THE RELATIVE POWER OF LOGSPACE AND POLYNOMIAL TIME REDUCTIONS

Harry Buhrman, Edith Spaan
and Leen Torenvliet

Abstract. There exist many different formalisms to model the notion of resource bounded 'truth-table' reduction. Most papers in which truth-table reductions appear refer to the seminal paper of Ladner, Lynch and Selman for a definition. The definition of truth-table reductions given there however, pertains to only one of the possible incarnations of the notion. For larger resource bounds, this definition often coincides with other—often implicitly used—definitions of the notion. In this paper we focus our attention on resource bounds for which the various definitions probably do *not* pertain to the same notion. In particular, we show that coincidence of different notions implies coincidence of complexity classes like NC_1, LOG, and P, which are widely believed to be different.
Key words. reductions; logspace; circuits; branching programs; truth-table; polynomial time.
Subject classifications. 68Q05;68Q15

1. Introduction

Efficient reductions are a central object of study in computational complexity theory. Since the first use of polynomial time bounded Turing reductions by Cook (1971), and shortly thereafter the introduction of polynomial time bounded many-one reductions by Karp (1972), considerable effort has been put in the investigation of properties and relative strengths of different reductions. In 1975, an extensive survey of different types of polynomial time bounded reductions and differences between these reductions on exponential time bounded classes was given by Ladner *et al.* (1975). One of the main interests in reductions is that they give rise to the notion of completeness for a complexity class. In 1987, Watanabe (1987), building upon earlier work of Berman (1976), showed that different polynomial time bounded reductions gave rise to different classes of complete sets in EXP and E.

Logspace bounded reductions have long been studied as strengthenings of polynomial time bounded reductions. It was realized early on that many NP-complete sets (with respect to many-one polynomial time bounded reductions) were in fact many-one logspace complete for NP (Karp 1972, Jones 1975). Shortly after the initial work on the isomorphism conjecture by Berman & Hartmanis (1977), Hartmanis (1978) studied the question of the logspace isomorphism of NP-complete sets and achieved many of the results in Berman & Hartmanis (1977) for logspace reductions.

An important advantage of logspace reductions is that they give rise to the definition of complete sets for smaller complexity classes such as $NLOG$, CSL and P. The properties of such complete sets have consequences for parallel algorithms and the study of parallel complexity classes. Nonetheless, logspace reductions have received much less attention than have polynomial time reductions.

Buhrman *et al.* (1991) showed that the differences obtained by Watanabe for polynomial time bounded reductions on EXP could also be obtained for $NEXP$. Moreover, the constructions presented in that paper also suffice to demonstrate these differences for logspace bounded reductions. They also differentiated on completeness notions inferred by logspace bounded reductions on nondeterministic space bounded classes.

Truth-table reduction (in recursive function theory) was originally defined by Post (1944). A set A is *truth-table reducible to* B ($A \leq_{tt} B$) if given x, one can effectively compute a finite set $y_1, \ldots, y_k$ of words, with k depending on x, and a Boolean function α such that $x \in A$ if and only if $\alpha(\chi_B(y_1), \ldots, \chi_B(y_k)) = 1$. From this starting point, Ladner *et al.* defined polynomial time bounded truth-table reductions. We give an informal description here. For precise definitions, the reader is referred to the definitions in the next section or to Ladner *et al.* (1975). Suppose that for sets A and B, polynomial time computable algorithms g and e exist such that on input x algorithm g produces the encoding of a Boolean function and a sequence $y_1, \ldots, y_k$ of strings, and algorithm e computes the value of this Boolean function when the value of χ_B working on the i-th string in the sequence is substituted for the i-th parameter of the Boolean function. Suppose further that for all x, $x \in A$ if and only if e outputs 1. Then A is (polynomial time) truth-table reducible to B.

