leqslant^{\mathcal{P}}_{\text{btt}} A$ is symmetric.

Suppose $A \leqslant^{\mathcal{P}}_{\text{btt}} B$. Let g and e be a generator and evaluator, respectively, that witness this reduction and suppose k is a bound on the number of questions of g (i.e. Range $g \subseteq \Delta^* c\,(c\,\{0,1\}^*)^k$). Let g be time bounded by rn^r and e time bounded by sn^s. Let g_i be a description of g and e_j a description of e. Choose l such that if $m = 2\langle i,j,k,l\rangle + 1$ then $h(m+1) > h(m) + 2k + 4$ and $2^{h(m)} > c_G|g_i|\, r(h(m) + k+3)^r + c_E|e_j|\, s\,[r(h(m)+k+3)^r]^s$. At stage m the algorithm proceeds through step 3. It can be easily checked that $0^{h(m)}10^k11 \in A$ if and only if $g(0^{h(m)}10^k11)$ is not e-satisfied by B. The main thing to notice is that if $g(0^{h(m)}10^k11) = \alpha ccy_1\, c \ldots cy_p$ with $\alpha \in A^*$ and $y_1, ..., y_p \in \{0,1\}^*$ then no member in the set $\{y_1, ..., y_p\}$ enters B at a stage $\geqslant m$ so that the value $e(\alpha cC_B(y_1) \ldots C_B(y_p))$ determined at stage m cannot change at a later time.

We now show that A and B can be computed in 2^n time. Let x be given with $n = |x|$. To determine the membership of x in A and B we need only compute the above algorithm through stage m where m is the greatest number such that $h(m) \leqslant n$. Using H, m can be determined in polynomial time as a function of n. Let $T(m) =$ the time to compute stage m. By an analysis similar to that of Theorem 2.3,

$$T(m) \leqslant T(m-1) + c2^{h(m)}$$

for some constant c. Hence $T(m) \leqslant c \sum_{l=0}^{m} 2^{h(l)}$ which is $\leqslant a2^{h(m)}$ for some constant a. Since $h(m) \leqslant n$ and by universal linear speed up A and B can be computed in time 2^n. □

Again the kind of diagonalization used in Theorem 3.2 can be applied to general time bounded reducibilities. For instance, given any time bound $t(n)$ one can find sets A and B (no longer of time complexity 2^n) such that A is truth table reducible to B in polynomial time, but not bounded truth table reducible to B in time $t(n)$. This kind of generalization is *not* true of the following result.

Theorem 3.3. *There exist sets A and B of time complexity 2^n with $A \equiv^{\mathcal{P}}_{\text{T}} B$ and $A \not\equiv^{\mathcal{P}}_{\text{tt}} B$.*

In contrast to our earlier results, this result does not generalize to time bounds other than polynomial. To be specific, it is not hard to show that if $A \leqslant^{\mathcal{P}}_{\text{T}} B$ then A is truth table redcible to B in time $2^{p(n)}$ for some polynomial p.

Proof of Theorem 3.3. We let h, H, $\{g_i\}_{i\in N}$, $\{e_i\}_{i\in N}$, G and E be as in Theorem 3.2. We construct A and B in such a way that the following conditions assuring $A \equiv_T B$ are satisfied.

(a) A contains only strings of the form 0^n11x or 0^n10y where $0 \leqslant |x| \leqslant n$ and $1 \leqslant |y| \leqslant n$. B contains only strings of the form 0^n10x or 0^n11y where $0 \leqslant |x| \leqslant n$ and $1 \leqslant |y| \leqslant n$.

(b) For each n there are strings y and z, both of length n, such that if $0^n10x \in A$ then x is a nonempty prefix of y and if $0^n11x \in B$ then x is a nonempty prefix of z.

(c) The string 0^n11 is in A if and only if there is a string y of length n such that $0^n11x \in B$ for all nonempty prefixes x of y. Likewise, $0^n10 \in B$ if and only if there is a string z of length n such that $0^n10x \in A$ for all nonempty prefixes x of z.

(d) If $1 \leqslant |y| \leqslant n$ then $0^n10y \in A$ if and only if $0^n10y \in B$ and $0^n11y \in A$ if and only if $0^n11y \in B$.

