Satisfiability Is Quasilinear Complete in NQL

C. P. SCHNORR

Universitat Frankfurt, Frankfurt am Main, West Germany

ABSTRACT Considered are the classes QL (*q*uasi*l*inear) and NQL (*n*ondet *q*uasi*l*inear) of all those problems that can be solved by deterministic (nondeterministic, respectively) Turing machines in time $O(n(\log n)^k)$ for some k Efficient algorithms have time bounds of this type, it is argued. Many of the "exhaustive search" type problems such as satisfiability and colorability are complete in NQL with respect to reductions that take $O(n(\log n)^k)$ steps This implies that QL = NQL iff satisfiability is in QL

KEY WORDS AND PHRASES NP-complete problems, nondeterministic Turing machines, logical networks, satisfiability, colorability, graph isomorphism, clique problem

CR CATEGORIES: 5.25

1. *Introduction*

There is a common agreement that the classes P and NP of all decision problems which can be solved in polynomial time by deterministic (nondeterministic, respectively) Turing machines are basic classes for the complexity classification of natural problems. Recently, much attention has been attracted by the question of whether P = NP? and by the class of polynomial complete problems in NP; see Cook [1], Karp [5], and Levin [7].

Here we introduce two similar classes of problems. Instead of polynomial time bounds we consider time bounds for multitape Turing machine computations of the type $O(n(\log n)^k)$ with k fixed These time bounds are called *quasilinear* in n. Let QL (*q*uasi*l*inear) and NQL (*n*ondet.*q*uasi*l*inear) be the classes of all decision problems which can be solved by deterministic (nondeterministic, respectively) multitape Turing machines within quasilinear time bounds. These classes are machine independent to a certain extent. Within the framework of multitape Turing machines they do not depend on the number of tapes, the number of heads per tape, and the size of the alphabet provided that there are at least two tapes and two alphabet symbols.

From the recursion-theoretic point of view, these classes are reasonable, too. Diagonalization can be applied in a standard manner within these classes. So it follows from Hennie and Stearns [4] that the classes QL_k of all those problems which are solvable within time bound $O(n(\log n)^k)$ by some multitape Turing machine yield an infinite hierarchy within QL, i.e. $QL = \bigcup_k QL_k$ but $QL \neq \bigcup_{k \leq k_0} QL_k$ for any k_0. It is also clear how to construct for given k a set $A \in QL$ such that every deterministic Turing machine which decides A takes more than $|x|\,(\log|x|^k)$ steps on all but finitely many input strings x. Here $|x|$ is the length of x.

Quasilinear time bounds are an important landmark in the field of concrete complexity They express the fact that a program works efficiently whereas polynomial time

The results of this paper were part of a main lecture at the annual meeting of the Gesellschaft fur Angewandte Mathematik und Mechanik, Gottingen, West Germany, April 2-5, 1975

Author's present address. Johann Wolfgang Goethe Universitat, Fachbereich Mathematik, Robert-Mayer-Strasse 6-10, 6 Frankfurt am Main, West Germany

bounds merely say that a program is feasible. Many fundamental problems, such as integer multiplication, sorting, and searching, can be done in quasilinear time. We prove that many of the "exhaustive search" type problems, such as satisfiability, 3-colorability, and graph isomorphism, can be solved in nondeterministic quasilinear time, i.e. they are in the class NQL.

Using an efficient simulation of Turing machines by logical networks which is based on the Fischer-Pippenger technique, we can prove that satisfiability is quasilinear complete in NQL, i.e. every problem in NQL can be reduced to satisfiability by a deterministic Turing machine with a quasilinear time bound. This result gives more information on the relation between satisfiability and nondeterminism than Cook's theorem that satisfiability is polynomial complete in NP. It reveals that satisfiability lies on the bottom within the "exhaustive search" type problems. Note that each quasilinear complete problem in NQL is polynomial complete in NP; however, the converse is not true. The hierarchy results on nondeterministic classes [2] imply that there are polynomial-complete problems in NP which are not in NQL. We conclude that satisfiability expresses the feature of nondeterminism as purely as possible and that, to understand the feature of nondeterminism, we should study quasilinear complete problems in NQL rather than arbitrary polynomial complete problems in NP.

The quasilinear completeness of satisfiability implies that QL = NQL if and only if satisfiability is in QL. This QL = NQL? problem seems to be as fundamental as Cook's P = NP? problem. Obviously P ≠ NP implies QL ≠ NQL but the converse is not clear. Since we expect P ≠ NP and QL ≠ NQL it might well be that "QL ≠ NQL" will be proved first. It would be sufficient to prove lower time bounds for satisfiability which are slightly higher than quasilinear.

2. *The Class NQL of Problems Solvable in Nondeterministic Quasilinear Time*

We consider Turing machines with a finite number of tapes over a finite alphabet Σ. Let Σ^* be the set of all finite sequences over Σ. $|x|$ is the length of $x \in \Sigma^*$. Let $K = \{0, 1\} \subset \Sigma$ be the binary input-output alphabet of the Turing machines under consideration.