The 'Boolean function' in the Ladner *et al.* (1975) paper may be any 0-1 valued function. As they remark, for polynomial time bounds this characterization coincides with the characterization of truth-table reduction by *non-adaptive* oracle machines. In this paper we study another, frequently used, interpretation of 'Boolean function', and compare the reduction induced by this interpretation to the more general form of truth-table reduction. We assume that algorithm g generates a Boolean formula, and that algorithm e evaluates this formula. We show that for logspace bounded machinery, this interpretation is not likely to coincide with the characterization by oracle machines. It is clear that any Boolean formula truth-table reduction reduction between two sets can also be modeled by the non-adaptive oracle machine. We show that if the Boolean formula truth-table reduction *coincides* with the oracle machine reduction for logarithmic space bounds, then $NC_1^u = LOG$, which is widely believed to be false. As the oracle-machine characterization of truth-table reductions *does* coincide for logarithmic space with a characterization derived from polynomial size *branching* programs (Ladner & Lynch 1976), we will dub this reduction *branching program reduction (bp-tt).*

The rest of this paper is organized as follows. In the next section, we will first give some definitions and notations. In particular, we will give a rather extensive definition of different types of reductions. In Section 3, we will present our main result on the relation between different types of logarithmic space bounded reductions. In Section 4, we will discuss some of the consequences of this result. Finally in Section 5, we show that though probably different, the two notions cannot be arbitrarily far apart. In particular we show that a quadratic overhead on the space bound suffices for a Boolean formula truth-table reduction to simulate a Turing reduction.

2. Definitions and Notations

All strings are elements of Σ^*, where $\Sigma = \{0, 1\}$, and are denoted by small letters $x, y, u, v, \ldots$. For any string x, the length of x is denoted by $|x|$. For a string x of length n and $1 \leq i \leq n$, the i^{th} bit of x is denoted by $x[i]$. 0^n is the string $00\ldots0$ of length n. Languages are subsets of Σ^*, and are denoted by capital letters $A, B, C, \ldots$. Tally sets are subsets of $\{0\}^*$. For any set S, the cardinality of S is denoted by $\|S\|$. χ_S is the characteristic function of the set S. For a set S and integer n, we use the notation $S^{=n}$ ($S^{\leq n}$, etc.) as shorthand for $\{x : x \in S \wedge |x| = n\}$ ($\{x : x \in S \wedge |x| \leq n\}$, etc.). The notation $\{0\}^n$ ($\{0\}^{\leq n}$) may also stand for $\{0^n\}$ ($\{0^i : i \leq n\}$).

Propositional variables are variables that can take on values 0 or 1 (false, resp. true) only and are denoted by $p_i, p_j, \ldots$. Boolean formulae are strings consisting of literals, which are either a propositional variable p_i, the negation $\overline{p_i}$ of a propositional variable, or a constant, the connectives $\vee$ and $\wedge$, and parentheses. Boolean formulae are denoted by small Greek letters $\phi, \psi, \ldots$.

For a Boolean formula $\phi(p_1, \ldots, p_k)$, the notation $\phi(a_1, \ldots, a_k)$ stands for the value of the evaluation of the formula that is obtained by substituting a_i for p_i on each occurrence.

We adopt standard notations for complexity classes as can be found, e.g., in Balcázar *et al.* (1988) and Balcázar *et al.* (1990) or immediate extensions and variations of these classes. Examples are:

- $LOG/poly$ is the class of languages recognizable by deterministic logarithmic space bounded Turing machines using polynomial size advice.
- $LOGF$ is the class of functions computable by deterministic logarithmic space bounded transducers.
- NC_1 is the class of languages recognized by families of circuits with a polynomial number of gates of logarithmic depth. We also use the *uniform* version NC_1^u of this class. For a language L in NC_1^u, the existence of a logarithmic space bounded transducer is required that on input n computes the circuit that returns value 1 on inputs in (and only in) $L^{=n}$.

We assume a bijective pairing function $<\_, \ldots, \_>$ computable in logarithmic space and polynomial time—as for example described in Balcázar *et al.* (1988)—which makes it possible not only to encode pairs of strings into a single string, but also to encode sequences of *varying length* into a single string. We assume that the reader is familiar with the standard Turing machine and oracle machine as defined in Balcázar *et al.* (1988). An oracle machine M equipped with oracle A will be denoted M^A. An oracle machine is called *adaptive*, if it is allowed to compute queries based on the answers to previous queries obtained from the oracle, otherwise the machine is *non-adaptive*.

