Theoretical Computer Science 68 (1989) 123–134
North-Holland

ON SOME BANDWIDTH RESTRICTED VERSIONS OF THE SATISFIABILITY PROBLEM OF PROPOSITIONAL CNF FORMULAS

V. ARVIND
Department of Computer Science and Engineering, Indian Institute of Technology, Delhi 110016, India

S. BISWAS
Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur 208016, India

Abstract. In the present paper we study the complexity of some restricted versions of the satisfiability problem for propositional CNF formulas. We define these restrictions through their corresponding languages which are identified using the self-reducibility property of satisfiable propositional CNF formulas. The notion of kernel constructibility (similar to self-reducibility) and that of bandwidth are used to define these languages. The results throw some light on the structure of the satisfiability problem. The proof methods illustrate the application of a certain method for reducing Turing machine acceptance problems to decision problems for logics.

1. Introduction

In this paper we consider the complexity of certain restricted versions of the satisfiability problem of propositional formulas in conjunctive normal form. We define the corresponding languages in terms of kernel constructibility, where log-bandwidth violation is allowed in the corresponding kernels, but the constructing relations obey the bandwidth restriction (appropriate definitions follow). Thus, these languages can be seen as ones which allow limited amount of log-bandwidth violation insofar as such violation may occur only at the kernel level. While one of the languages is readily seen to be in P, the rest are shown to be NP-complete. This is of interest as it is known [6] that the satisfiability problem is in P when no log-bandwidth violation is allowed. The languages considered here also illustrate how it is possible to define many restricted versions of an NP-complete language when it is viewed as a kernel-constructible language. Another aspect illustrated in this paper is the efficacy of Borger's approach [3] for reducing Turing machine acceptance problems to decision problems of logics (in our examples propositional). Without this approach proofs of Lemma 3.6 and Theorem 3.7 would have certainly been more difficult, if not impossible. A preliminary version of this paper was presented at the 7th FST-TCS Conference 1987 [2].

2. Basic definitions and the languages considered

The observation that many languages in NP, including all natural NP-complete languages, are self-reducible, motivates the definition of kernel-constructible languages. (As an example of self-reducibility: the set of all Hamiltonian graphs is self-reducible because a graph G is Hamiltonian iff at least one of the graphs obtained from G by removing one of its edges is also Hamiltonian *unless* G itself is just a Hamiltonian cycle).

A language $L \subseteq \Sigma^*$ is said to be *kernel-constructible* if there exists $K \subseteq L$ and a binary relation R on Σ^* such that

(1) K and the graph of R are both in P.

(2) If x is in L and $x\,R\,y$ then y is in L.

(3) For every y in $L-K$ there is an x in K and some n, such that $x\,R\,x_1, x_1\,R\,x_2, \ldots, x_{n-1}\,R\,x_n$, where $x_n = y$ and n and the lengths $|x|, |x_1|, \ldots, |x_{n-1}|$ are all bounded by a fixed polynomial in $|y|$.

(This definition is a modified version of the one given in [1]).

In the above definition, K is called a *kernel* of L and R a *constructing relation* of L. The pair (K, R) is called a construction for L. Often (as is the case in the rest of this paper except in Note 2.1 below) R is length-increasing from left to right, i.e. $x\,R\,y$ implies $|x| < |y|$. In such cases it is natural to fix the polynomial as $|y|$ itself, then the pair (K, R) uniquely defines a language denoted as $L(K, R)$.

Note 2.1. The notion of kernel constructibility is weaker than that of d-self-reducibility [5, 7, 8], which is another attempt to formalize the notion of self-reducibility commonly observed in NP. Kernel constructibility is weaker because of the following: to compare the two notions it will be natural to say x is a proper sub-object of y (in the terminology of self-reducibility as defined by Schnorr [7]) if $x\,R\,y$, where R is a constructing relation. Then, kernel constructibility will allow an object to have even infinitely many proper sub-objects; note that, for an x in a kernel constructible language, (3) in the above definition asserts that there *is a decreasing chain* of length bounded by a polynomial in $|x|$ with the maximum element x; this chain, however, need not be maximal. In the case of d-self-reducibility, though, the length of every decreasing chain is bounded by a polynomial in the size of its maximum element. In fact it is easy to show that every d-self-reducible language is kernel constructible.

