Theoretical Computer Science 4 (1977) 23–46.

ON THE COMPLEXITY OF REGULAR RESOLUTION AND THE DAVIS–PUTNAM PROCEDURE

Zvi GALIL

Computer Sciences Department, IBM Thomas J. Watson Research Centre, Yorktown Heights, NY 10598, U.S.A.

Communicated by A. Meyer
Received September 1975
Revised June 1976

Abstract. For infinitely many $n > 0$ we construct contradictory formulas α_n in conjunctive form with n literals such that every regular resolution proof tree which proves the contradiction must contain 2^{cn} distinct clauses for some $c > 0$. This implies a 2^{cn} lower bound for the number of distinct clauses which are generated by the Davis–Putnam procedure applied to α_n using any order of variable elimination.

1. Introduction

1.1 The Underlying Problem

One important problem in computer science is the "tautology problem": given a Boolean formula in disjunctive form, is it a tautology? (i.e., is it true for every truth assignment to its variables). The tautology problem is of central importance in computer science, since

(1) There is a polynomial time algorithm for the tautology problem if and only if the class P of problems solved by polynomial time bounded Turing machines is equal to the class NP of problems solved by nondeterministic polynomial time bounded Turing machines [2]; and

(2) There is a nondeterministic polynomial time bounded algorithm for the tautology problems if and only if NP is closed under complementation [4].

Thus the tautology problem is closely related to two of the most important problems in the theory of computational complexity. The dual problem of the tautology problem is: determine if a Boolean formula in conjunctive form (a set of clauses) is contradictory (or unsatisfiable) — i.e., is it false for every truth assignment to all its variables?

These problems are actually identical, since one can transform one problem into the other by taking the complement of the input and then applying DeMorgan's laws. The second is more natural in theorem proving. To prove a theorem from a

set of axioms, take the conjunction of all axioms plus the negation of the theorem and prove a contradiction.

1.2 Notation

We assume that there is an unbounded set $\{x_1, x_2, \ldots, y, \ldots\}$ of *variables*. A *literal* is either a variable or the complement of a variable, denoted x and $\bar{x}$ respectively (or x^1 and x^0). The complement of a literal x ($\bar{x}$) is $\bar{x}$ (x). A *clause* is a finite set of literals $(l_1, \ldots, l_k)$ such that no variable appears more than once, and is denoted by $l_1 \vee l_2 \vee \cdots \vee l_k$. A *formula* (in conjunctive form (CF)) is a finite set of clauses. A *valuation* (or *assignment*) is a function v from the set of literals to $\{0, 1\}$, (Zero is to be thought of as meaning false and one as meaning true.) The *value* (relative to v) of the literal x ($\bar{x}$) is $v(x)$ $(1 - v(x))$. The value of the clause $l_1 \vee l_2 \vee \cdots \vee l_k$ is the maximum of the values of the l_i's. (By convention if $k = 0$ we denote the clause by Λ, and define its value to be 0.) The value of a formula is 1 if the values of all its clauses are 1, and 0 otherwise. Two formulas are *equivalent* if their values are the same for all possible assignments.

A formula is *satisfiable* if there is some valuation for which the value of the formula is 1, and *unsatisfiable* (*contradictory*) otherwise. It is well known that every 0, 1 valued function of k arguments may be represented by a formula in CF with variables $\{x_1, \ldots, x_k\}$. It is also known ([1, 2, 14]) that any formula in CF of length n can be transformed, using additional variables, into another formula of length $O(n)$[1] in CF with at most three literals per clause, such that the second formula is satisfiable if and only if the original formula is satisfiable.

1.3 Resolution

A popular procedure to prove that a set of clauses is contradictory is the resolution procedure [12]. It uses the resolution rule to generate new clauses. A proof of contradiction is obtained if the empty clause is derived. Its dual, which was actually known earlier [11] uses the consensus rule to generate new conjuctions. The input is a tautology if and only if the empty conjunction can be obtained.

Two clauses C_1 and C_2 are said to *conflict* if there are literals in C_1 which appear complemented in C_2. If C_1 and C_2 do not conflict, then we denote by $C_1 \vee C_2$ the clause which contains all literals in C_1 and C_2. The clauses C_1 and C_2 are said to *clash* if there is exactly one literal in C_1 which appears complemented in C_2. If $C_1 \vee x$ and $C_2 \vee \bar{x}$ clash, then their *resolvent* is the clause $C_1 \vee C_2$. We will say that $C_1 \vee C_2$ is obtained from $C_1 \vee x$ and $C_2 \vee \bar{x}$ by applying the *resolution rule*, or by *annihilating* x.

A *resolution proof* of unsatisfiability of a set of clauses S is a sequence of clauses $C_1, \ldots, C_k$ such that $C_k = \Lambda$ and for $0 \leq i \leq k - 1$, C_{i+1} is the resolvent of some pair from $S \cup \{C_1, \ldots, C_i\}$. Robinson [12] showed that a set of clauses is unsatisfiable if and only if there is a resolution proof of its unsatisfiability.

[1] $f(n) = O(g(n))$ if for all n, $f(n) \leq cg(n)$ for some constant c.

We will use binary trees to represent resolution proofs. A *tree* T is a directed graph with a unique vertex which is called the *root*, such that (i) no edge enters the root, and (ii) to every vertex in T there is a unique path from the root. The vertices of the tree will be denoted by intergers, and 1 will always denote the root of the tree. If there is an edge from i to j in T, then j is called the *son* of i and i is called the *father* of j. Two sons of the same vertex are *brothers*. If there is a path from i to j in T, then i is an *ancestor* of j and j is a *descendant* of i. (Every vertex is an ancestor and a descendant of itself.) If vertex i has no sons, then i is a *leaf* of T, otherwise i is an *interior* vertex. All our trees will be binary, i.e., all interior vertices will have exactly two sons. The *height* of T is the length of the longest path from the root to a leaf of T. If i is a vertex of T we sometimes write $i \in T$. If $i \in T$, then T_i is the subtree of T with root i. (More precisely the tree obtained from T consisting of all descendants of i and the edges connecting them.) Obviously $T_1 = T$.