We adopt the definition of *branching programs* given by Boppana & Sipser (1990). A *deterministic branching program* is a directed acyclic graph in which all of the nodes have either out-degree 0 or out-degree 2. The nodes of out-degree 2 are labeled with a variable p_i, and the nodes of out-degree 0 are labeled with an output value. For each node of out-degree 2, one of the edges is labeled 0, and the other is labeled 1. One of the nodes is designated the start node. A setting of the inputs determines a path from the start node to an output node, giving an output value. A branching program *accepts* the input if this value is 1. The *size* of a branching program is the number of nodes.

0–1-valued functions appear in many places in the paper. (Turing machines as language acceptors, characteristic functions, etc.) To avoid confusion we shall avoid the term 'Boolean function', and use the term 'Boolean formula' to stand for a string consisting of 0-1 valued literals (variables, their negations and constants), the connectives $\vee$, $\wedge$, and parentheses, or the evaluation of such a string in case the string has no variables.

A tuple $<\alpha, y_1, \ldots, y_k>$ is a *truth-table condition of norm* k if $<y_1, \ldots, y_k>$ is a k-tuple of strings, and α is a k-ary 0–1 valued function. A function f is a *truth-table function* if f is total and $f(x)$ is a truth-table condition for every x in Σ^*. We say that a truth-table function (tt-function) is a:

- *Boolean formula* truth-table function (bf-tt function) if and only if $f(x) = <\phi_x, y_{x,1}, \ldots, y_{x,k(x)}>$ and ϕ_x is a Boolean formula for every x.
- *branching program* truth-table function (bp-tt function) if and only if $f(x) = <P_x, y_{x,1}, \ldots, y_{x,k(x)}>$ and P_x is (the encoding of) a branching program for every x.

Let b be a resource bound (e.g., logarithmic space or polynomial time). Let $A_1, A_2 \subseteq \Sigma^*$, let f be a b-computable truth-table function, where $f(x) = <g_x, y_{x,1}, \ldots, y_{x,k(x)}>$. We say that:

1. A_1 is b generate-evaluate truth-table (ge-tt) reducible to A_2 if and only if there exists a Turing machine M such that $x \in A_1$ if and only if M accepts $<g_x, \chi_{A_2}(y_{x,1}), \ldots, \chi_{A_2}(y_{x,k(x)})>$, within resource bound $b(|x|)$. (Ladner *et al.* 1975) Notation: $A_1 \leq^b_{ge-tt} A_2$.
2. A_1 is b Boolean formula truth-table (bf-tt) reducible to A_2 if and only if f is a bf-tt function, and $x \in A_1$ if and only if $g_x(\chi_{A_2}(y_{x,1}), \ldots, \chi_{A_2}(y_{x,k(x)}))$ $= 1$. Notation: $A_1 \leq^b_{bf-tt} A_2$.
3. A_1 is b branching program truth-table (bp-tt) reducible to A_2 if and only if f is a bp-tt function, and $x \in A_1$ if and only if g_x accepts $y = \chi_{A_1}(y_{x,1}) \ldots \chi_{A_2}(y_{x,k(x)})$. Notation: $A_1 \leq^b_{bp-tt} A_2$.
4. A_1 is b *non-adaptive* (na-tt) truth-table reducible to A_2 if and only if there exists a b bounded non-adaptive oracle machine M such that $x \in A_1$ if and only if M^{A_2} accepts x. Notation: $A_1 \leq^b_{na-tt} A_2$.
5. A_1 is b Turing (T) reducible to A_2 if and only if there exists a b-bounded (adaptive) oracle machine M such that $x \in A_1$ if and only if M^{A_2} accepts x. Notation: $A_1 \leq^b_T A_2$.