With any Turing program (i.e. Turing table) we associate a partial function $\text{res}_p : K^* \to K^*$ which is computed by p. Let

$$T_p(n) = \max_{x \in K^n} \{\text{running time of program } p \text{ on input } x\}$$

be the time bound of program p. Then we consider the class

$$\text{QL}' = \{\text{res}_p \mid \exists k : \forall n : T_p(n) \leq n(\log n)^k + k\}$$

of all functions that are computable by a Turing program in quasilinear time. Let

$$\text{QL} = \{A \subset K^* \mid \chi_A \in \text{QL}'\}$$

be the corresponding class of decision problems that are solvable in quasilinear time on Turing machines. Here χ_A is the characteristic function of set A.

We shall compare the time class QL with the corresponding nondeterministic time class. In a nondeterministic Turing program each instruction may have one or two successor instructions. If all successor instructions are carried out in parallel then the computational process of a nondeterministic program p on input x can be figured as the binary tree shown in Figure 1. With a nondeterministic program p we associate the set

$$\text{Acc}_p = \{x \in K^* \mid \text{ there exists a stop-path in the computation of program } p \text{ on input } x\}$$

of all words x such that some path in the nondeterministic computation on input x reaches a final configuration. Such a path is called a stop-path.

For $x \in \text{Acc}_p$ we define the running time $\text{RT}_p(x)$ as

$$\text{RT}_p(x) := \text{minimal length of a stop-path of program } p \text{ on input } x.$$

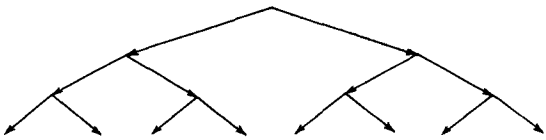

FIG 1

Then we are ready to introduce the class

$$\mathrm{NQL} = \{\mathrm{Acc}_p \subset K^* | \exists k : \forall x \in \mathrm{Acc}_p : \mathrm{RT}_p(x) \leq |x|(\log |x|)^k + k\}$$

of all decision problems that are solvable in nondeterministic, quasilinear time on Turing machines.

With a sequence $x = x_1x_2 \cdots x_n \in K^n$ we associate the double-sequence $\bar{x} = x_1x_1x_2x_2 \cdots x_nx_n01$. We use the following characterization of NQL:

PROPOSITION 2.1. *The following assertions are equivalent*:

(1) $A \in \mathrm{NQL}$,

(2) $\exists B \in \mathrm{QL} : \exists k \in N$:

$$A = \{x \in K^* | \exists u : \bar{u}x \in B \wedge |u| = |x|\lceil \log |x| \rceil^k + k\}.$$

PROOF. Consider the binary sequence u as a sequence of branching parameters.

(2) $\Rightarrow$ (1): A representation of $A \subset K^*$ as in (2) yields a nondeterministic decision procedure for A as follows:

1 Compute $\alpha_k(x) = |x|\lceil \log |x| \rceil^k + k$

2 Make a nondeterministic choice of $\alpha_k(x)$ branching parameters $u = u_1u_2 \cdots u_{\alpha_k(x)}$

3. Apply the decision procedure for B to the sequence $\bar{u}x$.

It is a straightforward matter to see that this procedure can be implemented as a nondeterministic Turing program with a quasilinear time bound.

(1) $\Rightarrow$ (2): Let p be a nondeterministic program for A with time bound $|x|\lceil \log |x| \rceil^k + k$. With p we associate the deterministic program $\tilde{p}$ which on input $\bar{u}x$ simulates p on input x for $|u|$ steps and which uses the sequence u as branching parameters, i.e. u_i describes the binary choice within the ith nondeterministic step of program p. For each u there is a corresponding path in the computation of p on input x. Program $\tilde{p}$ can be executed in quasilinear time and $\tilde{p}$ is a decision program for some set B which yields a representation for A as in assertion (2). □

Next we show that various famous problems are in NQL. We need some preliminaries on binary encodings. Indeed we have to be careful in choosing reasonable encodings. One can associate small time bounds to any problem A by encoding the inputs into extremely long binary strings. However, this makes it more difficult to reduce other problems efficiently to A. Therefore we cannot force any problem A to be quasilinear complete in NQL by choosing some pathological encoding for A. To obtain strong completeness results we must choose the binary encoding as short as possible without making the encoding inefficient.

In the following let $[n] = \{1, \ldots, n\}$.

Let $L = (l_i \in N | i \in [m])$ be a list (i.e. sequence of natural numbers $l_1, \ldots, l_m$).Then the binary encoding $C(L) \in K^*$ is defined as

$$C(L) = \overline{c(l_1)}\,\overline{c(l_2)} \cdots \overline{c(l_m)},$$

where $c(\nu)$ is the binary representation of $\nu \in N$ and $\bar{x}$ is the "doubled" sequence which is associated with x. Let $\Delta(L) \subset N$ be the set of elements of L and let $|L| = m$ be the number of elements in the list L.

Let $\mathscr{L} = (L_i | i \in [m])$ be a list of lists $L_1, \ldots, L_m$, i.e. $\mathscr{L}$ is a two-dimensional list. Then the encoding $C(\mathscr{L}) \in K^*$ of $\mathscr{L}$ is defined as

$$C(\mathcal{L}) = C(L_1)01C(L_2)01 \cdots 01C(L_m).$$

A directed graph G with the set $[n]$ of nodes and the set $E \subset [n] \times [n]$ of edges is considered to be given as a two-dimensional list $\mathcal{L} = (L_i | i \in [n])$ such that $\Delta(L_i) = \{j | (i, j) \in E\}$ for $i = 1, \ldots, n$. Then $C(\mathcal{L}) \in K^*$ is a binary encoding of the graph G.