A (*resolution*) *proof tree for a clause* C (*using* S) is a (binary) tree T whose vertices contain clauses in such a way that:

(1) C appears at the root of T;

(2) every leaf of T contains one of the clauses in S, and

(3) the clause in every interior vertex is the resolvent of the clauses at its sons.

A (*resolution*) *proof tree* (*using* S) is a proof tree for the empty clause using S. The complexity of a proof tree T, $N(T)$, is the number of distinct clauses which appear on vertices of T. Note that a proof tree T using S is just a way to represent a resolution proof of the unsatisfiability of S and that $N(T)$ is actually the length of the resolution (straight line) proof defined above. For a proof tree T using S and $i \in T$ let C_i be the clause at i. Obviously T_i is a proof tree for C_i using S.

Example. Let $S = \{\bar{u}, u \vee \bar{x}, x \vee y, \bar{y} \vee u\}$. Fig. 1.3.1 describes a proof tree using S.

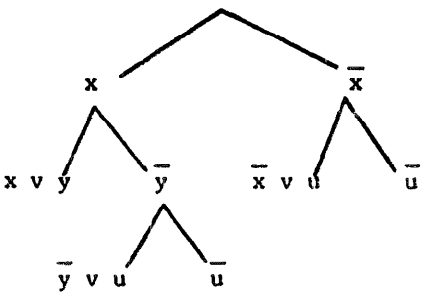

Fig. 1.3.1.

The (*unrestricted*) *resolution procedure* is simply constructing a resolution proof tree using the given set of clauses S.

Let $|S|$ be the number of occurrences of literals in S and let $T_S = \{T \mid T \text{ is a proof tree using } S\}$. The complexity of the resolution procedure is a function Comp defined by:

$$\mathrm{Comp}(n) = \max_{|S|=n} \min_{T \in \mathcal{T}_S} N(T).$$

The complexity of the resolution procedure is unknown at present. We conjecture that it is exponential, but so far we cannot prove that Comp(n) grows faster than linear in n.

1.4 Regular resolution

Let T be a proof tree for some clause C. T is *regular* if there is no vertex i in T such that C_i contains some literal which is annihilated in T_i. Note that a proof tree T (for Λ) is regular if and only if no path from a leaf to the root of T contains the annihilation of any variable more than once. Also note that the proof tree in Fig. 1.3.1 is regular. The occurrence of an irregularity is described in Fig. 1.4.1.

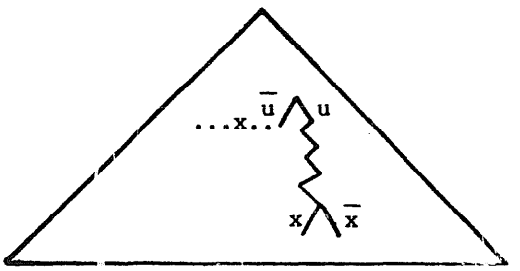

Fig. 1.4.1.

Regular resolution is simply constructing a regular resolution proof tree using S. One should note that a set of clauses S is contradictory if and only if there is a regular proof tree using S. The 'if part' is obvious. The 'only if' part follows from the fact that the inductive proof for the completeness of resolution happens to construct a regular tree. The complexity of regular resolution, $\mathrm{Comp}_{\mathrm{Reg}}$ is defined as Comp except that the minimum is taken only over regular trees. Tseitin [14] introduced regular resolution and proved that $\mathrm{Comp}_{\mathrm{Reg}}(n) \geq 2^{\sqrt{n/32}}$. Unfortunately, his proof is formal, unintuitive, and does not contain most of the details.

Our main result is proving that $\mathrm{Comp}_{\mathrm{Reg}}(n) \geq 2^{cn}$ for some $c > 0$, hence showing that in the worst case regular resolution is almost as bad as the straightforward way to check for contradiction, i.e. evaluating the set of clauses for all 2^n possible assignments. Note that the 2^{cn} lower bound for $\mathrm{Comp}_{\mathrm{Reg}}(n)$ is best possible except for the choice of the constant c.

Our proof of the main result was profoundly influenced by Tseitin's proof. Some of the lemmas and arguments are directly due to Tseitin. Others are implicit in his proof. Following our proof we shall indicate in more detail how our work depends on that of Tseitin.

1.5 The Davis–Putnam procedure

The Davis–Putnam procedure (DPP) [5] is the following routine:

1. Choose a variable x.

II. Replace all the clauses which contain x (or $\bar{x}$) by the set of all clauses which can be obtained from two of them by the resolution rule (by annihilating x).

III. (a) If the empty clause is present, then the original set is unsatisfiable.

(b) If the current set is empty, then the original set is satisfiable.

(c) Otherwise, repeat steps I–III for the new set of clauses.

Step II is referred to as *eliminating x*.

The procedure above works since one can show that all the transformations used preserve satisfiability (unsatisfiability). The complexity of DPP is the number of distinct clauses it generates.

Examples have been found for sets of clauses for which the DPP has exponential complexity ([3, 13]). However, those exploited redundancy in the formula, which can easily be removed by adding the *subsumption rule*: delete from the current set of clauses S any clause C such that there is another clause C' in S, with $C' \subseteq C$ (C' *subsumes* C).