The procedure of Fig. 2 reduces A to B in polynomial time.

```
begin
    read x;
    if x of form 0^n11 then
        begin
            z ← x;
            while |z| < 2|x|−2 do
                if z0 ∈ B then z ← z0 else
                if z1 ∈ B then z ← z1 else
                REJECT;
            ACCEPT;
        end;
    if x of form 0^n10y or 0^n11y where 1 ⩽ |y| ⩽ n then
        if x ∈ B then ACCEPT else REJECT;
    REJECT;
end
```

Fig. 2. Procedure reducing A to B in polynomial time.

A similar algorithm reduces B to A in polynomial time.

As in Theorem 3.2 we construct A and B in stages. At stage m we determine the membership of x in A or B where $h(m) \leqslant |x| < h(m+1)$. Initially λ is in neither A nor B.

Stage m: We have $m = 2\langle i, j, k\rangle + \sigma$ for some $i, j, k \in N$ and $\sigma = 0$ or 1. If $h(m+1) \leqslant 2h(m)+2$ go to the next stage. Otherwise set $n = h(m)$.

1. Simulate g_i on input $0^n1\sigma$ for 2^n-1 moves of G. If g_i doesn't halt, go to the next stage. Suppose g_i halts with output $\alpha ccy_1c \ldots cy_p$ where $\alpha \in \Delta^*$ and $y_1, \ldots, y_p \in \{0, 1\}^*$.

2. Find a y of length n such that $0^n1\sigma y \notin \{y_1, ..., y_p\}$. (Such a y exists because there are 2^n strings of length n and $p < 2^n$ since g_i was simultated for $\leqslant 2^n-1$ moves.)

3. Put $0^n1\sigma x$ into both A and B for all proper prefixes x of y. (x is a proper prefix of y if x is a prefix of y and $x \neq \lambda$ and $x \neq y$.)

4. If $\sigma = 0$ set $w = C_A(y_1) ... C_A(y_p)$ and if $\sigma = 1$ set $w = C_B(y_1) ... C_B(y_p)$.

5. Simulate e_j on input αcw for 2^n-1 moves of E. If e_j doesn't halt or outputs 1 go to the next stage. If e_j outputs 0 then put 0^n10 into B and 0^n10y into A and B if $\sigma = 0$ or put 0^n11 into A and 0^n11y into A and B if $\sigma = 1$.

End of stage m.

The diagonalization succeeds because, if say, 0^n11 enters A in step 5 then 0^n11y enters B without changing the value of $C_B(y_1) ... C_B(y_p)$.

An analysis of the algorithm to show A and B have time complexity 2^n is similar to that of Theorems 2.3 and 3.2. □

As we promised at the beginning of the section, we now argue that our definition of truth table reducibility in polynomial time is one that models the intuitive idea of a polynomial time reducibility where the questions asked of the oracle are "independent" of each other. We do this by presenting two alternative definitions.

Proposition 3.4. *For all sets A and B, $A \leqslant_{tt}^{\mathcal{P}} B$ if and only if there is an oracle Turing machine* M *and a polynomial time computable function f: $\{0, 1\}^* \to (c\{0, 1\}^*)^*$ such that* M *reduces A to B in polynomial time and on each input x,* M *only asks questions of B from the list $f(x)$.*

(What we mean by M only asking questions of B from the list $f(x)$ on input x is that on input x and with oracle B if M enters state Q then the string currently written on the oracle tape is in the set $\{y_1, ..., y_k\}$ where $f(x) = cy_1cy_2c ... cy_k$ and $y_1, ..., y_k \in \{0, 1\}^*$.)

Proof. Suppose $A \leqslant_{tt} B$ via tt-generator g and tt-evaluator e. Define $f(x) = cy_1\, c ... cy_k$ where $g(x) = \alpha ccy_1\, c ... cy_k$. Define M by the following algorithm.

> On input x compute $g(x) = \alpha ccy_1\, c ... cy_k$ where $y_1, ..., y_k \in \{0, 1\}^*$. Evaluate $w = C_B(y_1) ... C_B(y_k)$. Compute $\sigma = e(\alpha cw)$. If $\sigma = 0$, reject and if $\sigma = 1$, accept.

Such a computation can be carried out by an oracle Turing machine in polynomial time.

Assume A, B, M and f are given satisfying the property of the proposition. Define $\Delta = \{0, 1\}$ and define $g(x) = xcf(x)$. Define e by the following algorithm.