The isomorphism problem for graphs is the decision problem for the set

$$\text{graph-iso} = \{C(G)10C(\bar{G}) | \text{the graphs } G \text{ and } \bar{G} \text{ are isomorphic.}\}$$

Here $C(G)$ and $C(\bar{G})$ are the encodings of isomorphic graphs iff G and $\bar{G}$ have the same number m of nodes (i.e. $C(G)$ and $C(\bar{G})$ both consist of m lists for some m) and if there exists a permutation $\sigma:[m] \to [m]$ such that $\Delta L_i = \sigma(\Delta \bar{L}_{\sigma(i)})$ for $i = 1, \ldots, m$, with L_i and $\bar{L}_i$ being the lists of G and $\bar{G}$.

THEOREM 2.1. *graph-iso is in NQL.*

The nondeterministic procedure for graph-iso will use some subprograms which will also be useful later on. We describe various subprograms.

Program p_1
Input: $C(L)$ (encoding of a list L of natural numbers)
Output. $C(L^o)$ (encoding of the ordered list L^o)
Properties. (1) $|L| = |L^o|$ and L^o is a permutation of L, (2) $l_j^o \le l_{j+1}^o$, $j = 1, \ldots, |L| - 1$
Time bound quasilinear in the length of the input

PROOF. In stage i we sort the segments

$$L(k, i) = \{l_j | k2^i \le j < k2^i + 2^i\} \quad \text{for} \quad k \le |L|/2^i$$

by merging the ordered segments $L(2k, i-1)$, $L(2k+1, i-1)$ which have been generated at stage $i-1$. This requires $O(\log|L|)$ stages beginning with stage 1. Since merging of two lists can be done within linear time with three tapes, each stage can be done within $O(|C(L)|)$ steps. □

Let $L = (l_i | i \in [n])$ be a list and let $\sigma:[n] \to [n]$ be a permutation; then we define L^σ as $L^\sigma = (l_{\sigma(i)} | i \in [n])$. The encoding $C(\sigma)$ of a function $\sigma:[n] \to N$ is the encoding of the list $(\sigma(i) | i \in [n])$.

Program p_2.
Input. $C(L)10C(\sigma)$, where $L = (l_i | i \in [n])$ is a list and σ $[n] \to [n]$ is a permutation.
Output $C(L^\sigma)$
Time bound: quasilinear in the length of the input

Description of p_2. Sort σ and apply the same transformation simultaneously to the list $(i | i \in [n])$. This yields σ^{-1}. Sort σ^{-1} and apply the same transformation simultaneously to the list L. This yields L^σ. □

Let $\Delta\mathcal{L}$ be the set of elements of a two-dimensional list $\mathcal{L}$, i.e. $\Delta\mathcal{L}$ consists of the elements of the lists in $\mathcal{L}$. Let $\gamma:\Delta\mathcal{L} \to N$ be a function. Then $\gamma(\mathcal{L})$ is the two-dimensional list which is obtained from $\mathcal{L}$ by replacing each element ν of $\mathcal{L}$ by $\gamma(\nu)$.

Program p_3:
Input: $C(\mathcal{L})10C(\gamma)$ where $\Delta(\mathcal{L}) = [n]$ and γ $[n] \to N$ is some function
Output: $C(\gamma(\mathcal{L}))$
Time bound quasilinear in the length of the input
Sketch of p_3

1. Transform $\mathcal{L}$ into a one-dimensional list L by "forgetting" the two-dimensional structure of $\mathcal{L}$ Let m be the length of L

2 Sort L into an ordered list L^σ preserving the multiplicity of elements Apply the same permutation simultaneously to the unit list $(i | i \in [m])$. This yields the permutation σ^{-1}

3 Compute $\gamma(L^\sigma)$ within one pass over the lists L^σ and γ

4 Compute $L = (\gamma(L^\sigma))^{\sigma^{-1}}$ by inverting the permutation σ This can be done by sorting the list σ and a simultaneous application of the same transformation to $\gamma(L^\sigma)$

5 Construct $\gamma(\mathcal{L})$ by implementing the two-dimensional structure of $\mathcal{L}$ to L This can be done within one pass over $\mathcal{L}$ and L

We are now ready to prove Theorem 2.1.

Sketch of a nondeterministic program for graph-iso

1. Decide whether the input x is of the type $C(G)01C(\bar{G})$ where $C(G)$, $C(\bar{G})$ are binary encodings of directed graphs. In this case goto 2

2 Count the number of nodes n and $\bar{n}$ of G and $\bar{G}$ This can be done by counting the number of lists of G and $\bar{G}$ If $n = \bar{n}$ then goto 3

3 Make a nondeterministic choice of a list $\sigma = (\sigma(i) \in [n] | i \in [n])$. This can be done by a nondeterministic choice of $O(n \log n)$ bits goto 4.