We call an execution of steps I–III a stage. Note that if the original set is unsatisfiable one can construct a proof tree by tracing back the history of the empty clause (as was done for resolution proof). We call such a tree a *DP-tree*. Note that every DP tree T satisfies the following property: x is annihilated above a point at which y is annihilated in T if and only if y was eliminated before x. Hence, every DP tree is regular but not every regular tree is a DP tree. Thus, the fact that DPP is exponential follows immediately from the fact that regular resolution is exponential. (This was first observed in [4].)

1.6 Extended resolution

Extended resolution, a powerful extension of resolution, was introduced by Tseitin [14]. We will refer to it in our concluding remarks.

For three literals x, y, z and a Boolean function f over 2 variables, let $\alpha(f, z, x, y)$ be the set of clauses over x, y, and z which is equivalent to $z \equiv f(x, y)$.[2] One can verify that $\alpha(f, z, x, y)$ exists for all 16 possible choices of f.

Example. $\alpha(\vee, z, x, y) = \{(\bar{x} \vee z), (\bar{y} \vee z), (\bar{z} \vee x \vee y)\}$

$$\alpha(\oplus, z, x, y) = \{(\bar{x} \vee y \vee z), (x \vee \bar{y} \vee z), (x \vee y \vee \bar{z}), (\bar{x} \vee \bar{y} \vee \bar{z})\}.$$

The *extension rule* is applied by choosing f, x and y, creating a new variable z, and adding $\alpha(f, z, x, y)$. *Extended Resolution* is resolution augmented with the extension rule. The use of the extension rule is similar to the use of additional variables in formal algebraic manipulation to denote repeated subexpressions. (Clever substitutions sometimes help to reduce the lengths of some arithmetic computations.)

[2] $\equiv$, $\vee$ and $\oplus$ are equivalence, or and sum mod 2 respectively.

2. Graphs and sets of clauses associated with them

2.1 Notation

We will use extensively sets of clauses corresponding to graphs. We present below the notation which we use for dealing with graphs. All graphs will be connected, undirected, and without self loops. Thus a *graph* is a pair $\bar{G} = (\bar{V}, \bar{E})$, where $\bar{V}$ is its set of vertices $(u, v, w, \ldots)$ and $\bar{E}$ its set of edges (where an edge is an unordered pair (u, v), $u, v \in \bar{V}$, $u \neq v$). A graph $G = (V, E)$ is a *subgraph* of $\bar{G}$ if $V \subseteq \bar{V}$ and $E \subseteq \bar{E}$. (In the literature G is sometimes called a partial subgraph of $\bar{G}$.) We will use the following operations on subgraphs of $\bar{G}$:

$$G = G_1 \cup G_2, \quad \text{where } V = V_1 \cup V_2 \text{ and } E = E_1 \cup E_2 \cup E_{12}.$$

E_{12} is the set of all edges between vertices in V_1 and V_2.

$$G = G_1 \cap G_2 \quad \text{if } V = V_1 \cap V_2 \text{ and } E = E_1 \cap E_2,$$

$$G = G_1 - G_2 \quad \text{if } V = V_1 - V_2 \text{ and } E \text{ consists of all edges in } E_1 \text{ between any two vertices in } V.$$

A vertex v is of *degree* l if there are exactly l edges incident with it. A graph is of degree l if all its vertices are of degree at most l. A graph $G = (V, E)$ is *bipartite* if $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$ and $E \subseteq \{(u, v) \mid u \in V_1, v \in V_2\}$. V_1 and V_2 will be called the two *sides* of G.

Let $G = (V, E)$ be a subgraph of $\bar{G}$. The *boundary of* G, F (or F_G when G is not clear from the context) is the set of edges of $\bar{G}$ which are incident with at least one vertex of G and do not belong to E. The *exterior boundary* (interior boundary) of G, F^{ext} (F^{int}), consists of those edges in the boundary of G which are incident with exactly one vertex (two vertices) of G.

Let s_G be the maximum number of edges in the boundary of G that can be deleted from $\bar{G}$ without disconnecting it. Let $\{H_j\}_{j=1}^{m}$ be the connected components of $\bar{G} - G$ and let k_j, $1 \leq j \leq m$ be the number of edges connecting G and H_j. By definition we have

$$s_G = |F^{int}| + \sum_{j=1}^{m} (k_j - 1) \tag{1}$$

$$|F^{ext}| = \sum_{j=1}^{m} k_j \tag{2}$$

$\bar{G}$:

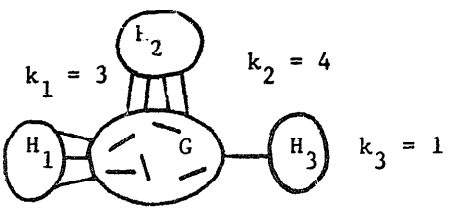

Fig. 2.1.1. $s_G = 10$ ($|F^{int}| = 5$).

In the sequel we will use F_i, s_i for F_{G_i} and s_{G_i} respectively. A graph has *edge connectivity* 3 if the deletion of any two edges from it does not disconnect it.

Lemma 2.1.1. *If $\bar{G}$ has edge connectivity* 3, *then* $s_G \geq |F^{int}| + \frac{2}{3} \cdot |F^{ext}|$.

Proof. This follows immediately from (1), (2) and the fact that $k_j \geq 3$ for $1 \leq j \leq m$ since $\bar{G}$ has edge connectivity 3. □

2.2 Sets of clauses corresponding to graphs

Usually when someone wants to prove something about a proposed procedure for theorem proving in the propositional calculus, he faces the following problem: How does one construct a nontrivial infinite collection of unsatisfiable formulas? One set of examples is the following:

Let $\beta_{n,k}$ be a formula in CF over n variables which contains all clauses of size k such that the variables in any term are either all complemented or all noncomplemented. For every $n \geq 0$ and $k \leq \lceil n/2 \rceil$[3], $\beta_{n,k}$ is unsatisfiable since every assignment has either k zeros or k ones, and thus at least one of the clauses must have the value 0.