In this paper, we shall in particular prove theorems on logarithmic space and/or polynomial time bounded reductions. For these resource bounds, the notions 1 and 4 coincide. Therefore, we shall use the notation $\leq^b_{tt}$ for both. Furthermore, it is not hard to verify that the notions 2 and 3 imply 1 and 4, which in turn imply 5. For brevity, a reduction of type r with resource bounds logarithmic space or polynomial time will be denoted $\leq_r^L$ and $\leq_r^P$ respectively.

3. Main Result

In this section, we prove our main theorem on the relation between different types of logarithmic space bounded reductions. We show that logarithmic space bounded Boolean formula truth-table reductions cannot be used intermixedly with logarithmic space bounded Turing reductions unless $NC_1^u = LOG$, which is very unlikely. This is all the more worth cautioning, since logarithmic space bounded Turing reductions *can* be used intermixedly with logarithmic space bounded branching program truth-table reductions as follows from the proof of Theorem 1 (Theorem 4.2 in Ladner & Lynch 1976).

THEOREM 1. (LADNER & LYNCH 1976) *For all A and $B \subseteq \{0,1\}^*$: $A \leq_{tt}^L B$ if and only if $A \leq_T^L B$.*

In the proof, Ladner and Lynch observe that the entire computation tree of a logspace bounded oracle machine, i.e., the tree of all possible computations, generated independently of any oracle, can contain only a polynomial number of different queries. It is not hard to see, although Ladner and Lynch do not say this in so many words, that the graph obtained by identifying nodes of this tree that stand for the same query is precisely equivalent to a polynomial size branching program. Hence, Ladner and Lynch actually prove the following theorem.

THEOREM 2. *For all sets A and $B \subseteq \{0,1\}^*$: $A \leq_{bp-tt}^L B$ if and only if $A \leq_T^L B$.*

Ladner and Lynch in fact *do* make distinction between Boolean formula truth-table reductions and (general) truth-table reductions. They observe that truth-table reductions characterized by non-adaptive Turing machines are at least as powerful as Boolean formula truth-table reductions. However they state: "We do not know whether or not $\leq_{bf-tt}^L$ and $\leq_{tt}^L$ are equivalent". Our main theorem can therefore be viewed as a partial answer to their open problem. We state our theorem in the uniform version. Later we will remark that one side of the equivalence has a nonuniform extension.

THEOREM 3. *$NC_1^u = LOG$ if and only if $\leq_{bf-tt}^L = \leq_T^L$*

We first prove that $NC_1^u \neq LOG$ implies that $\leq^L_{bf-tt} \neq \leq^L_T$.
PROOF. Let A be any set in $LOG - NC_1^u$. Then, for any tally set T, define the tally set U_T as follows:

$$U_T = \{0^n : \chi_T(0^{n+1}) \ldots \chi_T(0^{2n}) \in A\}.$$

We will show by direct construction the existence of a tally set D such that $U_D \not\leq^L_{bf-tt} D$.

The main effort in this part of the proof is to show that for any *particular* bf-tt function, we can find arbitrarily large n that witness the failure of this bf-tt function as a candidate reduction. We say that a number n *witnesses* the fact that $U_T \not\leq^L_{bf} T$ via f, if either $f(0^n)$ is not of the form $<\phi, 0^{b_1}, \ldots, 0^{b_c}>$, i.e., $f(0^n)$ is not a Boolean formula truth-table condition and hence f is not a bf-tt reduction, or $0^n \in U_T \iff \phi(\chi_T(0^{b_1}), \ldots, \chi_T(0^{b_c})) = 0$. We will isolate and prove this statement in the following lemma.