Note 2.2. By a formula we shall always mean in this paper a propositional formula in conjunctive normal form with ordered clauses. Let SAT be the set of all satisfiable formulas. A variable or its negation is called a *literal.*

Definition 2.3 (*Kernel constructibility of* SAT). Let the kernel K be defined as the following set of formulas:

$$K = \{F \mid F \text{ is satisfiable and } F \text{ is a conjunction of literals}\}.$$

Define the following binary relation R on formulas: for two formulas F_1 and F_2, the relation $F_1 R F_2$ holds iff F_2 is obtained from F_1 by disjuncting some literal to some clause in F_1. That is, if $F_1 = \{C_1 \wedge C_2 \wedge \cdots \wedge C_m\}$ then there is a j such that $1 \leqslant j \leqslant m$ and $F_2 = \{C_1 \wedge \cdots C_{j-1} \wedge D_j \wedge C_{j+1} \wedge \cdots C_m\}$ where D_j is the disjunction of literals $l_1, \ldots, l_n$, y with C_j being the disjunction of $l_1, \ldots, l_n$ only. Here y is some literal.

We say that F_2 is obtained from F_1 by *literal-adding* (y to the jth clause).

It can be easily seen that $L(K, R)$ is precisely SAT.

Definition 2.4 (*Constructionally restricting* SAT: *languages* L_1, L_2 *and* L_3). Various restrictions of SAT can be obtained by considering different polynomial-time subsets of SAT as kernels and by restricting literal-adding in different ways. A natural restriction on literal-adding is the *log-bandwidth constraint.* Here literal-adding of the literal y to the jth clause of a formula $F = \{C \wedge \cdots \wedge C_m\}$ is allowed provided there exists no k, $|j-k| > c \log_2 m$, such that the kth clause of F contains an occurrence of the variable from which y is built, c being some a priori fixed constant. We define three restrictions of SAT.

(1) $L_1 = L(K_1, R_1)$ where K_1 is the same as K of the construction given earlier for SAT. As for R_1, for two formulas F_1 and F_2, $F_1 R_1 F_2$ holds iff F_2 is obtained from F_1 by log-bandwidth constrained literal-adding.

(2) $L_2 = L(K_2, R_2)$ where the kernel K_2 is defined as:

$$K_2 = \{F \mid F \text{ is satisfiable and any clause in } F \text{ has at most two literals}\}.$$

And the relation R_2 is the same as R_1 above.

(3) $L_3 = L(K_3, R_3)$, where K_3 is the set of all satisfiable Horn formulas, i.e. formulas where each clause contains at most one positive literal. As for R_3, for formulas F_1 and F_2, $F_1 R_3 F_2$ holds provided F_2 is obtained from F_1 by log-bandwidth constrained literal-addding, where the *added literal is positive* (i.e. it is a non-negated variable).

Note 2.5. We have defined above what we mean by log-bandwidth constrained literal-adding. In [6], the concept of *log-bandwidth constrained formulas* is defined: a formula $\{C_1 \wedge C_2 \wedge \cdots \wedge C_m\}$ is said to be log-bandwidth constrained provided there exists *no* i, j with $|i-j| > c \log_2 m$, such that C_i and C_j share a variable in common, c being some fixed constant.

Note 2.6. Each of these languages L_1, L_2 and L_3 is clearly a proper subset of SAT. Further, in each there is no bandwidth constraint in the kernel, but the literal-adding obeys the log-bandwidth constraint. Therefore, in the entire formula only those literals which occur in the kernel part can violate the log-bandwidth constraint. Thus, these languages can be seen as those which allow log-bandwidth violation in a limited manner.