> Let $xcz \in \{0, 1\}^* c \{0, 1\}^*$ be an input. Compute $f(x) = cy_1\, c ... cy_k$ where $y_1, ..., y_k \in \{0, 1\}^*$. If $k \neq |z|$ set $e(xcz) = 0$. Otherwise let $z = \sigma_1\, \sigma_2 ... \sigma_k$ where $\sigma_i \in \{0, 1\}$ for $1 \leqslant i \leqslant k$. Simulate M on input x with the modification that should M enter state Q with y on the oracle

tape, then (1) if $y \notin \{y_1, ..., y_k\}$ then set $e(xcz) = 0$ and (2) if $y = y_i$ for $1 \leqslant i \leqslant k$ then go to state YES if $\sigma_i = 1$ and go to state NO if $\sigma_i = 0$. Should M accept, set $e(xcz) = 1$ and should M reject, set $e(xcz) = 0$.

The reader can verify that $A \leqslant_{\text{tt}}^{\mathcal{P}} B$ via g and e. □

The following equivalent definition was suggested by Albert R. Meyer [8]. Let Δ be an alphabet for encoding combinational circuits with just AND and NOT as operations. For a precise definition of combinational circuits, see Savage [13].

Proposition 3.5. *For all sets A and B, $A \leqslant_{\text{tt}}^{\mathcal{P}} B$ if and only if there is a polynomial time computable function $f: \{0, 1\}^* \to \Delta^* c (c \{0, 1\}^*)^*$ such that if $f(x) = \alpha c c y_1 c ... c y_k$ with $y_1, ..., y_k \in \{0, 1\}^*$ and $\alpha \in \Delta^*$, then $x \in A$ just in case $\alpha(C_B(y_1), ..., C_B(y_k)) = 1$.*

Proof. There is an efficient method for constructing combinational circuits which simulate Turing machines, which has been noticed by several researchers and was communicated to us by A. R. Meyer.

Let p be a polynomial and let M be a Turing machine acceptor that runs in $p(n)$ time. There is an algorithm (Turing machine) that runs in approximately $p(n)^2$ time which given any input of length n produces the encoding in the alphabet Δ of a combinational circuit α with the property that for all $\sigma_1, ..., \sigma_n \in \{0, 1\}$, $\sigma_1 ... \sigma_n$ is accepted by M if and only if $\alpha(\sigma_1, ..., \sigma_n) = 1$. Roughly speaking, the circuit α has $p(n)$ levels each of length approximately $p(n)$. The t-th level corresponds to the state, tape contents and head position of M at time t. The circuitry connecting the t-th and $(t+1)$-st level simulates the action of M.

Let $A \leqslant_{\text{tt}}^{\mathcal{P}} B$ via a tt-generator g and tt-evaluator e. For each $x \in \{0, 1\}^*$ consider the Boolean function α_x defined by:

Domain $(\alpha_x) = \{0, 1\}^k$ where $g(x) = \alpha c c y_1 c ... c y_k$ and $y_1, ..., y_k \in \{0, 1\}^*$.
$\alpha_x(\sigma_1, ..., \sigma_k) = 1$ if and only if $e(\alpha c \sigma_1 ... \sigma_k) = 1$.

Using a method somewhat like that above, there is a polynomial time bounded algorithm for producing α_x given x as an input. Hence A and B satisfy the property of the proposition.

Assuming A and B satisfy the property of the proposition, it is easy to see that $A \leqslant_{\text{tt}}^{\mathcal{P}} B$. We are assuming, of course, that we have a nice encoding of the combinational circuits so that given an encoded circuit α and $\sigma_1, ..., \sigma_n \in \{0, 1\}$, $\alpha(\sigma_1, ..., \sigma_n)$ can be evaluated in polynomial time as a function of $|\alpha|$. □

Proposition 3.4 suggests a close connection between polynomial time Turing reducibility and polynomial time truth table reducibility. Polynomial time truth table reducibility is just polynomial time Turing reducibility where the questions of the oracle can be computed in polynomial time ahead of time. The equivalent definition of Proposition 3.5 is reminiscent of Rogers' [12] definition of truth table reducibility.