LEMMA 4. *For all $f \in LOGF$, for all $k \in \mathbb{N}$, and tally sets $T_0 \subseteq \{0\}^{\leq k}$, there exists an $n > k$ and a tally set $T_1 \subseteq \bigcup_{n<i\leq 2n}\{0^i\}$, such that 0^n witnesses $U_{T_0 \cup T_1} \not\leq^L_{bf-tt} T_0 \cup T_1$ via f.*

PROOF. Suppose that there exists an $f \in LOGF$ such that f outputs the truth-table condition $<\phi_n, 0^{b_1}, \ldots, 0^{b_{c(n)}}>$ on input 0^n, and for certain k and tally set $T_0 \subseteq \{0\}^{\leq k}$, it holds for any $n > k$ and for any tally set $T_1 \subseteq \bigcup_{n<i\leq 2n}\{0\}^i$ that $0^n \in U_{T_0\cup T_1}$ if and only if $\phi_n(0^{b_1}, \ldots, 0^{b_{c(n)}}) = 1$. Since for $i \notin \{n+1, \ldots, 2n\}$, the value of $\chi_{T_0\cup T_1}(0^{b_i})$ is either 0 or $\chi_{T_0}(0^{b_i})$, we infer that a Boolean formula $\phi'(p_1, \ldots, p_n)$ can be derived from ϕ_n such that for each x of length n, $\phi'(x[1], \ldots, x[n])$ is true if and only if $x \in A$. Moreover, since $f \in LOGF$, the length of $f(0^n)$ and more in particular the size of ϕ' is polynomial in n. From a result of Spira (1971), we infer that f generates an encoding of an NC_1 circuit C such that C outputs 1 if and only if the inputs of C are set to the bits of x for some $x \in A$ of length n. This contradicts our assumption on A. □

The proof of the right-to-left implication is now simply putting things together. Let $f_1, f_2, \ldots$ be an enumeration of functions in $LOGF$. Let T be a finite tally set, and let $b_1, \ldots, b_{n-1}$ be a sequence of numbers that witness $U_T \not\leq^L_{bf-tt} T$ via f_i, for $1 \leq i < n$. Choose m so large that $\max\{|f_i(0^{b_i})|\} < m$. Lemma 4 states that we can find a tally set T' and $m' > m$, with $T' \subseteq \bigcup_{m'<j\leq 2m'}\{0\}^j$ such that $U_{T\cup T'} \not\leq^L_{bf-tt} T \cup T'$ via f_n. Let $b_n = m'$. Then, by choice of m, the numbers in $b_1, \ldots, b_n$ witness that $U_{T\cup T'} \not\leq^L_{bf-tt} T \cup T'$ via any f_i, for $i \leq n$. □

Next we show that $NC_1^u = LOG$ implies $\leq_{bf-tt}^L = \leq_T^L$.

PROOF. Let A and B be sets such that $A \leq_T^L B$. Let machine M witness the $\leq_T^L$ reduction from A to B. Machine M generates on input x, in space $c\log|x|$, a sequence of queries $q_{x,1}, \ldots, q_{x,k(x)}$, where $k(x) \leq |x|^c$. (Since the difference between adaptive and nonadaptive oracle fades with logarithmic space bounds, we may indeed assume that the sequence is generated from the input alone.) Let a string y, of length at most $|x|^c$, be called an *answer sequence* for M on input x if the computation of M on input x is simulated with $y[i]$ substituted for the answer on query $q_{x,i}$. (That is, we assume a yes answer if and only if this bit is 1.) Consider the following set:

$$D = \{<x, y> : M \text{ accepts } x \text{ with answer sequence } y\}.$$

Then $D \in LOG$. Suppose that $D \in NC_1^u$. Then there exists a family of NC_1 circuits (and hence a family of polynomial size Boolean formulas) that recognize D, and that can be generated by a $LOGF$ transducer. The following machine is now a bf-tt reduction from A to B: On input x simulate machine M on input x to determine the *number* of queries. Let this number be $k(x)$. Since $D \in NC_1^u$, it is possible to construct in space $c\log n$ a formula $\phi_x(p_1, \ldots, p_{k(x)})$ such that $\phi_x(a_1, \ldots, a_{k(x)})$ is true if and only if $<x, a_1 \ldots a_{k(x)}> \in D$. Since $\phi_x(\chi_B(q_{x,1}), \ldots, \chi_B(q_{x,k(x)}))$ is true if and only if $x \in A$, the bf-tt reduction is then $<\phi_x, q_{x,1}, \ldots, q_{x,k(x)}>$. $\square$

4. Some Consequences

We will look at some consequences of the theorem derived in the previous section. First of all, it is not hard to see that in the nonuniform case, a difference between NC_1 and $LOG/poly$, that is a set A in $LOG/poly - NC_1$, would immediately imply a set in $LOG - NC_1^u$, and therefore the following nonuniform extension of our theorem holds.