3. Complexity of L_1, L_2 and L_3

Proposition 3.1. *The languages L_1 is in* P.

Proof. For a formula F in L_1, every occurrence of a literal that violates the log-bandwidth constraint must be in every kernel formula from which F can be constructed; further, since a kernel formula consists of one literal clauses, F will be satisfiable when all these log-bandwidth violating literals are given the truth value true. But then the resultant simplified formula F' obeys the log-bandwidth constraint, hence its satisfiability can be checked in polynomial time of its length [6]. □

Theorem 3.2. *The language L_2 is* NP-*complete.*

To prove this we first note that L_2 is in NP (as all kernel constructible languages are [1]). To show its NP-completeness, we define another language L_4, prove that $L_4 \leqslant_m^p L_2$ (in Lemma 3.4), then we show that L_4 is NP-complete (Lemmas 3.5 and 3.6).

Definition 3.3

$L_4 = \{F \mid F$ is a satisfiable formula in conjunctive normal form where a clause has *at most* one literal which violates the log-bandwidth constraint$\}$.

Lemma 3.4. $L_4 \leqslant_m^p L_2$.

Proof. The following function f is the desired polynomial-time reduction. If the given formula $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ has a clause with two or more literals violating the log-bandwidth constraint, then $f(F) = \{x\} \wedge \{\sim x\}$, because such an F cannot, by definition, be in L_4. Otherwise, $f(F) = F$. This would work as a reduction because $L_4 \subseteq L_2$. To see this, if F is in L_4 then it has a satisfying interpretation, say I. Let I set true the literal x_i in C_i of F. Let y_i denote the literal in C_i which violates the log-bandwith constraint if C_i has such a literal (only one such literal is possible, by definition *and* this literal is different from x_i), otherwise let y_i be the constant "false". Then the formula $\bigwedge_{1 \leqslant i \leqslant m} (x_i \vee y_i)$ is in K_2 and clearly F can be obtained from this formula by log-bandwidth constrained literal-adding. Therefore, F is in L_2. □

Next, our task is to prove that L_4 is NP-complete. We achieve this in two parts; the following lemma shows that a certain NP-complete language is accepted by a somewhat specialized kind of non-deterministic Turing machine (henceforth they will be called NDTMs). Subsequently we prove in Lemma 3.6 that the accepting computations of such machines can be captured by elements of L_4.

The language corresponding to the independent set problem which is known to be NP-complete [4] is defined for our purpose as a somewhat different language as follows:

$$\text{INDSET} = \{(n, k, G) \mid G \text{ is an undirected graph with } n \text{ vertices and has an independent set of size } k, \text{ i.e. there are } k \text{ vertices no two of which are joined by an edge}\}.$$

The coding of (n, k, G) that we use is explained below.

Lemma 3.5. INDSET *can be accepted by an* NDTM *with the following properties*:

(1) *M uses three tapes, one read-only input tape, one guess tape and a work tape. Once the guess is written on it, the guess tape is used in a read-only manner.*

(2) *For an input instance with a graph of n vertices, the space used on the guess tape is n and on the work tape is* $O(\log_2 n)$.

(3) *Given an input instance, for each time instant t, one can compute in polynomial time what the position of each head will be for the computation of M on that input instance. In fact, the three head positions depend solely on n, the number of vertices in the input graph.*

Proof. The input to M is in the following form: $n \# k \#$ edge-list bit-array; the first two components each occupying $\lfloor \log_2 n \rfloor$ space, the last component $\binom{n}{2}$ bits. Let the n vertices be numbered from 1 to n. Let the edge between vertices i and j be denoted by the tuple $(\min(i,j), \max(i,j))$, consider the lexicographic ordering of all possible edges, namely $(1,2), (1,3), \ldots, (1,n), (2,3), \ldots, (2,n), \ldots, (n-1,n)$. If the jth edge in this ordering is present in the input graph then the jth bit of the edge-list bit-array is 1, else it is 0. If the vertex j is included in the independent set guessed, then the jth bit of the guess tape is 1, else it is 0.