4 Check whether the function σ $[n] \to [n]$ is a permutation This can be done by first sorting the list σ If σ is a permutation then goto 5

5 Compute $\sigma(\bar{\mathscr{L}}) = (\sigma(\bar{L}_i) | i \in [n])$ by applying program p_3 Here $\bar{\mathscr{L}}$ is the list that describes $\bar{G}$ Compute $\sigma(\bar{\mathscr{L}})^\sigma = (\sigma(\bar{L}_{\sigma(i)}) | i \in [n])$ by applying program p_2 to the list $\sigma(\bar{\mathscr{L}})$ with elements $\sigma(\bar{L}_i)$ goto 6

6 Sort all lists in $\mathscr{L}$ (which describes G) and in $\sigma(\bar{\mathscr{L}})^\sigma$. If the two-dimensional lists that result from $\mathscr{L}$ and $\sigma(\bar{\mathscr{L}})^\sigma$ coincide then stop

Using the fact that programs p_1, p_2, p_3 have quasilinear time bounds, it can easily be seen that a suitable implementation of the above procedure requires a quasilinear number of nondeterministic steps on a Turing machine. This proves Theorem 2.1. □

Next we consider the satisfiability problem. Let $V = \{x_i | i \in N\}$ be a set of Boolean variables that take the logical values "1 ~ true" and "0 ~ false". Let $\wedge, \vee : K^2 \to K$ and $\neg : K \to K$ be the logical functions conjuction, disjunction, and negation. We abbreviate $x_i = x_i^1$ and $\neg x_i = x_i^0$. A Boolean *clause* is a disjunction of variables and negated variables $(x_{\pi(1)}^{\sigma(1)} \vee x_{\pi(2)}^{\sigma(2)} \vee \cdots \vee x_{\pi(k)}^{\sigma(k)})$, with $\tau(i) \in N$, $\sigma(i) \in K$. A Conjunctive Form (CF) γ is a conjunction of Boolean clauses

$$\gamma = \bigwedge_{i=1}^{k} \bigvee_{j=1}^{l(i)} x_{\tau(i,j)}^{\sigma(i,j)},$$

with $\tau(i, j) \in N$, $\sigma(i, j) \in K$.

A CF γ is called *satisfiable* if there is a map $g : V \to K$ which associates Boolean values with all Boolean variables such that $\gamma_{(x_i = g(x_i) | \text{all } i)} = 1$, i.e. γ is satisfied under g. A CF γ is described as a two-dimensional list $\mathscr{L}$ of pairs $(\sigma(i, j), \tau(i, j))$. The ith list L_i of $\mathscr{L}$ encodes the ith clause of γ. Each pair $(\sigma(i, j), \tau(i, j))$ is encoded as $\overline{c(\sigma(i, j))}\ \overline{c(\tau(i, j))}$ and γ is encoded as a two-dimensional list of these encoded pairs.

The satisfiability problem is the decision problem for the set:

$$\text{satisfiability} = \{C(\gamma) | \gamma \text{ is a satisfiable CF}\}.$$

THEOREM 2.2. *Satisfiability is in NQL*

PROOF. We sketch a nondeterministic program for satisfiability.

1 Decide whether the input x is of the type $C(\mathscr{L})$ where $\mathscr{L}$ is a two-dimensional list of pairs $(\sigma(i, j), \tau(i, j)) \in K \times N$. (Then x is the encoding of a CF γ) In this case goto 2

2 Sort the set $\tau(i, j)$ of indices of variables that occur in $\mathscr{L}$ according to their size and without repetition of elements Let L be the resulting sorted list with length $m = |L|$. goto 3

3 Make a nondeterministic choice of a binary list $\delta = (\delta(i) \in K | i \in [m])$ Compute the list $\bar{\mathscr{L}}$ which is obtained from $\mathscr{L}$ by replacing each element $(\sigma(i, j), \tau(i, j))$ by $(\sigma(i, j), \delta(\tau(i, j)))$. This can be done by applying program p_3. goto 4

4 Evaluate $\gamma_{(x_k = \delta(k) | \text{all } k)}$ This can be done by one pass over the list $\bar{\mathscr{L}}$ If γ takes the value 1 then stop

Using the fact that program p_3 has a quasilinear time bound and that sorting can be done in quasilinear time, the above procedure for satisfiability can easily be implemented as a Turing program with a quasilinear time bound. □

Let nonprimes $= \{c(\nu) \in K^* | \nu \in N \text{ is not a prime}\}$ be the encoding of the set of all nonprimes. Then it is a straightforward consequence of the Schönhage-Strassen fast multiplication algorithm that nonprimes is in NQL.

3. *A Quasilinear Reduction of NQL to Satisfiability*

We shall use the following concept of quasilinear reduction instead of polynomial time

bounded reduction which has been used in the work of Cook, Karp, and Levin.

The quasilinear reducibility $A \leq_{ql} B$ for sets $A, B \subset K^*$ is defined as follows:

$$A \leq_{ql} B \Leftrightarrow \exists \psi \in \mathrm{QL}^f : \forall x \in K^* : x \in A \Leftrightarrow \psi(x) \in B.$$

The relation $\leq_{ql}$ is reflexive and it can easily be seen that $\leq_{ql}$ is transitive, i.e. $A \leq_{ql} B$ and $B \leq_{ql} C$ implies $A \leq_{ql} C$.

Definition 3.1. $A \subset K^*$ is *quasilinear complete* (i.e. $\leq_{ql}$ complete) if (1) $A \in \mathrm{NQL}$ and (2) $\forall B \in \mathrm{NQL} : B \leq_{ql} A$.