4. Nondeterministic reducibilities

A natural way to generalize the definitions of Sections 2 and 3 is to allow nondeterminism in the reducibility procedures. Such notions exist in the literature; Baker, Gill and Solovay [1] and Meyer and Stockmeyer [9] have used a notion of nondeterministic polynomial time Turing reducibility. Define $\mathcal{P}^A$ ($\mathcal{NP}^A$) to be the family of sets (nondeterministic) polynomial time Turing reducible to A. Baker, Gill and Solovay [1] have discovered recursive sets A and B such that

(1) $\mathcal{P}^A = \mathcal{NP}^A$,

(2) $\mathcal{P}^B \neq \mathcal{NP}^B$.

Since many standard diagonalization and simulation arguments relativize to oracle machines, their theorem supports the contention that no standard diagonalization or simulation method can be used to demonstrate that $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$.

Meyer and Stockmeyer [9] construct a potential hierarchy (paralleling the arithmetical hierarchy) using the notion of nondeterministic polynomial time reducibility.

A nondeterministic (oracle) Turing machine *runs in polynomial time* or *is polynomial time bounded* if there is a polynomial p such that for every n and every input x of length n (and oracle A) all possible courses of computation beginning in the initial configuration halt in $\leqslant p(n)$ moves. An oracle Turing machine M *accepts x with oracle A* if there is some accepting computation by M on input x with oracle A. The *set accepted by* M *with oracle A* is the set of all $x \in \{0, 1\}^*$ such that M accepts x with oracle A. A nondeterministic Turing machine transducer M *computes y on input x* if on input x, for some computation, M halts with y written on the output tape.

$A \leqslant_{\mathrm{T}}^{\mathcal{NP}} B$ (A is *nondeterministic polynomial time Turing reducible to B*) if and only if there is a nondeterministic oracle Turing machine M that runs in polynomial time and A is accepted by M with oracle B.

$A \leqslant_{m}^{\mathcal{NP}} B$ (A is *nondeterministic polynomial time many-one reducible to B*) if and only if there is a nondeterministic Turing machine transducer M that runs in polynomial time such that $x \in A$ just in case there is a y computed by M on input x with $y \in B$.

$A \leqslant_{\mathrm{tt}}^{\mathcal{NP}} B$ (A is *nondeterministic polynomial time truth table reducible to B*) if and only if there is a nondeterministic Turing machine transducer M that runs in polynomial time and a polynomial time computable evaluator e such that $x \in A$ just in case on input x, M computes a tt-condition y which is e-satisfied by B.

Note: The definition of $\leqslant_{\mathrm{tt}}^{\mathcal{NP}}$ introduces nondeterminism into the generator but not the evaluator. Allowing nondeterminism in the evaluator as well yields the same reducibility notion.

We may make appropriate modifications in the last definition to obtain definitions for $\leqslant_{\mathrm{btt}}^{\mathcal{NP}}$, $\leqslant_{k\text{-tt}}^{\mathcal{NP}}$, $\leqslant_{\mathrm{p}}^{\mathcal{NP}}$, $\leqslant_{\mathrm{c}}^{\mathcal{NP}}$, $\leqslant_{\mathrm{d}}^{\mathcal{NP}}$.

The following theorem indicates an interesting collapse that is very different from

the deterministic reducibilities. In part, this difference comes from nondeterminism replacing the power of disjunctions.

Theorem 4.1. *For all sets A and B,*

(i) $A \leqslant_{\mathrm{T}}^{\mathcal{NP}} B \Leftrightarrow A \leqslant_{\mathrm{tt}}^{\mathcal{NP}} B$,

(ii) $A \leqslant_{\mathrm{p}}^{\mathcal{NP}} B \Leftrightarrow A \leqslant_{\mathrm{c}}^{\mathcal{NP}} B$,

(iii) $A \leqslant_{\mathrm{d}}^{\mathcal{NP}} B \Leftrightarrow A \leqslant_{m}^{\mathcal{NP}} B$.

Proof. Let $\Delta = \{0, 1, \wedge, \neg\}$. A tt-condition has the form $\sigma_1\, x_1 \wedge \sigma_2\, x_2 \wedge \ldots \wedge \sigma_k x_k\, ccx_1 c \ldots cx_k$ where $x_1, \ldots, x_k \in \{0, 1\}^*$ and $\sigma_1, \ldots, \sigma_k \in \{\lambda, \neg\}$. Define f by

$$f(c) = 0,$$

$$f(\sigma_1\, x_1 \wedge \ldots \wedge \sigma_k\, x_k\, c\tau_1 \ldots \tau_k) = \begin{cases} 1 & \text{if for all } i,\ \sigma_i = \neg \text{ coincides with } \tau_i = 0, \\ 0 & \text{otherwise,} \end{cases}$$

for $\sigma_1, \ldots, \sigma_k \in \{\lambda, \neg\}$, $x_1, \ldots, x_k \in \{0, 1\}^*$ and $\tau_1, \ldots, \tau_k \in \{0, 1\}$, and f evaluated at other inputs is 0. Clearly, f is computable in polynomial time.