The work tape consists of nine buffers $B_1, B_2, \ldots, B_9$, separated by distinguishing markers. The first six buffers are of $\lfloor \log_2 n \rfloor$ bits each, whereas each of the last three buffers store a one-bit flag. Thus, the size of the work tape is bounded by $c \log_2 n$, for some constant c.

Let us name the contents of the work tape as shown in Fig. 1. M first reads in n and k from the input tape into the first two buffers of the work tape, then M counts the number of 1's in the guess tape into the counter C, then checks that contents of the second and third buffer are equal, i.e. $k = C$. Then M enters into a phase that verifies that the guessed set of vertices is indeed an independent set. This is done essentially by considering each possible edge (i, j), traversing the guess tape fully to ensure that both i and j do not belong to the guessed set when the edge

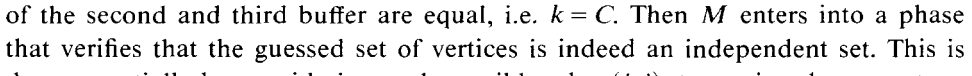

Fig. 1. Organization of the work tape.

(i, j) is there in the graph. During the iteration, v_1 has the value i, v_2 the value j and v the current head position on the guess tape. The algorithm, in detail, is as follows.

begin
initialize the buffers to contain only zeros; {in particular $d = 0$ now}
copy the first two components of the input tape, i.e. n and k into the first two buffers of the work tape;
count the number of 1's in the guess tape into the third buffer of work tape;
if the second and third work-tape buffer's contents are unequal
then {guess set cardinality not equal to k}
$d := 1$; {at this point the input tape head is scanning the first entry of type edge-list bit-array}
{initialization for the phase to verify that the guessed set is indeed independent}
repeat {iteration for each possible edge}
{at this point input head is scanning the entry for the possible edge (v_1, v_2)}; position guess head at the left end;
$v := 1$; $a := 0$; $b := 0$;
repeat {iteration for each entry on guess tape}
{at this point the guess head is scanning the entry for the vertex v}
if $(v_1 = v)$ and guess head is scanning 1 and input head scanning 1
then $a := 1$;
if $(v_2 = v)$ and guess head scanning 1 and input head scanning 1
then $b := 1$;
move guess tape head one square to the right;
$v := v + 1$
until guess tape head has reached the right end marker;
if $(a = 1)$ and $(b = 1)$
then {the edge (v_1, v_2) is present in the graph and both these vertices are in the guessed independent set}
$d := 1$; {now updating v_1, v_2}
if $v_2 = n$ {the variable n is present in the first work tape buffer}
then begin
$v_1 := v_1 + 1$; $v_2 := v_1 + 1$;
end
else $v_2 := v_2 + 1$;
move input tape head by one square to the right
until input tape is exhausted; {at this stage $d = 1$ implies either the guessed set cardinality not equal to k or there is an edge (v_1, v_2) in the graph with both v_1 and v_2 in the guessed set}
if $d = 0$ **then** accept
end.

It is clear from the algorithm, that M accepts INDSET. This completes the proofs of (1) and (2) of Lemma 3.5.