COROLLARY 5. *If $NC_1 \neq LOG/poly$ then $\leq_{bf-tt}^L \neq \leq_T^L$.*

Next, polynomial time generated Boolean formulas are also of polynomial size and hence correspond to NC_1-circuits (by the same result of Spira (1971)). Furthermore, logspace generatable branching programs can also be generated in polynomial time. We obtain the following corollary to the first implication in the proof of Theorem 3 and Corollary 5.

COROLLARY 6. *If $NC_1 \neq LOG/poly$, then $\leq_{bf-tt}^P \neq \leq_{tt}^P$.*

As another immediate consequence of our proof technique, we can prove the following theorem.

THEOREM 7. *The following are equivalent:*

1. $P = LOG$,
2. $\leq^P_{bf-tt} = \leq^L_{bf-tt}$ and $\leq^P_{bp-tt} = \leq^L_{bp-tt}$ and $\leq^P_{tt} = \leq^L_{tt}$,
3. $\leq^P_{tt} = \leq^L_{bp-tt}$.

PROOF. (1⇒ 2) From standard argumentation on the relation between language and function classes, it follows that if $P = LOG$ then $PF = LOGF$. (cf. Balcázar *et al.* 1988). From this, 2 follows immediately.
(2⇒3) By assumption we have that $\leq^P_{tt} = \leq^L_{tt}$. As quoted before, Ladner & Lynch (1976) observed that $\leq^L_{tt} = \leq^L_{bp-tt}$, and by assumption $\leq^L_{bp-tt} = \leq^P_{bp-tt}$.
(3⇒1) We prove this by showing that if $P \neq LOG$ then $\leq^P_{tt} \neq \leq^P_{bp-tt}$. The proof is a straightforward application of the proof of Theorem 3. We take a set $A \in P - LOG$ and define for each tally set T the set U_T as in the proof of Theorem 3 i.e., $U_T = \{0^n : \chi_T(0^{n+1}) \dots \chi_T(0^{2n}) \in A\}$. Now a machine witnessing $U_T \leq^P_{tt} T$ works as follows: First query $0^{n+1}, \dots, 0^{2n}$ to form the bits of a string y. Then accept if and only if $y \in A$. Since $A \in P$, machine M works in polynomial time. On the other hand, Lemma 4 can be translated to the P-case in a straightforward way, since (uniformly generated) polynomial size branching programs can be simulated by logarithmic space bounded Turing machines (Boppana & Sipser 1990 Theorem 6.3). With the same sort of construction we now construct a set D such that $U_D \not\leq^P_{bp-tt} D$. □

Different types of truth-table reductions (i.e., Boolean formula, branching program, and truth-table) coincide only if classes that are believed to be different coincide. As a consequence of this, it is important in the future to specifically state which notion of truth-table reduction is used. The converse of these theorems, namely that classes coincide if and only if certain types of reductions coincide can also be viewed as 'positive relativization results'. A *positive relativization* is not a formal concept, but an intuitive property of complexity classes. We might paraphrase the extensive survey on this subject by Balcázar *et al.* (1990) as follows: Suppose that C and D are complexity classes, and that there is some restriction on the oracle machine model such that $C^A = D^A$ for *every oracle* A if and only if $C = D$. Then this restriction is called a *positive* relativization. Our theorems now give positive relativizations for the following questions on the relation between complexity classes.

- $P \stackrel{?}{=} LOG$ (With restriction: non-adaptive queries.)
- $NC_1^u \stackrel{?}{=} LOG$ (With restriction: a special model for circuit-oracle access.)
- $NC_1^u \stackrel{?}{=} P$ (With restriction; a special model for circuit oracle access *and* non-adaptive queries.)