The first equivalence is obtained by showing that if $A \leqslant_{\mathrm{T}}^{\mathcal{NP}} B$ then $A \leqslant_{\mathrm{tt}}^{\mathcal{NP}} B$. Let M be a nondeterministic oracle Turing machine that reduces A to B in polynomial time. Consider the nondeterministic transducer g defined by the following algorithm:

> Let $x \in \{0, 1\}^*$ be an input. Nondeterministically simulate M with the following modifications. Maintain sets Y and N which are initially empty. Should M enter state Q then let y be the current string on the oracle tape. If $y \notin Y \cup \mathbf{N}$ then nondeterministically put y into one of the two sets. If $y \in Y$ then go to state YES and if $y \in \mathbf{N}$ go to state NO. Should M halt and accept, then output $y_1 \wedge \ldots \wedge y_i \wedge \neg z \wedge \ldots \wedge \neg z_j\, ccy_1\, c \ldots cy_i\, cz_1\, c \ldots cz_j$ where $Y = \{z_1, \ldots, z_j\}$ and $\mathbf{N} = \{z_1, \ldots, z_j\}$. ($cc$ is outputted if $Y \cup \mathbf{N} = \emptyset$ or if M halts and rejects.)

It is not difficult to check that $x \in A$ if and only if g computes a tt-condition that is f-satisfied by B.

For the second equivalence, assume $A \leqslant_{\mathrm{p}}^{\mathcal{NP}} B$ via a nondeterministic Turing machine transducer g and a tt-condition evaluator e (with the condition guaranteeing positivity). Define f by $f(y_1 \wedge \ldots \wedge y_k\, c1^k) = 1$ for all $k \geqslant 0$ and $y_1, \ldots, y_k \in \{0, 1\}^*$ and f is 0 at all other arguments.

Consider the nondeterministic transducer h defined by the following algorithm:

> Let $x \in \{0, 1\}^*$ be an input. Nondeterministically simulate g to obtain an output $\alpha ccy_1\, c \ldots cy_k$. Nondeterministically select $\tau_1, \ldots, \tau_k \in \{0, 1\}$. Compute $\sigma = e(\alpha c\tau_1 \ldots \tau_k)$. If $\sigma = 0$ output $0cc$ (garbage) and if $\sigma = 1$ output $y_{i_1} \wedge \ldots \wedge y_{i_m}\, ccy_{i_1} c \ldots cy_{i_m}$ where $\{i_1 < \ldots < i_m\} = \{i : \tau_i = 1\}$.

The reader may verify that $x \in A$ if and only if h computes a tt-condition that is f-satisfied by B, and that the reduction procedure is conjunctive.

We leave the verification of the third equivalence to the reader. There is a small pathology in (iii), namely that depending on the precise definitions of $\leqslant_{\mathrm{d}}^{\mathcal{NP}}$ and $\leqslant_{\mathrm{m}}^{\mathcal{NP}}$ (iii) may fail if A or B equals $\emptyset$ or $\{0, 1\}^*$. □

As we mentioned in the Introduction, nondeterministic reducibilities are not in general transitive, with nontransitivity seeming to accompany the allowance of negations.