As for (3), the very point of the above algorithm for M to follow was to ensure that this condition can be satisfied by suitable coding. During the entire computation, the input head makes just one left to right complete sweep. The guess tape head, it can be seen from the above algorithm, makes a number (in fact, $\binom{n}{2}+1$) of left to right and back complete sweeps. As for the work tape head, M can be coded in such a way that this head too does its work through complete sweeps, the number of sweeps depending solely on n. For example, consider the work needed to make C have the count of the number of 1's on the guess tape. This can be implemented, by making the guess tape head make a left to right sweep, and for each of its positions, the work tape head makes one complete sweep and in the process adds the symbol currently being scanned by the guess tape head to the buffer containing C. Frequently, M needs to compare two workspace buffers of size $\log_2 n$ (e.g. checking if $v_1 = v$). This can be implemented by requiring the work tape head to make $\log_2 n + 1$ complete sweeps, with auxiliary symbols replacing the kth symbol in each buffer, $1 \leqslant k \leqslant \log_2 n$, in the first $\log_2 n$ sweeps and the last sweep turning the auxiliary symbols back into the original ones. Finally, let us consider the if-then-else statement in the algorithm that updates v_1 and v_2. Both in the "then" part and the "else" part a buffer is incremented by 1; this can be implemented in one sweep. Next, in the "then" part v_1 needs to be copied into v_2 and then v_2 incremented by 1; in the "else" part, however, similar operations are not needed. But to make the head movements identical in the two parts, in the "else" part too redundant head movements are made similar to the "then" part without, of course, making any final changes on the tape.

Thus, we see that, the coding of M can be done satisfying the condition (3) of Lemma 3.5. □

Lemma 3.6. *The acceptance problem of M (defined in the Lemma 3.5) is polynomially many-one reducible to the decision problem of L_4.*

Proof. Given an input (n, k, G) for M, we shall construct a formula F such that F is in L_4 iff (n, k, G) is accepted by M. This construction of F shall be carried out in time polynomial in n and this would prove L_4 to be NP-complete.

We code the computation of the machine M on an input instance using the approach of Borger [3]. Essentially, this encoding does away with some assertions that are logically unnecessary, e.g. it is unnecessary to assert that the machine is in a unique state at every time-instant or to assert that every tape head scans at most one symbol at every time-instant. The usual practice is to include these assertions into the sentence F so that if F is satisfiable it has a unique model, i.e. the categorical model [3]. Here, following the idea in [3] we give an encoding which only has to ensure the following:

(i) The initial conditions are correctly specified.

(ii) The description of the quintuples of M is given, so that there will exist a model of F coding the correct transitions. Further it is asserted that if at a cell some symbol is there at time t, the same symbol remains there at time $t+1$ as well, unless the head was scanning this cell at time t.

(iii) To capture acceptance, a part of the formula asserts that finally the machine is in no state that is an unaccepting state. In this somewhat round about way it is ensured that F admits only models corresponding to accepting computations.

The formula F is a conjunction of assertions for (i), (ii) and (iii) so that F is satisfiable iff M accepts the input.

The propositional variables that F uses are given below with their intended meanings: $C[i, j, t]$ is true if the ith work tape cell has the jth symbol in it at time t, $S[a, t]$ is true if M is in the ath state at time t, $0 \leq i \leq c \log_2 n$, and $1 \leq t \leq p(n)$ (where $p(n)$ is the running time of M and $c \log_2 n$ the work tape size). $Q[i]$ for $1 \leq i \leq n$ are variables corresponding to the n guess tape cells; $Q[i]$ is true if the ith guess tape cell is 1, else $Q[i]$ is false.

Now we describe the different components of F which capture the computation of M.

Coding of the appropriate quintuples

At any time t, the positions of the three heads are known (Lemma 3.5). Further we also know exactly what is the symbol being scanned, for any given t, by the input head and by the guess tape head for a given guess, the contents of the corresponding tapes do not change during the course of the computation. Using this information we define $E_{a,b,t}$ which codes the appropriate quintuples applicable at time t.