These definitions immediately imply Proposition 3.1.

PROPOSITION 1. *The following assertions are equivalent for all $\leq_{ql}$-complete sets A:* (1) $NQL = QL$, (2) $A \in QL$.

PROOF. (1) $\Rightarrow$ (2): Trivial. (2) $\Rightarrow$ (1): $B \leq_{ql} A$ and $A \in \mathrm{QL}$ implies $B \in \mathrm{QL}$.

We are now ready to state the main result of the paper.

MAIN THEOREM 3.1. *Satisfiability is quasilinear complete.*

One part of Theorem 3.1 has already been proved in Theorem 2.2. So it remains to prove "$\forall A \in \mathrm{NQL} : A \leq_{ql}$ satisfiability" This part of the proof will be prefaced by Propositions 3.2–3.4, which describe a sequence of simulations.

Let $A \in \mathrm{NQL}$ be given by a representation according to Proposition 2.1:

$$A = \{x \in K^* \mid \exists u : \bar{u}x \in B \wedge |u| = |x| \lceil \log |x| \rceil^k + k\},$$

with $k \in N$ and $B \in \mathrm{QL}$.

In a first step we remark that we may restrict our considerations to oblivious Turing programs for B. A Turing program p is called *oblivious* if the position of head i in the jth configuration of program p on input x is a function $\mathrm{pos}(i, j, |x|)$ that only depends on i, j and the length $|x|$ of x.

PROPOSITION 3.2. *For every $B \in QL$ there exists an oblivious program p for B (i.e. p computes χ_B) which has a quasilinear bounded running time and which uses two tapes.*

This proposition follows immediately from a theorem of Fischer [3], who proved that for every Turing program p there exists an oblivious Turing program p' which for all m simulates p for m steps by using $O(m \log m)$ steps of p'.

The next step in the proof of Theorem 3 1 is to simulate oblivious Turing programs by logical networks. Let $V = \{x_i \mid i \in N\}$ be a countable set of Boolean variables and let Ω be the set of all Boolean functions with variables in V.

A Boolean computation (logical network) β is a finite directed acyclic graph such that the following are true:

(1) Every node ν of β has either 2 or 0 entering edges. A node without entering edges is called an entry; all other nodes are called nonentries of β.

(2) Every entry ν of β is labeled with a Boolean function $\mathrm{op}(\nu) \in V \cup K$ which is either a variable or a constant.

(3) Every nonentry ν of β is labeled with some binary logical operation $\mathrm{op}(\nu) : K^2 \to K$. The edges which enter ν correspond in a fixed ordered way to the arguments of $\mathrm{op}(\nu)$.

With every node $\nu \in \beta$ we associate an output function $\mathrm{res}^\nu_\beta \in \Omega$ as follows. $\mathrm{res}^\nu_\beta = \mathrm{op}(\nu)$ for all entries ν. For a nonentry ν, res^ν_β is obtained by applying $\mathrm{op}(\nu)$ to the outputs of the preceding nodes. We say β computes res^ν_β for $\nu \in \beta$. Let $\mathrm{size}(\beta)$ be the number of nonentries (i.e. gates) in β.

We can restrict our considerations to logical networks β with the following property: There exist $n, m \in N$ such that $1, 2, \ldots, n$ are the entries and $n + 1, \ldots, m$ are the nonentries of β and $x_i = \mathrm{op}(i)$ for $i = 1, \ldots, n$.

Each node ν of β is described by a triple $(\sigma(\nu), \tau(\nu), \overline{\mathrm{op}}(\nu))$ of natural numbers. $\sigma(\nu)$ ($\tau(\nu)$, respectively) is 0 for all entries ν and is the first (second, respectively) predecessor of ν for all nonentries ν. Then the binary encoding $C(\beta)$ is the encoding of this list of triples where each triple is encoded as $\overline{c(\sigma(\nu))}\ \overline{c(\tau(\nu))}\ \overline{c(\mathrm{op}(\nu))}$.

The following Propositions (3.3 and 3.4) contain the heart of the proof for Theorem 3.1.

PROPOSITION 3.3. *For every oblivious program p with quasilinear bounded running time T_p there exists a function $\Gamma_p \in QL^f$ such that for all $x \in K^*$ with length n,*

$$\Gamma_p(x) = C(\beta_{p,n})01c(v),$$

where $\beta_{p,n}$ is a logical network and v is a node of $\beta_{p,n}$ such that $res^v_{\beta_{p,n}} = res_{\beta|K^n}$.

PROOF. The proof is based on a theorem which is due to Fischer and Pippenger [3]. For every oblivious Turing program p there exist logical networks $\beta_{p,n}$ such that $\beta_{p,n}$ simulates p on inputs of length n and size$(\beta_{p,n}) = O(T_p(n))$. See also [8] for a stronger version of this fact. We have to prove that this simulation can be done efficiently in the sense that $\beta_{p,n}$ can be computed in quasilinear time with respect to n.