Theorem 4.2.
(i) $\leqslant_{m}^{\mathcal{NP}}$ *and* $\leqslant_{\mathrm{c}}^{\mathcal{NP}}$ *are transitive,*
(ii) $\leqslant_{\mathrm{tt}}^{\mathcal{NP}}$, $\leqslant_{\mathrm{btt}}^{\mathcal{NP}}$, *and* $\leqslant_{\mathrm{k\text{-}tt}}^{\mathcal{NP}}$ *are not transitive.*

Proof. (i) is proved by straightforward simulation, and (ii) is a consequence of the following lemma. □

Lemma 4.3. *There exist computable sets A, B and C with $A \leqslant_{\mathrm{1\text{-}tt}}^{\mathcal{NP}} B$, $B \leqslant_{\mathrm{1\text{-}tt}}^{\mathcal{NP}} C$, but $A \not\leqslant_{\mathrm{tt}}^{\mathcal{NP}} C$.*

Proof. We construct A, B and C satisfying the properties

$$x \in A \Leftrightarrow (\exists y)\,[|y| = |x| \text{ and } y \in \bar{B}] \text{ and}$$
$$x \in B \Leftrightarrow (\exists y)\,[|y| = |x| \text{ and } y \in C].$$

These conditions guarantee that $A \leqslant_{\mathrm{1\text{-}tt}}^{\mathcal{NP}} B$ and $B \leqslant_{\mathrm{1\text{-}tt}}^{\mathcal{NP}} C$.

In diagonalizing to assure $A \not\leqslant_{\mathrm{tt}}^{\mathcal{NP}} C$ we look at a typical nondeterministic tt-condition generator g and typical tt-condition evaluator e. Let n be large enough so that strings of length $\geqslant n$ entering C do not ruin earlier diagonalizations. Find all tt-conditions that can be outputted by g on input 0^n in $\leqslant 2^n$ moves. For each tt-condition $\alpha c c y_1 c \ldots c y_k$ outputted by g, try to compute $e(\alpha c C_C(y_1) \ldots C_C(y_k))$ for 2^n moves where $C_C(y_i)$ is determined by the current C. Should this testing yield a tt-condition $\alpha c c y_1 c \ldots c y_k$ that is e-satisfied by C then a diagonalization is accomplished by leaving all strings of length n out of A, putting all strings of length n into B and putting a string $y \notin \{y_1, \ldots, y_k\}$ with $|y| = n$ into C. If no tt-condition is satisfied, then put all strings of length n into A and no strings of length n into either B or C. □

In this lemma we did not make an attempt to get A, B, and C to have time complexity 2^n. We do not know if it is possible to do so. As part of the diagonalization, we seem to need to calculate all possible computations of $\leqslant 2^n$ moves on an input of length n on a nondeterministic Turing machine. This leads to a 2^{2^n} time bounded algorithm. This can be improved somewhat by using a time bound that dominates all polynomials and is smaller than 2^n, but this improvement does not yield a simple exponential (2^n) time bound.

The relation $\leqslant_c^{\mathcal{NP}}$ has several interesting properties other than being transitive. It may be easily shown that $A \leqslant_c^{\mathcal{NP}} B$ and $B \in \mathcal{NP}$ implies $A \in \mathcal{NP}$. It seems that $\leqslant_c^{\mathcal{NP}}$ is the polynomial time bounded analogue of enumeration reducibility ($\leqslant_e$) (cf. Rogers [12, page 145]). The following equivalent definition of $\leqslant_c^{\mathcal{NP}}$ parallels quite closely Rogers' [12, page 146] definition of $\leqslant_e$:

$A \leqslant_c^{\mathcal{NP}} B$ if and only if there is a polynomial p and a member W of $\mathcal{NP}$ (in the alphabet $\{0, 1, c\}$) with the property that $x \in A$ just in case there is a string α with $|\alpha| \leqslant p(|x|)$, $xc\alpha \in W$, and $\{y_1, \ldots, y_k\} \subseteq B$ where $\alpha = cy_1\, cy_2 \ldots cy_k$ with $y_1, \ldots, y_k \in \{0, 1\}^*$.

For nondeterministic reducibilities whose definitions do not collapse, our stratification results of Theorem 3.2 are strengthened by allowing diagonalization over nondeterministic reducibilities. However, we seem to lose the nice 2^n time bound as a consequence of the more powerful diagonalization.

Theorem 4.4.

(i) $\leqslant_m^{\mathcal{NP}}$ *stratifies* $\leqslant_{1\text{-tt}}^{\mathcal{P}}$,

(ii) *for any* k, $\leqslant_{k\text{-tt}}^{\mathcal{NP}}$ *stratifies* $\leqslant_{k+1\text{-c}}^{\mathcal{P}}$ (*the corresponding statement for* $\leqslant_{k+1\text{-d}}^{\mathcal{P}}$ *is false by Theorem* 4.1(iii)),

(iii) $\leqslant_d^{\mathcal{NP}}$ *stratifies* $\leqslant_{2\text{-c}}^{\mathcal{P}}$ (*the corresponding statement with* c *and* d *interchanged is false*),

(iv) $\leqslant_p^{\mathcal{NP}}$ *stratifies* $\leqslant_{1\text{-tt}}^{\mathcal{P}}$,

(v) $\leqslant_{btt}^{\mathcal{NP}}$ *stratifies* $\leqslant_d^{\mathcal{P}}$ *and* $\leqslant_c^{\mathcal{P}}$.