Below $H_2(t)$ and $H_3(t)$ denote the position of the guess tape head and that of the work tape head, respectively, at time t. Note that for a given t, the values of $H_2(t)$ and $H_3(t)$ are known in advance, and these values are used below.

$$\begin{aligned} E_{a,b,t} = ((&C[H_3(t), a, t] \wedge S[b, t] \wedge Q[H_2(t)]) \\ &\rightarrow (C[H_3(t), a', t+1] \wedge S[b', t+1])) \\ &\wedge ((C[H_3(t), a, t] \wedge S[b, t] \wedge \neg Q[H_2(t)]) \\ &\rightarrow (C[H_3(t), a'', t+1] \wedge S[b'', t+1])) \end{aligned}$$

where a is a work tape symbol, b is a state and $1 \leq t \leq p(n)$. Other symbols are explained below. $E_{a,b,t}$ essentially describes a move by the machine M when the work tape symbol scanned is a and the state is b at time t. The two clauses give the two possible moves depending on the guess tape cell $H_2(t)$, i.e. whether $Q[H_2(t)]$ is true or false. (Thus, if $Q[H_2(t)]$ is true, then from the appropriate machine quintuple we have that the symbol written in the work tape as a' and the machine goes to state b'. Similarly, for the case when $Q[H_2(t)]$ is false.) We note that neither the input head position nor the symbol it is scanning figure in $E_{a,b,t}$ because by

computing in polynomial time the head position, as the input tape is a read-only tape, we know exactly which symbol the input head is scanning at time t, and this knowledge is used to choose the appropriate quintuples in defining $E_{a,b,t}$.

Only the work tape cell under the head can change

$$D_{k,d,t} = (C[k, d, t] \rightarrow C[k, d, t+1]) \quad \text{for } k \neq H_3(t),$$

$1 \leqslant k \leqslant c \log_2 n$, $1 \leqslant t \leqslant p(n)$ and for all work tape symbols d. This component asserts that contents of no work tape cell other than $H_3(t)$ can change at any time t.

Initial work tape contents

$$\text{Init} = C[1, 0, 0] \wedge \cdots \wedge C[c \log_2 n, 0, 0] \wedge S[1, 0].$$

Initially the work tape contains only zeros. This component simply asserts that. We need such an initializing component only for the work tape because only work tape contents can change. Also we specify the initial state.

Final state description

$$\text{Final} = \prod_{q \neq q_f} \bar{S}[q, p(n)]$$

where we assume that q_f is the *only* accepting state of M. Let

$$\text{ED} = \prod_{t} \prod_{\substack{k \neq H_3(t) \\ a,b,d}} E_{a,b,t} \wedge D_{k,d,t}.$$

Now, let $F = \text{Init} \wedge \text{ED} \wedge \text{Final}$. The proof that F is satisfiable iff M accepts the input instance in question, can be constructed following Borger [3]. This particular layout of F has been chosen to ensure that the required bandwidth constraints are satisfied. Let

$$\text{ED}_t = \prod_{\substack{k \neq H_3(t) \\ a,b,d}} E_{a,b,t} \wedge D_{k,d,t}.$$

We observe that the length of ED_t is bounded by $c' \log_2 n$, for some constant c' and, apart from $Q[i]$'s, variables in ED_t will not occur in ED_j when $j < t-1$ or $j > t+1$. Thus, it is clear that the only literals that violate the bandwidth of $c_0 \log_2 n$ (where c_0 is a constant) are $Q[i]$ and $\bar{Q}[i]$ for $1 \leqslant i \leqslant n$. But from our definition of components these occur only in the $E_{a,b,t}$ and at most most one such literal is present in any clause. Therefore F is satisfiable iff F is in L_4 and so F is in L_4 iff (n, k, G) is in INDSET. Hence L_4 is NP-complete. □

Next, we shall prove that L_3 is NP-complete through the simulation of a different NDTM model where we again use Borger's approach.

Theorem 3.7. *The language is* NP*-complete.*

Proof. It is easy to see that L_3 is in NP. For, given a formula S in L_3 we can guess a satisfiable Horn formula in the kernel from which S is constructed and verify that this Horn formula is satisfiable and that S can be constructed from it using the relation R_3 in polynomial time.

To prove the NP-completeness of L_3 we shall use a somewhat non-standard NDTM model that captures NP. Its description is as follows: the machine, call it M, has two tapes.