This means that by the oracle model induced by the appropriate reduction, these complexity classes are equal if and only if they are equal relative to every oracle.

5. An upper bound

We observed that it is very unlikely that a Boolean formula can be generated by a logarithmic space bounded transducer such that this Boolean formula exactly characterizes the behavior of a logspace-bounded Turing reduction. The question may be raised, which (if any) space bound is sufficient to produce a formula that does characterize a Turing reduction. A (rather straightforward) application of the well-known construction of Savitch (1970) will show this claim for any $S(n)$-space bounded oracle machine M that witnesses $A \leq_T^{space(S(n))} B$ and input x. A formula ϕ and strings $y_1, \ldots, y_{k(x)}$ can be produced such that $\phi(\chi_B(y_1), \ldots, \chi_B(y_{k(x)}))$ is true if and only if $x \in A$. Moreover ϕ and $y_1, \ldots, y_{k(x)}$ can be generated on input x by an $S^2(n)$-space bounded transducer.

THEOREM 8. *Let $S(n) \geq \log n$ be any space-constructible function. For any sets A and B: If $A \leq_T^{space(S(n))} B$ then $A \leq_{bf-tt}^{space(S^2(n))} B$*

PROOF. Let $A = L(M^B)$ with M an $S(n)$ space-bounded oracle machine. The number of different configurations of M on a given input of length n is bounded by $2^{c \cdot S(n)}$. Let $\{I_1, \ldots, I_{2^{c \cdot S(n)}}\}$ be the set of all possible configurations. Each of these configurations I_j identifies a query string q_j as follows: Machine M, started in configuration I_j after some amount of computation either reaches a final state without querying the oracle, or reaches a state $I_{j'}$ of which the state part is QUERY. Meanwhile some string y (possibly λ) is written on the oracle tape. We let q_j be this string.

We will define a $S^2(n)$-space bounded transducer that produces on input x a tuple $<\psi, q_1, \ldots, q_{2^{c \cdot S(n)}}>$ such that $\psi(\chi_B(q_1), \ldots, \chi_B(q_{2^{c \cdot S(n)}}))$ is true if and only if M^B accepts x. For configuration I, let I^Y, I^N, I^Q denote the same configuration, but with the state part replaced respectively by YES, NO, and QUERY. (Then $I = I^Q$ is short for "the state of I is QUERY".)

Let us consider tuples consisting of two configurations, a Boolean constant and an integer $<I_j, I_k, b, i>$, for which the state of I_j is either the initial state or YES or NO, and the state of I_k is either accept, reject or QUERY. We will define a Boolean formula $\phi(I_j, I_k, b, i)$ corresponding to this pair.

First let $i = 0$. If M, started in configuration I_j, does not reach I_k *without querying the oracle* then $\phi(I_j, I_k, b, 0) = 0$. Otherwise, if M started in configuration I_j reaches I_k without querying the oracle, and $I_k \neq I_k^Q$, then $\phi(I_j, I_k, b, 0) = 1$. Otherwise, if $I_k = I_k^Q$, we let $\phi(I_j, I_k, b, 0) = p_j$, if $b = 1$ and $\phi(I_j, I_k, b, 0) = \overline{p_j}$, if $b = 0$.

For $i > 0$, we define the Boolean formula corresponding to this pair as

$$\begin{aligned} \phi(I_j, I_k, b, i) \quad = \quad & \bigvee_{I=I^Y} \phi(I_j, I^Q, 1, \lfloor i/2 \rfloor) \wedge \phi(I^Y, I_k, b, \lfloor i/2 \rfloor) \vee \\ & \bigvee_{I=I^N} \phi(I_j, I^Q, 0, \lfloor i/2 \rfloor) \wedge \phi(I^N, I_k, b, \lfloor i/2 \rfloor) \end{aligned}$$

Finally for the set F of accepting configurations, and the initial configuration I_0, we define $\psi = \bigvee_{I \in F} \phi(I_0, I, 0, 2^{c \cdot S(n)})$.