The previous example is derived from the following general principle: choose a set of clauses which altogether exclude the 2^n possible assignments (which can be viewed as the corners of an n-dimensional cube).

Another general principle is: take an invalid statement and formalize its instances in the propositional calculus. As an example consider the "occupancy problem" by Cook and Karp (cf. [6] page 45). The statement is, "n objects can occupy all of $n+1$ holes". Another example: for every n one can formalize the false statement: "A clique of size n can be colored with less than n colors".

Tseitin used the following invalid statement:

A: "If you compute the sum modulo 2 of a number of binary variables in such a way that every variable appears exactly twice you get 1".

To construct a formula α in CF which expresses this statement we take a graph and label its vertices with zeros and ones and its edges with literals. Next, for every vertex v we express in CF the statement

B: "The sum modulo 2 of all the edge labels incident with v is equal to the vertex label of v".

α is the union of all these sets of clauses. Summing up statement B for all the vertices implies that the sum modulo 2 of all vertex labels must be 0, since each edge label appears in the sum exactly twice (once for each of its endpoints). By choosing a labelling with sum modulo 2 of vertex labels equalling 1, we formalize statement A in an indirect manner. By choosing a graph with a small degree we can express statement B by a relatively short formula. This is essential, since otherwise

[3] $\lceil n/2 \rceil$ is the smallest integer greater than or equal to $n/2$.

we will have too long an input for which even a brute force algorithm (like truth table computation) takes nonexponential run-time.

Formally, let $G = (V, E)$ be a given undirected graph without loops (i.e., there is no edge (v, v)). Let l be a labelling which assigns to every vertex v a label $\varepsilon_v \in \{0, 1\}$, the *parity of* v, and each edge a literal in such a way that no two edges are labelled by a literal corresponding to the same variable. $G(l)$ will stand for the labelled graph.

Consider a vertex v in V. Assume that v has degree k and $z_1, z_2, \ldots, z_k$ are the literals which label the edges incident with v. A clause C corresponds to the vertex v if and only if

$$C = z_1^{a_1} \vee \cdots \vee z_k^{a_k}$$

and the number of complemented z_i's is of parity opposite to the parity of v, i.e.,

$$\sum_{i=1}^{k} (1 - a_i) \neq \varepsilon_v \qquad \left(\sum \text{ is sum modulo } 2\right),$$

which will be referred to as the parity condition with respect to v.

Note that there are exactly 2^{k-1} distinct clauses which correspond to v. The following lemma proves that the formula which consists of all of them is equivalent to statement B mentioned before, namely that the parity of the number of z_i's assigned the value 1 must be ε_v. Intuitively it is clear, since each clause which corresponds to v excludes one assignment which does not satisfy the above, and all the clauses exclude all unwanted assignments.

Lemma 2.2.1. *The assignment δ given by $z_1 = \delta_1, \ldots, z_k = \delta_k$ satisfies all the clauses corresponding to v if and only if*

$$\sum_{i=1}^{k} \delta_i = \varepsilon_v.$$

Proof. If $\sum_{i=1}^{k} \delta_i \neq \varepsilon_v$, then $C = z_1^{1-\delta_1} \vee \cdots \vee z_k^{1-\delta_k}$ corresponds to v and is not satisfied by the assignment δ.

Assume $\sum_{i=1}^{k} \delta_i = \varepsilon_v$ and let $C = z_1^{a_1} \vee \cdots \vee z_k^{a_k}$ be an arbitrary clause corresponding to v. The only assignment which does not satisfy C is $z_1 = 1 - a_1, \ldots, z_k = 1 - a_k$. Since $\sum_{i=1}^{k} (1 - a_i) \neq \varepsilon_v$ it must differ from the assignment δ. Thus, δ must satisfy C. □

We denote by $\alpha(G(l))$ the set of all clauses which correspond to all vertices in G (l is the labelling).

Claim. $\alpha(G(l))$ is unchanged if we apply the following transformations to the labelling:

Transformation 1: change exactly one edge label (from z to $\bar{z}$) and simultaneously change the vertex labels of its endpoints.

Transformation 2: change all edge labels along a path from vertex u to vertex v and simultaneously change ε_u and ε_v.

Transformation 2 is a sequence of applications of Transformation 1. The latter does not affect $\alpha(G(l))$ since if a clause C corresponds to a vertex w under the old labelling it does so under the new labelling: for $w \notin \{u, v\}$ this is obvious since nothing has changed; for $w = u (w = v)$ the change of ε_u (ε_v) and of z to $\bar{z}$ implies a double change in parity. Hence, C satisfies the parity condition with respect to u (v) under the new labelling.

Let $\varepsilon(G(l)) = \sum_{v \in V} \varepsilon_v$.

Lemma 2.2.2. *For a connected graph G and a labelling l, $\alpha(G(l))$ is satisfiable if and only if $\varepsilon(G(l)) = 0$.*

Proof. Assume $\alpha(G(l))$ is satisfiable. Using Lemma 2.2.1 and summing (mod 2) all ε_v's we get $\varepsilon(G(l)) = 0$, since every δ_i appears twice (once for each endpoint of the edge labelled by z_i).

Assume $\varepsilon(G(l)) = 0$. Using Transformation 2 we can set all vertex labels to 0. If the new edge labels are $z'_1, \ldots, z'_n$ the assignment $z'_i = 0$ for $1 \leq i \leq n$ satisfies $\alpha(G(l))$. □

Thus, if $\varepsilon(G(l)) = 1$, $\alpha(G(l))$ is unsatisfiable, since it formalizes statement A mentioned before.