We may suppose that p uses the binary alphabet K and 2 tapes. We consider p on inputs of length n. Suppose p has 2^m internal states and uses at most $\alpha_k(n)$ tape squares and $\alpha_k(n)$ steps. A configuration of p (modulo the head positions) is encoded as a binary string of length $m + \alpha_k(n)$ where the first m bits encode the program state and the following $\alpha_k(n)$ bits correspond in one-to-one manner to the symbols in the used tape squares. Only $m + 2$ bits of the configuration are involved in each step of the oblivious computation: the m bits of the program state and the 2 bits of the observed symbols. Let η be a network which computes the successor configuration $SC: K^{m+2} \to K^{m+2}$ where $SC(c(r)x_1x_2) = (c(s)y_1y_2)$ means that $c(r)$ and $c(s)$ are the encodings of the present and the next program state, x_1, x_2 are the presently observed symbols, and x_i is replaced by y_i within the present step. Each step of program p is simulated by a copy of η with the present program state and the presently observed symbols as inputs. $\alpha_k(n)$ steps of program p are simulated by a suitable composition of $\alpha_k(n)$ copies of η. In order to compose the jth copy of η (simulating the jth step) in the right way we have to know the head positions of program p within the jth step. This enables us to feed into η the symbols which are observed within the jth step of program p.

Let β_j be the composition of the first j copies of η and $C(\beta_j)$ its encoding. We sketch stage j of the computation where $C(\beta_{j-1})$ is extended to $C(\beta_j)$ by composing a copy of η with β_{j-1}. The head positions within the jth step of program p can be obtained by simulating the jth step of program p on input 1^n within each stage j. With these head positions available one concatenates to $C(\beta_{j-1})$ the encoding $C(\eta_j)$ of a correct composition of η with β_{j-1}. This can be done in a straightforward way.

$\beta_{\alpha_k(n)}$ has size $(\eta)\cdot\alpha_k(n)$ nodes and a suitable binary encoding of $\beta_{\alpha_k(n)}$ has length $O(\alpha_k(n)\log \alpha_k(n))$. The above sketch yields a Turing program for $\beta_{\alpha_k(n)}$ which requires $O(\alpha_k(n)\log \alpha_k(n))$ steps. This finishes the proof of Proposition 3.3. □

The final step of the proof of Theorem 3.1 requires a simulation of logical networks by conjunctive forms. This simulation uses an idea which is due to Sanden (a student with the author):

PROPOSITION 3.4 (Sandeń-Schnorr). *Let β be any logical network with input variables $x_1, \ldots, x_n$ and nonentries $n+1, \ldots, m$ and $op(i) = x_i$ for $i = 1, \ldots, n$. Then there is a CF γ^β which depends on $x_1, \ldots, x_n, \ldots, x_m$ such that*

(1) *for all nodes v,*

$$\exists y \cdot res^v_\beta(y) = 1 \Leftrightarrow \gamma^\beta_{|x_v=1} \text{ is satisfiable};$$

here $\gamma_{|x_v=1}$ is obtained from γ^β by substituting 1 for x_v;

(2) *γ^β has at most four size(β) clauses, each clause having at most three literals;*

(3) *there is a function $\varphi \in QL^f$ such that $\varphi(C(\beta)) = C(\gamma^\beta)$ for all logical networks β.*

PROOF. We may assume that β has the entries $1, \ldots, m$ where $x_i = op(i)$ is the input of node i. Let $n+1, \ldots, m$ be the nonentries of β. The variable x_v will describe the output function of res^v_β of node v. Consider the Boolean function

$$\gamma = \bigwedge_{i=n+1}^{m} [x_i = op(i)(x_{\sigma(i)}, x_{\pi(i)})],$$

where $\sigma(i)$ is the first and $\tau(i)$ is the second predecessor of node i. This construction implies for all nodes v:

$$\exists y \in K^n : \mathrm{res}^v_\beta(y) = 1 \Leftrightarrow \gamma_{|x_v=1} \quad \text{is satisfiable.}$$

It remains to rewrite a factor $[x_i = \mathrm{op}(i)(x_{\sigma(i)}, x_{\tau(i)})]$ as a product of Boolean clauses. For instance, $x_i = x_\nu \wedge x_\mu$ can be written as $(x_\nu \vee \neg x_i)(x_\mu \vee \neg x_i)(\neg x_\nu \vee \neg x_\mu \vee x_i)$ and $[x_i = (x_\nu \oplus x_\mu)]$ can be written as a product with four Boolean clauses. Therefore, γ^β is obtained from γ by transforming the factors of γ into products of Boolean clauses. This proves (1) and (2). Observe that $C(\gamma^\beta)$ can be constructed from $C(\beta)$ by one pass over the input. The running time of a suitable program is linearly bounded in the length of the input. □

Proof of Theorem 3.1. Using Propositions 3.2-3.4, we can prove $\forall A \in \mathrm{NQL}$; $A \leq_{\mathrm{ql}}$ satisfiability. Let $A \in \mathrm{NQL}$ be given by $B \in \mathrm{QL}$ according to Proposition 2.1. Let p be an oblivious program for B such that $T_p(n) \leq n\lceil \log n \rceil^k + k$ for all n. We abbreviate $\alpha_k(n) := n\lceil \log n \rceil^k + k$.

With every input $y \in K^n$ of program p we associate a CF γ_y as follows:

(1) Apply Γ_p in Proposition 3.3 in order to compute a network $\beta(y)$ and a node ν of $\beta(y)$ such that

$$\mathrm{res}^\nu_{\beta(y)} = \lambda u[\mathrm{res}_p(\bar{u}y)],$$

where u ranges over binary sequences with length $\alpha_k(n)$.