(a) The first is an input-cum-work tape. For a given input x of length n, the main tape has $p(n)$ cells marked off where x is in the leftmost part of the marked off region (where $p(n)$ is the running time of M).

(b) The second is a guess tape on which a guessed binary string of length $p(n)$ is written down by the guessing module. The guess tape head starts at the left most cell and moves only to the right one step at a time and finishes at the right most cell after $p(n)$ steps.

After a guessed string w has been written on the guess tape by a guessing module, M functions as a DTM M' running in polynomial time $p(n)$ which accepts or rejects the pair $\langle w, x\rangle$. In fact, we have $L(M) = \{x \mid (\exists w)\ (\langle w, x\rangle$ is in $L(M')$ and $|w| = p(|x|))\}$. Clearly every language in NP can be expressed like this [4]. Hence our NDTM model is powerful enough to capture NP. Given such a machine M and input x, we shall construct a sentence F such that F is in L_3 iff M accepts x.

Simulation of the machine M

We again shall use Borger [3] in the encoding of a computation of M. First we give the variables used in F and their intended meanings:

- $G[i, 1]$ is true if the ith guess tape cell contains 1.
- $G[i, 0]$ is true if the ith guess tape cell contains 0.
- $C[i, j, t]$ is true if the ith work tape cell has the symbol j at time t.
- $S[k, t]$ is true if M is in the kth state at time t.
- $H[i, t]$ is true if the work tape head is on the ith cell at time t.

Throughout $1 \leq i,\ t \leq p(n)$ and j and k have both some constant range. We also have auxiliary variables $D[i, 0]$ and $D[i, 1]$, $1 \leq i \leq p(n)$, to make sure that the variables $G[i, 0]$ and $G[i, 1]$ do not violate bandwidth.

We now describe the various components of the sentence F.

Guess tape contents

The contents of the guess tape are encoded in the sentence B below:

$$B = \prod_{1 \leq i \leq p(n)} (G[i, 0] \vee G[i, 1]) \wedge (\tilde{}\, G[i, 0] \vee \tilde{}\, G[i, 1])$$
$$\wedge (\tilde{}\, G[i, 1] \vee D[i, 1]) \wedge (\tilde{}\, G[i, 0] \vee D[i, 0]).$$

We note from the clauses of B that $D[i, 1]$ is true if $G[i, 1]$ is true and $D[i, 0]$ is true if $G[i, 0]$ is true.

Initial configuration description

$$\text{Init} = S[1,0] \wedge H[1,0] \wedge \prod_{i=1}^{n} C[i, b_i, 0] \wedge \prod_{i=n+1}^{p(n)} C[i,0,0]$$

where $x = b_1 b_2 \ldots b_n$ is the input string and we assume the initial state to be 1.

Encoding of appropriate quintuples

$$\begin{aligned} E_{i,j,k,t} = {} & (C[i,j,t] \wedge H[i,t] \wedge S[k,t] \wedge D[t,1] \\ & \rightarrow C[i,j',t+1] \wedge S[k',t+1] \wedge H[i',t+1]) \\ & \wedge (C[i,j,t] \wedge H[i,t] \wedge S[k,t] \wedge D[t,0] \\ & \rightarrow C[i,j'',t+1] \wedge S[k'',t+1] \wedge H[i'',t+1]). \end{aligned}$$

The two parts of $E_{i,j,k,t}$ together describe the machine move at time t, if the machine at time t scans in state k the ith cell which has the symbol j. The two parts are true according as $D[t,1]$ is true or, respectively, $D[t,0]$. Here $1 \leq i,\ t \leq p(n)$ and both j and k have constant range.

Define

$$E_1 = \prod_{\substack{1 \leq i,t \leq p(n) \\ j,k}} E_{i,j,k,t}.$$

Only cell under H can change

$$E_2 = \prod_{i \neq i_1} ((H[i,t] \wedge C[i_1,j,t]) \rightarrow C[i_1,j,t+1]).$$

The meaning for each clause in E_2 is quite clear.