Let T be a proof tree using a contradictory set of clauses which corresponds to a graph, and let i be a vertex in T. We denote by V_i the set of all vertices that correspond to clauses that appear on the leaves of T_i. The following lemma proves that every proof tree using $\alpha(G(l))$ must use at least one clause corresponding to each vertex in G.

Lemma 2.2.3. *Assume $\alpha(G(l))$ is contradictory and T is a tree which proves the contradiction. Then $V_1 = V$ (where 1 is the root).*

Proof. Assume the claim is wrong and that $v \in V - V_1$. Define a new labelling l' by changing ε_v. By Lemma 2.2.2, $\alpha(G(l))$ is contradictory implies $\varepsilon(G(l)) = 1 \Rightarrow \varepsilon(G(l')) = 0 \Rightarrow \alpha(G(l'))$ is satisfiable. But all the clauses on the leaves of T belong also to $\alpha(G(l'))$ since none corresponds to v. Thus, we have a proof tree using a satisfiable set of clauses — a contradiction which proves the lemma. □

We have just proved that if T is a proof tree using $\alpha(G(l))$, then for each vertex in G there must appear at least one clause corresponding to it on a leaf of T. Must all the clauses in $\alpha(G(l))$ appear on leaves of T? The answer is negative and is proved by the example in Fig. 2.2.1.

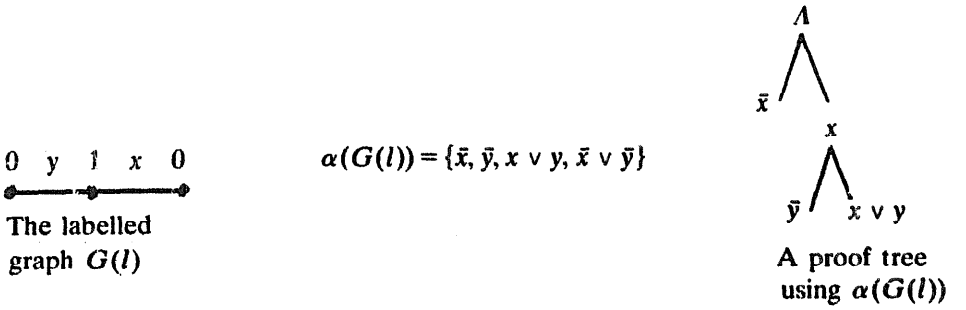

Fig. 2.2.1.

2.3 The specific collection of graphs

We consider a graph $G = (\bar{V}, \bar{E})$ which depends on a parameter m and hence it actually represents a collection of graphs. When speaking on constants (like c, d below) we shall mean constants with respect to m. $\bar{G}$ is the graph H_m which was introduced by Margulis [10]. We do not describe H_m below. We only mention some of its properties. H_m is a bipartite graph of degree 5, such that each of its sides contains m^2 vertices. Theorem 2.3.1 below is a rewording of a special case of Margulis' main theorem. (Namely, taking $\alpha = 1/2$ in Theorem 2.3 in [10].)

Theorem 2.3.1. *There is a constant $d > 1$, such that if $\hat{V}_1$ is contained in one side of H_m, $|\hat{V}_1| \leq m^2/2$, and $\hat{V}_2$ consists of all the vertices in the other side of H_m which are connected to vertices of $\hat{V}_1$ by an edge, then $|\hat{V}_2| \geq d\,|\hat{V}_1|$.*

Corollary 2.3.1. *There is a constant $c > 0$, such that if $G = (V, E)$ is a subgraph of H_m and $m^2/4 \leq |V| < m^2/2$, then $|F_G^{ext}| \geq cm^2$.*

Proof. Assume $V = V_1 \cup V_2$ and V_1 (V_2) consists of vertices in the first (second) side of H_m. W.l.o.g. $|V_1| \geq |V_2|$. Let $\hat{V}_1 = V_1$ and $\hat{V}_2$ be as in Theorem 2.3.1. It follows that $|\hat{V}_2| \geq d\,|\hat{V}_1|$. Hence,

$$|F_G^{ext}| \geq |\hat{V}_2| - |V_2| \geq (d-1)|V_1| \geq \frac{d-1}{2}|V| \geq \frac{d-1}{8}m^2. \quad \square$$

We take a labelling l such that $\varepsilon(\bar{G}(l)) = 1$, and $\alpha(\bar{G}(l))$ which has at most five literals per clause, is our set of clauses. (It is contradictory by Lemma 2.2.2.) n will stand for the size of $\alpha(\bar{G})$, i.e. the number of occurrences of literals in $\alpha(\bar{G})$. It is easy to see that $n = 0(m^2)$.

Lemma 2.3.1. *Let $\{G_i = (V_i, E_i)\}_{i=1}^k$ be a sequence of subgraphs of $\bar{G}$ such that*: (1) *$G_1 = \bar{G}$*, (2) *$|V_k| = 1$ and* (3) *G_{i+1} is related to G_i in the following way: There is an edge $x \in E_i$ such that if $G_i' = (V_i, E_i - x)$ is connected, then $G_{i+1} = G_i'$. Otherwise G_{i+1} is a connected component of G_i' that has the largest number of vertices. Then there is an i with $|F_{G_i}^{ext}| \geq cn$ for some $c > 0$.*

Proof. By (3) $|V_{i+1}| \geq |V_i|/2$. So $|V_i|$ decreases from $2m^2$ to 1 and never decreases by more than half in any one step. Thus, there must be an i on the path with $m^2/4 \leq |V_i| < m^2/2$. By Corollary 2.3.1, $|F_G^{ext}| \geq cm^2 \geq c'n$. □