(2) Compute $\varphi(C\beta(y)) := C(\gamma^{\beta(y)})$ according to Proposition 3.4. Let x_ν correspond to the output function $\mathrm{res}^\nu_{\beta(y)}$; then compute the encoding of $\gamma_y := \gamma^{\beta(y)}_{|x_\nu=1}$.

This construction implies

$$\begin{aligned} y \in A &\Leftrightarrow \exists u \in K^{\alpha_k|y|} : \bar{u}y \in B \Leftrightarrow \exists u \in K^{\alpha_k|y|} : \mathrm{res}_p(\bar{u}y) = 1 \\ &\Leftrightarrow \exists u \in K^{\alpha_k|y|} : \mathrm{res}^\nu_{\beta(y)}(u) = 1 \Leftrightarrow \gamma_y \quad \text{is satisfiable.} \end{aligned}$$

Using Propositions 3.3 and 3.4 one can easily see that the function $\psi : y \mapsto C(\gamma_y)$ is in QL^f. Observe that the composition of functions in QL^f always yields a function QL^f. This finishes the proof of Theorem 3.1. □

4. *Further Quasilinear Complete Problems*

There is a number of known reductions ψ of satisfiability to other problems where ψ has a quasilinear time bound. We shall only specify these problems and the corresponding reductions.

3-satisfiability = $\{C(\gamma) | \gamma$ is a satisfiable CF with at most three literals per clause$\}$.

It is a straightforward matter to see that 3-satisfiability is quasilinear complete. We define

$$\text{3-colorability} := \{C(G) | G \text{ is a finite graph which is 3-colorable}\}.$$

Theorem 4.1. *3-colorability is quasilinear complete.*

Proof. It can easily be seen that 3-colorability is in NQL. On the other hand there is a quasilinear reduction ψ:3-satisfiability → 3-colorability which is due to Specker [10]. ψ works as follows: Let the CF

$$\gamma = \bigwedge_{i=1}^{m} (x^{\sigma(i,1)}_{\tau(i,1)} \vee x^{\sigma(i,2)}_{\tau(i,2)} \vee x^{\sigma(1,3)}_{\tau(1,3)})$$

depend on $x_1, \ldots, x_n$. Then the graph $\psi(\gamma)$ with the set of vertices V and set of edges E is defined as:

$$\begin{aligned} V &= \{0, 2\} \cup \{x^1_j, x^0_j | j \leq n\} \cup \{p^i_k | i \leq m, k \leq 5\}, \\ E &= \{(0, 2)\} \cup \{(x^1_j, 2), (x^0_j, 2), (x^1_j, x^0_j) | j \leq n\} \cup \{(x^{\sigma(i,k)}_{\tau(i,k)}, p^i_k) | i \leq m, k \leq 3\} \cup \{\text{the} \\ &\quad \text{edges of Figure 2} | i \leq m\}. \end{aligned}$$

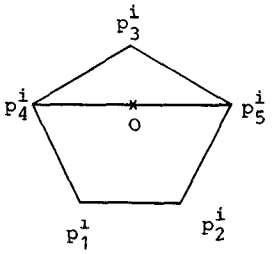

FIG 2

The reduction works [10] and $\psi(\gamma)$ can be computed in linear time from $C(\gamma)$. □

Remember that $\mathscr{L} = (L_i | i = 1, \ldots, m)$ denotes a two-dimensional list and $\Delta(\mathscr{L})$, $\Delta(L_i)$ are the set of elements of $\mathscr{L}$ and L_i.

$$\text{partition} = \left\{ C(\mathscr{L}) \,\middle|\, \begin{array}{l} \exists \text{ subfamily } L_{i_1}, \ldots, L_{i_\sigma} : \Delta(L_{i_\nu}) \text{ are pairwise} \\ \text{disjoint and } \bigcup_{\nu=1}^{n} \Delta(L_{i_\nu}) = \Delta(\mathscr{L}) \end{array} \right\}.$$

THEOREM 4.2. *Partition is quasilinear complete.*

PROOF. By the use of sorting techniques, it can be proved in a straightforward manner that partition is in NQL. A reduction ψ:3-colorability $\rightarrow$ partition can be defined as follows [10]:

Let $G = (V, E)$ be a graph. Then the associated double list $\mathscr{L}$ is defined such that $\Delta(L_i)$, $i = 1, \ldots, m$, is the following family of sets:

$S_{v,f} = \{v\} \cup \{(e, f) | v \text{ is a node of } e \in E\} \subset V \cup E \times [3]$ for $v \in V$, $f = 1, 2, 3$,
$S_{e,f} = \{(e, f)\}$ for $e \in E$, $f = 1, 2, 3$.

It follows that G is 3-colorable $\Leftrightarrow C(\mathscr{L}) \in$ partition and the reduction $\psi: C(G) \rightarrow C(\mathscr{L})$ can be computed in linear time.