Final state condition

$$\text{Final} = \prod_{q \neq q_f} \bar{}S[q, p(n)]$$

where q_f is the only accepting state for M. As in the case of the previous theorem, the sentence Final forces M to enter q_f at $t = p(n)$ *and no other state than* q_f.

Let $F = B \wedge \text{Init} \wedge E_1 \wedge E_2 \wedge \text{Final}$. Clearly, every $E_{i,j,k,t}$ can be easily put as a conjunction of Horn clauses. Therefore it is immaterial as to which literal in $E_{i,j,k,t}$ violates log-bandwidth because the kernel consists of satisfiable Horn formulas. Therefore E_1 is a conjunction of Horn clauses. Similarly E_2, Init and Final are all Horn formulas. So it follows that $\text{Init} \wedge E_1 \wedge E_2 \wedge \text{Final}$ is a Horn formula. Further, in B if a clause contains a log-bandwidth violating literal (which has to be either $D[i,1]$ or $D[i,0]$ for some i) that clause is either $(\bar{}G[i,1] \vee D[i,1])$ or $(\bar{}G[i,0] \vee D[i,0])$. But all such clauses are Horn clauses and therefore can be considered to be part of the Horn formula in K_3 from which F is constructed. The only non-Horn clauses are of the form $(G[i,1] \vee G[i,0])$ for $1 \leq i \leq p(n)$. But here both literals satisfy log-bandwidth. Therefore, F is satisfiable iff it can be constructed

from a satisfiable Horn formula (which is in the kernel) using log-bandwidth constrained literal adding. Thus F is satisfiable iff F is in L_3. Since F encodes the computation of M on x, it follows that L_3 in NP-complete. □

4. Concluding remarks

The NP-completeness proofs for L_2 and L_3 use Borger's simulation of Turing machine computations through propositional formulas in CNF. It is of interest to note that the machine model used to accept INDSET captures precisely languages of the type:

$$L = \{x \mid (\exists y) R(x, y) \text{ and } |y| \leq p(|x|) \text{ for some polynomial } p \text{ and } R(x, y) \text{ is computable in DSPACE}(\log_2 n)\}.$$

Such languages are obviously in NP. Moreover, it can be shown that any computation on such a machine M can be captured (as shown for INDSET in Lemma 3.5) through formulas in L_4. Thus, in some sense, L_4 naturally captures computations of the restricted machine model just as propositional CNF formulas capture the polynomial-time computations of the usual NDTM model.

References

[1] V. Arvind and S. Biswas, Kernel constructible languages, in: *Proc. 3rd FSTTCS Conference* (1983).
[2] V. Arvind and S. Biswas, On certain bandwidth restricted versions of the satisfiability problem, in: *Proc. 7th FSTTCS Conference*, Lecture Notes in Computer Science **287** (Springer, Berlin, 1987) 456–469.
[3] E. Borger, Spectralproblems and completeness of logical decision problems, in: *Proc. Symposium "Rekursive Kombinatorik"*, Lecture Notes in Computer Science **171** (Springer, Berlin, 1983) 333–353.
[4] M.R. Garey and M.S. Johnson, *Computers and Intractibility: A Guide to the Theory of NP-completeness* (Freeman, San Francisco, CA, 1979).
[5] A.R. Meyer and M.S. Paterson, With what frequency are apparently intractable problems difficult? Technical Report, MIT/LCS/TM-126, 1979.
[6] B. Monien and I.H. Sudborough, Bandwidth constrained NP-complete problems, in: *Proc. 13th Annual ACMSTOC* (1981) 207–217.
[7] C.P. Schnorr, On self-transformable combinatorial problems, *Math. Program. Study* **14** (1979) 225–243.
[8] A.L. Selman, Natural self-reducible sets, manuscript, 1986.