One can easily verify that $\bar{G} = H_m$ has edge connectivity 3. Thus, by Lemma 2.3.1 we have

Lemma 2.3.2. *If G is a connected subgraph of $\bar{G}$, then $s_G \geq |F_G^{int}| + \frac{3}{4}|F_G^{ext}|$.*

From now on we consider a fixed connected graph $\bar{G} = (\bar{V}, \bar{E})$ with a fixed labelling l such that $\varepsilon(\bar{G}(l)) = 1$. Thus, $\alpha(\bar{G}(l))$ is unsatisfiable. We will refer to $\alpha(\bar{G}(l))$ as the original set of clauses. We will identify the edges and their labellings if no confusion will arise. Without loss of generality we assume that the edge labels are all variables (i.e., they are noncomplemented). We do not specify $\bar{G}$, since most of the proofs do not depend on the specific graph. But at the end of the proof of the main result we will specify $\bar{G}$ as the graph described above.

3. Some useful correspondence relations

3.1 Introduction

By definition every clause in $\alpha(\bar{G})$ corresponds to a vertex in $\bar{G}$. It turns out that if T is a regular proof tree using $\alpha(\bar{G})$, every clause in T corresponds in some sense to a subgraph of $\bar{G}$. This correspondence is described by the Correspondence Lemma in subsection 3.2. In subsection 3.3 we define a process of deleting edges from the graph $\bar{G}$, called a deletion process. Then, we prove a one to one correspondence between deletion processes on $\bar{G}$ and regular proof trees using $\alpha(\bar{G})$.

3.2 The correspondence lemma

Let $G = (V, E)$ be a connected subgraph of $\bar{G}$. A clause C is *associated with* G if its variables are the labels of the edges in the boundary of G. If C is associated with G, then the *parity of* C, $\delta(C)$, is the parity of the number of complemented variables in C which correspond to the exterior boundary of G, i.e., if $C = z_1^{a_1} \vee \cdots \vee z_k^{a_k}$, then $\delta(C) = \sum_{z_i \in F^{ext}} (1 - a_i)$. The *parity of* G, $\varepsilon(G)$, is defined to be the sum modulo 2 of the vertex labels of G. C is a *regular clause associated with* G if C is associated with G and the parity of C is opposite the parity of G, i.e., $\delta(C) \neq \varepsilon(G)$. C is a *regular clause* if there is a connected subgraph G of $\bar{G}$ such that C is a regular clause associated with G.

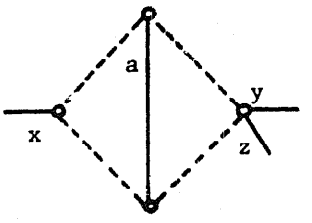

Fig. 3.2.1. A subgraph G of $\bar{G}$ and all edges of $\bar{G}$ which are incident with its vertices.

Example. Consider the subgraph $G = (V, E)$ of $\bar{G}$ in Fig. 3.2.1. V consists of the circles and E consists of the dotted lines. Assume that all vertex labels are 0, thus $\varepsilon(G) = 0$, $F^{\text{int}} = \{a\}$ and $F^{\text{ext}} = \{x, y, z\}$. Let $C_1 = a \vee x \vee y \vee z$, $C_2 = \bar{a} \vee x \vee y \vee z$ and $C_3 = a \vee x \vee y \vee \bar{z}$. All three clauses are associated with G, but only C_3 is a regular clause since $\delta(C_3) = 1$ while $\delta(C_1) = \delta(C_2) = 0$.

One can make the following observations:

(i) For every clause C, there can be at most one connected subgraph G of $\bar{G}$, such that C is a regular clause associated with G.

Proof. Assume C is a regular clause associated with G. By deleting all the edges of the boundary of G from $\bar{G}$ we obtain a possibly unconnected graph G', where G is one of its connected components. If the interior boundary of G is nonempty, C can be associated only with G. This is also the case when G' has more than two connected components. Otherwise, i.e., in case $F^{\text{int}} = \emptyset$ and $G \cup G_1 = \bar{G}$ (G_1 is the other component), C is also associated with G_1. C cannot be a regular clause associated with G_1 since $\varepsilon(G) \oplus \varepsilon(G_1) = \varepsilon(\bar{G}) = 1 \Rightarrow \varepsilon(G_1) \neq \varepsilon(G) \neq \delta(C) \Rightarrow \varepsilon(G_1) = \delta(C)$. □

The second observation is obvious:

(ii) C is one of the original clauses which corresponds to a vertex v in $\bar{G}$ if and only if C is a regular clause associated with the subgraph $G = (v, \emptyset)$ consisting of an isolated vertex.[4]

The next lemma proves that if T is a regular proof tree for a clause C, then C is a regular clause. The lemma specifies exactly the connected subgraph G with which C is associated.

Lemma 3.2.1. (The Correspondence Lemma). *If T is a regular proof tree for a clause C, then C is a regular clause associated with some connected subgraph $G = (V, E)$ of $\bar{G}$ where:*

$V = \{v \mid v \in \bar{V}$ *and a clause corresponding to v appears on a leaf of $T\}$, and*

$E = \{x \mid x$ *is annihilated in $T\}$.*

Proof. Induction on the height h of T. The basis of the induction ($h = 0$) follows immediately from observation (ii) preceding the lemma.

[4] We identify the vertex v and the singleton $\{v\}$.

Induction step: Let T be the derivation in Fig. 3.2.2 and assume $x \vee C_1$ ($\bar{x} \vee C_2$) is a regular clause associated with G_1 (G_2) which satisfies the hypothesis of the lemma.