If substitution of $\chi_B(q_j)$ for variable p_j in ψ makes ψ true, then one of the disjunctions in ψ is true. This is a conjunction $1 \wedge p_{i_1} \wedge 1 \wedge p_{i_2} \wedge \ldots \wedge p_{i_m} \wedge 1$ such that there exist tuples $<I_j, I_k, b, 0>_{i_j}$ such that I_j in $<I_j, I_k, b, 0>_{i_1}$ is the initial configuration, I_k in $<I_j, I_k, b, 0>_{i_m}$ is an accepting configuration, and p_{i_ℓ} is a (negated) variable that corresponds to the YES/NO state in the subsequent tuple. This formula evaluates to true if the right values are substituted. It is clear that the computation of M^B on input x is then accepting.

The program that produces ψ is obtained by carefully mimicking Savitch's construction. As there are, however, no technical caveats left in this construction, we leave details to the reader. □

Acknowledgements

We are grateful to Steve Mahaney and two anonymous referees for their many comments and suggestions. These comments led us to rewrite rather than revise the paper. In our view the paper has greatly benefited from that action. Finally, we would like to thank Peter van Emde Boas for carefully proofreading the rewritten manuscript.

References

J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity I.* Springer-Verlag, 1988.

J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity II.* Springer-Verlag, 1990.

L. Berman, On the structure of complete sets: Almost everywhere complexity and infinitely often speedup. *Proc. 17th IEEE Symposium on Foundations of Computing* (1976), 76–80.

L. Berman and H. Hartmanis, On isomorphisms and density of NP and other complete sets. *SIAM J. Comput.* **6** (1977), 305–322.

R.B. Boppana and M. Sipser, The complexity of finite functions. In *Handbook of theoretical computer science*, ed. J. van Leeuwen, vol. A Algorithms and Complexity, 757–804. Elsevier, Amsterdam.New York.Oxford.Tokyo, 1990.

H. Buhrman, S. Homer, and L. Torenvliet, On complete sets for nondeterministic classes. *Math. Systems Theory* **24** (1991), 179–200.

S. Cook, The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symposium Theory of Computing*, Shaker Heights, Ohio, 1971, 151–158.

J. Hartmanis, On the logtape isomorphism of complete sets. *Theoretical Computer Science* **7** (1978), 273–286.

N. Jones, Space bounded reducibilities among combinatorial problems. *J. Comput. System Sci.* **11** (1975), 68–85.

R. Karp, Reducibility among combinatorial problems. In *Complexity of Computer Computations*, ed. R. Miller and J. Thatcher, 85–103. Plenum Press, New York, 1972.

R.E. Ladner and N. Lynch, Relativization of questions about logspace computability. *Math. Systems Theory* **10** (1976), 19–32.

R. Ladner, N. Lynch, and A. Selman, A comparison of polynomial time reducibilities. *Theoret. Comput. Sci.* **1** (1975), 103–123.

E. Post, Recursively enumerable sets of integers and their decision problems. *Bull. Amer. Math. Soc.* **50** (1944), 284–316.

W.J. Savitch, Relationships between nondeterministic and deterministic tape complexities. *J. Comput. System Sci.* **4**(2) (1970), 177–192.

P.M. Spira, On time-hardware complexity tradeoffs for Boolean functions. In *Proc. 4th Hawai Symp. on System Sciences*, North Hollywood, 1971, Western Periodicals Company, 525–527.

O. WATANABE, A comparison of polynomial time completeness notions. *Theoret. Comput. Sci.* **54** (1987), 249–265.

Manuscript received April 1, 1991

HARRY BUHRMAN
Department of Computer Science
University of Amsterdam
Plantage Muidergracht 24
1018 TV Amsterdam, The Netherlands
`buhrman@fwi.uva.nl`

EDITH SPAAN
Department of Computer Science
SUNY at Buffalo
226 Bell Hall, Buffalo,
NY 14260 USA
`spaan@cs.buffalo.edu`

LEEN TORENVLIET
Department of Computer Science
University of Amsterdam
Plantage Muidergracht 24
1018 TV Amsterdam, The Netherlands
`leen@fwi.uva.nl`