In each case the verifying sets can be found with time complexity roughly 2^{2^n}.

Proof. In general, the proof follows the lines of the proof of Theorem 3.2 with strengthened diagonalization over nondeterministic reduction procedures instead of just deterministic ones. □

Fig. 3 illustrates the relative strengths of the most important transitive reducibilities that we have studied.

The deterministic portion of Fig. 3 follows from Proposition 3.1 and Theorems 3.2 and 3.3. The nondeterministic portion of Fig. 3 follows from Theorem 4.4 (iii). The two implications connecting the deterministic portion to the nondeterministic portion follow from Theorem 4.1. The incomparabilities between the deterministic and nondeterministic portions can be demonstrated first by Theorem 4.4 (iii) and (iv), which imply respectively:

(1) there are A and B with $A \leqslant_{btt}^{\mathcal{P}} B$, $A \leqslant_c^{\mathcal{P}} B$ and $A \not\leqslant_m^{\mathcal{NP}} B$; and

(2) there are A and B with $A \leqslant_{btt}^{\mathcal{P}} B$ and $A \not\leqslant_c^{\mathcal{NP}} B$.

Finally, the method of Baker, Gill, and Solovay [1] mentioned earlier actually yields sets A and B with $A \leqslant_m^{\mathcal{NP}} B$ and $A \not\leqslant_T^{\mathcal{P}} B$.

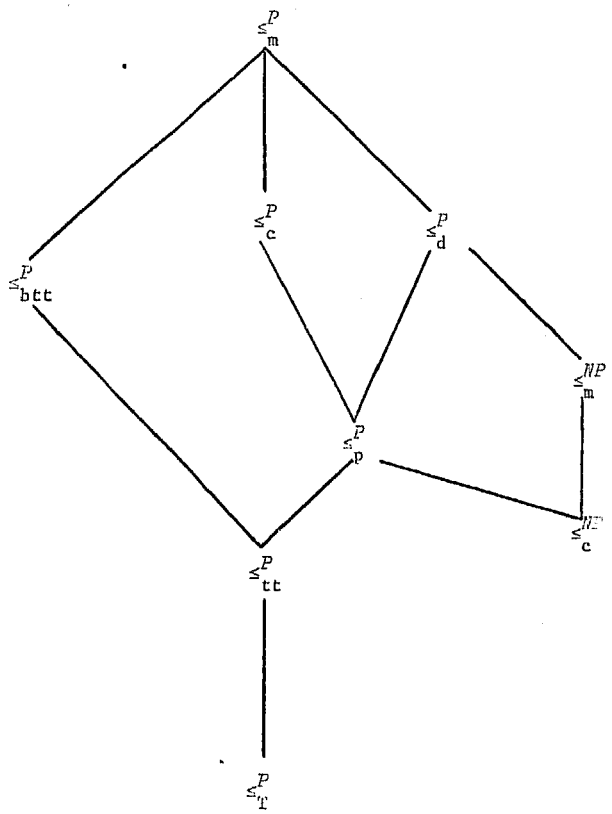

Fig. 3. Relative strengths of polynomial time reducibility notions.

5. Conclusion

We know that our deterministic polynomial time reducibilities differ on the sets computable in 2^n time. With a little more effort, we can show that these reducibilities differ on the sets computable in $t(n)$ time whenever $t(n)$ is time countable and eventually dominates each polynomial. (A function $t(n)$ is *time countable* if there is a Turing machine T such that for each n if x is a string of length n then T halts in exactly $t(n)$ moves on input x.) We would like to show that they differ on $\mathcal{NP}$ (for example, that there exist A, $B \in \mathcal{NP}$ such that $A \leqslant^{\mathcal{P}}_{\mathrm{T}} B$ but $A \not\leqslant^{\mathcal{P}}_{m} B$ for A, $B \notin \{\emptyset, \{0,1\}^*\}$). This, of course, would imply $\mathcal{P} \neq \mathcal{NP}$. On the other hand, we conjecture but have not yet proved that $\mathcal{P} \neq \mathcal{NP}$ implies $\leqslant^{\mathcal{P}}_{\mathrm{T}}$ and $\leqslant^{\mathcal{P}}_{m}$ differ on $\mathcal{NP}$. Perhaps a stronger result is possible; namely, $\mathcal{P} \neq \mathcal{NP}$ implies there are polynomial T-complete sets that are not polynomial m-complete. Similar questions can be asked of other deterministic reducibilities.