T:

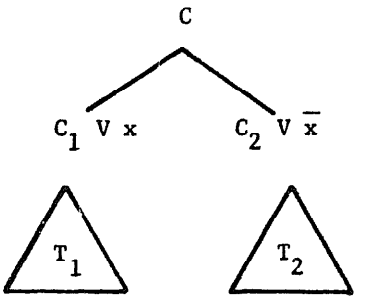

Fig. 3.2.2.

Without loss of generality there are four cases:

1. $G_1 = G_2$ (the subgraphs are identical).
2. $V_1 \cap V_2 = \emptyset$[5] (they are vertex disjoint).
3. $V_1 = V_2$ and there is $z \in E_1 - E_2$.
4. $V_1 \cap V_2 \neq \emptyset$ and there is $u \in V_1 - V_2$.

Claim. Cases 3 and 4 are not possible.

Proof. We show that in both cases $E_1 \cap F_2 \neq \emptyset$, i.e., there is a variable z annihilated in T_1 such that either z or $\bar{z}$ appears in $C_2 \vee \bar{x}$ — a contradiction since T is regular. For case 3 it is obvious that $z \in E_1 \cap F_2$. For case 4 let $v \in V_1 \cap V_2$. G_1 is connected, thus there is a path in G_1 from u to v. There must be an edge z on this path such that one of its endpoints is in V_2 and one is not. Obviously $z \in E_1 \cap F_2$. □

To complete the proof of the lemma we now consider cases 1 and 2.

Case 1. $G_1 = G_2$ and thus $F_1 = F_2 . x \vee C_1$ and $\bar{x} \vee C_1$ consist of the same variables and have exactly one conflict (x vs. $\bar{x}$), thus $C_1 = C_2 = C$. Since the parity of $C_1 \vee x$ and $C_2 \vee \bar{x}$ both differ from the parity of $G_1 = G_2$, they must be the same. Hence, $x \in F_1^{\text{int}} = F_2^{\text{int}}$. Let $G = (V_1, E_1 \cup x)$. It is easy to check that the pair (C, G) satisfies the induction hypothesis. The parity of G differs from the parity of C since $\varepsilon(G) = \varepsilon(G_1)$ and $\delta(C) = \delta(C_1 \vee x)$.

Case 2. $V_1 \cap V_2 = \emptyset$. Let $G = (V_1 \cup V_2, E_1 \cup E_2 \cup x)$. Obviously, G is connected, C is associated with G, $E = E_1 \cup E_2 \cup x$ consists exactly of the variables annihilated in T and $V = V_1 \cup V_2$ consists exactly of those vertices which correspond to the clauses on the leaves of T. To complete the proof that the pair (C, G) satisfies the induction hypothesis we now show that G has parity opposite to that of

[5] For $i = 1, 2$ $G_i = (V_i, E_i)$, F_i, F_i^{ext}, F_i^{int} are defined in the obvious way.

C. Since $\delta(C_1 \vee x) \neq \varepsilon(G_1)$ and $\delta(C_2 \vee \bar{x}) \neq \varepsilon(G_2)$, $\delta(C_1 \vee x) \oplus \delta(C_2 \vee \bar{x}) = \varepsilon(G_1) \oplus \varepsilon(G_2) = \varepsilon(G)$. We show below that

$$\delta(C_1 \vee x) \oplus \delta(C_2 \vee \bar{x}) \neq \delta(C) \tag{1}$$

which will imply that $\delta(C) \neq \varepsilon(G)$ and will complete the proof of the lemma.

To prove (1), recall that if $C = z_1^{a_1} \vee \cdots \vee z_r^{a_r}$, then $\delta(C) = \sum_{z_i \in F^{ext}} (1 - a_i)$. But $F_1^{ext} \cap F_2^{ext} \equiv x \cup F_{12}$ and $F_{12} \subseteq F^{int}$. (See Fig. 3.2.3.) In $\delta(C_1 \vee x) \oplus \delta(C_2 \vee \bar{x})$ the terms corresponding to F^{ext} appear exactly once, the terms for F_{12} appear exactly twice (hence they can be dropped) and since x appears once as x and once as $\bar{x}$, $\delta(C_1 \vee x) \oplus \delta(C_2 \vee \bar{x}) = \delta(C) \oplus 0 \oplus 1$ and (1) follows. □

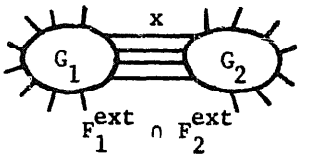

Fig. 3.2.3.

Corollary 3.2.1. *Let T be the regular proof tree of Fig. 3.2.2 and let C ($C_1 \vee x$, $C_2 \vee \bar{x}$) be associated with $G = (V, E)$ (G_1, G_2 respectively). Then $x \in E$ and if we delete x from E and obtain $G' = (V, E - x)$ there are two cases:*

Case 1: G' is still connected. In this case $G' = G_1 = G_2$ and we will say that x *breaks a cycle in* G.

Case 2: G' is not connected. In this case G_1 and G_2 are its connected components and we will say x *splits* G (into G_1 and G_2).

In [2] Cook observed that the order of elimination of variables in DPP is important. A restatement of his result is given by Proposition 3.2.1.

Proposition 3.2.1. *For infinitely many n, there are sets of clauses S_n with n clauses and* log n *literals per clause such that* (a) *one order of elimination of variables yields at least 2^{cn} distinct clauses and* (b) *another order of elimination yields less than dn distinct clauses (c, d are constants).*

In order to give a simple example of using the Correspondence Lemma we prove a slightly stronger claim:

Proposition 3.2.2. *Same as Proposition* 3.2.1, *except that S_n has n clauses of size* 3.