Finally, we consider a quasilinear complete problem which seems to be particularly interesting: the anticlique problem, which is also called the discrete subgraph problem. Let G be a directed graph with vertex set V; then a subset $U \subset V$ of pairwise nonadjacent vertices is called an anticlique of G We set

$$\text{anticlique} := \left\{ C(G)01c(k) \,\middle|\, \begin{array}{l} \text{there is an anticlique} \\ U \text{ of } G \text{ with } \|U\| = k \end{array} \right\}.$$

Under a somewhat different encoding the anticlique problem is almost identical to the clique problem. Let the Boolean variable $x_{i,j}$ be true iff there is no edge from i to j in G; then the variables $(x_{i,j} | 1 \leq i, j \leq n)$ encode the directed graphs with vertex set $[n]$ and the problem of deciding whether there exists an anticlique of size k in such a graph is precisely the problem of computing the following Boolean function:

$$\mathrm{CL}_{n,k} = \bigvee_{1 \leq i_1 < \cdots < i_k \leq n} \bigwedge_{1 \leq \nu, \mu \leq k} x_{i_\nu, i_\mu}.$$

If the representation of graphs is changed such that $x_{i,j}$ is true iff there is an edge from i to j then $\mathrm{CL}_{n,k}$ encodes the problem of deciding whether there exists a clique of size k in a graph with vertex set $[n]$. It has been proved by Schnorr [9] that any rational monotone computation for $\mathrm{CL}_{n,k}$ requires at least $\binom{n}{k} - 1$ additions, which shows that the anticlique problem is exponentially hard, at least in a restricted model of computation.

THEOREM 4.3. *Anticlique is quasilinear complete in NQL.*

PROOF. It can easily be seen that anticlique is in NQL: Given a graph G with vertex set $[n]$ and given $k \leq n$, one guesses k nodes $j_1, \ldots, j_k$ which takes $O(n \log n)$ nondeterministic steps. Then, by using techniques for sorting, one can check in quasilinear time whether $j_1, \ldots, j_k$ is an anticlique.

On the other hand, we reduce 3-satisfiability to anticlique. Let a CF

$$\gamma = \bigwedge_{i=1}^{m} (x_{\pi(i,1)}^{\sigma(i,1)} \vee x_{\pi(i,2)}^{\sigma(i,2)} \vee x_{\pi(i,3)}^{\sigma(i,3)})$$

be given. At first we transform γ into another CF $\tilde{\gamma}$ such that each variable x_ν occurs at most 3 times in the clauses of $\tilde{\gamma}$ (negated occurrences of x_ν included). If x_ν occurs more than three times in γ, then we introduce a new variable $x_{\bar{\nu}}$, we substitute one occurrence of x_ν in γ by $x_{\bar{\nu}}$, and we add the clauses $(x_\nu \vee \neg x_{\bar{\nu}})$ $(x_{\bar{\nu}} \vee \neg x_\nu)$ to γ. These additional clauses imply $x_\nu = x_{\bar{\nu}}$. Obviously the transformation $C(\gamma) \mapsto C(\tilde{\gamma})$ can be done in quasilinear time and $\tilde{\gamma}$ is satisfiable iff γ is satisfiable. So far we have proved that in our reduction of 3-satisfiability to anticlique we may restrict to CFs γ such that each variable occurs at most three times. Now let γ be such a CF with m clauses as above. We associate with γ a graph G with vertex set

$$V = \{(y, i) | \text{the literal } y \text{ occurs in the } i\text{th clause of } \gamma\}$$

and edge set

$$E = \{((y_1, i_1), (y_2, i_2)) | i_1 = i_2 \text{ or } y_1 = \neg y_2\}.$$

It is known from the reduction of satisfiability to clique in Karp [5] that the above reduction works, i.e. γ is satisfiable iff (V, E) has an anticlique of size m. On the other hand, it follows from our assumptions on γ that there are at most $3m + 3n$ edges in E, where n is the number of variables in γ. Therefore, the length of the binary encoding of the graph (V, E) is quasilinear in the length of $C(\gamma)$. Hence the transformation $C(\gamma) \mapsto C(V, E)$ can be done in quasilinear time. □

REFERENCES

(Note Reference [6] is not cited in the text)

1 Cook, S A The complexity of theorem-proving procedures Proc Third Annual ACM Symp on Theory of Computing, 1971, pp 151-158
2 Cook, S A A hierarchy for non-deterministic time complexity Proc Fourth Annual ACM Symp on Theory of Computing, 1972, pp 187-192
3 Fischer, M J Lectures on network complexity Preprint, U Frankfurt, 1974
4 Hennie, F C, and Stearns, R E Two-tape simulation of multitape Turing machines *J ACM 13*, 4 (Oct 1966), 533-546
5 Karp, R M Reducibility among combinatorial problems In *Complexity of Computer Computations*, R E Miller and J W Thatcher, Eds, Plenum Press, New York, 1972, pp 85-104
6 Knuth, D E *The Art of Computer Programming, Vol 3. Sorting and Searching*. Addison-Wesley, Reading, Mass., 1973
7. Levin, L.A Universal enumeration problems (in Russian) *Problemi Peredaci Informacii, Tom IX*, 1972, pp 115-116
8 Schnorr, C P The network complexity and the Turing machine complexity of finite functions *ACTA Informatica 7* (1976), 95-107
9 Schnorr, C P A lower bound on the number of additions in monotone computations. *Theoretical Comptr. Sci 2* (1976), 305-315
10 Specker, R, and Strassen, V Komplexitat von Entscheidungsproblemen *Lecture Notes in Computer Science, Vol 43*, Springer-Verlag, Berlin, 1976

RECEIVED JULY 1975, REVISED APRIL 1977