If one reducibility is properly stronger than another, then it is usually not too difficult to show that the stronger stratifies the weaker. Other notions of distinctness between reducibilities are possible; for instance, is it true that for all $A \notin \mathcal{P}$ there is B such that $A \leqslant_{\mathrm{T}}^{\mathcal{P}} B$ and $A \not\leqslant_{m}^{\mathcal{P}} B$?

It may be interesting to compare notions of log space reducibility (Meyer and Stockmeyer [10] and Jones [4]) with polynomial time reducibilities. It is not too difficult to show that log space Turing reducibility is stronger than polynomial time truth table reducibility. Does the converse hold? It is also an open question as to whether or not log space many-one reducibility and polynomial time many-one reducibility are the same notion. Definitions of reducibilities are meaningful not only when time and space bounds are placed on the computation, but when the questions asked are of bounded length (for example, the log-lin reducibility of Meyer and Stockmeyer [10]). These can be compared also.

The case of $\leqslant_{\mathrm{c}}^{\mathcal{NP}}$ as a reducibility notion is quite interesting. It is analogous to enumeration reducibility from recursive function theory. It is a transitive relation, but is incomparable with $\leqslant_{\mathrm{T}}^{\mathcal{P}}$. The set $\mathcal{NP}$ is the zero degree of the $\leqslant_{\mathrm{c}}^{\mathcal{NP}}$-degrees; in particular, for all $A \in \mathcal{NP}$ and for all B, $A \leqslant_{\mathrm{c}}^{\mathcal{NP}} B$. It would be nice to find applications of this notion to classifying the complexity of concrete problems. It is interesting to note that the third author has recently shown that $\leqslant_{\mathrm{c}}^{\mathcal{NP}}$ is a maximal transitive subrelation of $\leqslant_{\mathrm{T}}^{\mathcal{NP}}$.

Finally, degree-theoretic questions about all these reducibilities remain, as well as questions about complete sets at various complexity levels.

Acknowledgments

We would like to thank Albert Meyer and Michael Machtey for some very valuable suggestions on this work.

References

[1] T. Baker, J. Gill and R. Solovay, Relativizations of the $\mathcal{P} = ?\ \mathcal{NP}$ question, SIAM J. Comput., to appear.

[2] S. A. Cook, The complexity of theorem-proving procedures, Third annual ACM Symposium on Theory of Computing (1971).

[3] J. E. Hopcroft and J. D. Ullman, Formal Languages and Their Relation to Automata (Addison-Wesley, Reading, Mass., 1969).

[4] N. D. Jones, Reducibility among combinatorial problems in log n space, Proc. of Seventh Annual Princeton Conference on Information Sciences and Systems (1973).

[5] R. M. Karp, Reducibility among combinatorial problems, in: Miller and Thatcher (eds.), Complexity of Computer Computations (Plenum Press, New York, 1973).

[6] R. E. Ladner, On the structure of polynomial time reducibility, J. ACM 22 (1975).

[7] N. A. Lynch, Relativization of the theory of computational complexity, Ph. D. Thesis, M. I. T. (1972).

[8] A. R. Meyer, Private communication.
[9] A. R. Meyer and L. J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, 13th Annual IEEE Symposium of Switching and Automata Theory (1972).
[10] A. R. Meyer and L. J. Stockmeyer, Word problems requiring exponential time, Fifth Annual Symposium on Theory of Computing (1973).
[11] E. L. Post, Recursively enumerable sets of positive integers and their decision problems, Bull. AMS 50 (1944).
[12] H. Rogers, Theory of Recursive Functions and Effective Computability (McGraw-Hill, New York, 1967).
[13] J. E. Savage, Computational work and time on finite machines, J. ACM 9 (4) (1972).