Proof. Consider the labelled graph in Fig. 3.2.4. and let $S_n = \alpha(\hat{G}(l))$. Obviously all the clauses are of size 3.

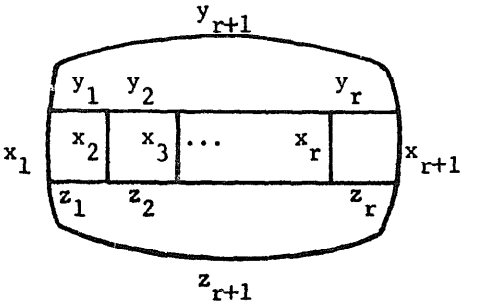

Fig. 3.2.4. $\hat{G}(l)$, l is a fixed labelling with $\varepsilon(\hat{G}(l)) = 1$.

Order 1: Eliminate $y_1, \ldots, y_{r+1}, z_1, \ldots, z_{r+1}, x_1, \ldots, x_{r+1}$.

Order 2: Eliminate $x_1, y_1, z_1, x_2, y_2, z_2, \ldots, x_{r+1}, y_{r+1}, z_{r+1}$.

By the remarks in subsection 1.5 and by the Correspondence Lemma all clauses generated by DPP are regular.

Consider Order 1 after the elimination of x_1, and let G be the connected subgraph of $\hat{G}$ which is obtained from $\hat{G}$ by deleting the edges labelled by $x_2, \ldots, x_{r+1}$. One can easily verify that all 2^r clauses associated with G are generated at this point. Thus, (a) above follows since $n = O(n)$. On the other hand, Order 2 yields clauses associated with $O(n)$ distinct subgraphs the boundaries of which contain at most 3 edges. This implies (b) above. □

3.3 Deletion processes

A *deletion process* on a graph $\bar{G}$ is a process which constructs a tree T whose vertices are labelled by subgraphs of $\bar{G}$. The tree is constructed from the root down:

Generate the root of T and label it by $\bar{G}$.

If i has been constructed and has been labelled by G_i, then if G_i consists of a single vertex, then i is a leaf of T.

Otherwise, generate two vertices, i_1 and i_2, as sons of i.

Choose an edge x in G_i and delete it from G_i to obtain G_i'.

If G_i' is connected set $G_{i_1} = G_{i_2} = G_i'$.

Otherwise, define G_{i_1} and G_{i_2} to be the connected components of G_i'.

Lemma 3.3.1. (a) *Every regular proof tree has a unique corresponding deletion process and* (b) *every deletion process has a unique corresponding regular proof tree.*

Proof. (a) follows from Corollary 3.2.1: x is deleted at i if and only if it is annihilated at i in the regular proof tree. To prove (b) assume we are given a deletion process. Let T and $\{G_i\}$ be the tree and the set of graphs which are generated by the deletion process. We can label each vertex i in T by a clause C_i in such a way that (1) C_i is the regular clause associated with G_i, and (2) the clause at each vertex is the resolvent of the clauses at its sons. Since we terminate when G_i is an isolated vertex, observation (ii) above implies that T labelled with the clauses is a regular proof tree.

We use an additional labelling: If x is chosen at vertex i we label one of the edges $(i, i_1), (i, i_2)$ by x and one by $\bar{x}$. Given the labels from the root to i we take $C_i = \{x^a \mid x \in F_i, x^a$ appears on the path from the root to $i\}$. Note that literals corresponding to all the variables in F_i must appear on this path. Thus, (2) above holds automatically. (See example below.) We now show how to choose the labels for the edges (i, i_1) and (i, i_2) in such a way that if (1) holds for C_i it holds for C_{i_1} and C_{i_2} which are obtained in the way described above. If x does not split G_i, then the choice does not matter: We label one of the edges by x and the other by $\bar{x}$. An argument similar to that in Case 1 of the Correspondence Lemma implies that C_{i_1} and C_{i_2} are regular clauses associated with G_{i_1} and G_{i_2} respectively. Otherwise, (x splits G_i into G_{i_1} and G_{i_2}) a parity argument as in Case 2 of the Correspondence Lemma shows that there is exactly one way to assign x to one of the edges $(i, i_1), (i, i_2)$ and $\bar{x}$ to the other, such that C_{i_1} and C_{i_2} are regular clauses associated with G_{i_1} and G_{i_2} respectively.

We start with $C_1 = \Lambda$, which is indeed a regular clause associated with $G_1 = \bar{G}$ and construct the regular proof tree from the root down. Note that this construction is the only way to satisfy (1) and (2) above. Since the only free choice (when x does not split G_i) does not yield distinct proof trees (we do not distinguish between left and right subtrees) the regular proof tree is determined uniquely. □

Example. Assume G_i is the subgraph G in Fig. 3.2.1, and the labelled path from the root is as in the figure below. Then $C_i = z \vee \bar{x} \vee a \vee y$.

(u and v label edges which are not incident with vertices of G.)

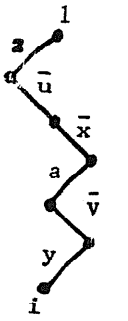

Fig. 3.2.5.

4. Exponential lower bound for regular resolution

4.1 Introduction

We use a contradictory set of clauses which correspond to a graph $\bar{G}$. We consider a regular proof tree T which has the minimal number of distinct clauses $N(T) \equiv \hat{N}$. The trick is to define a weight function w such that for any clause C which appears on T, $\Sigma w(C) \leq 1$, where the sum is taken over all occurrences of C in T. This will imply $\Sigma_{i \in T} w(C_i) \leq N(T) = \hat{N}$. Next, by obtaining a lower bound for the sum above we derive a lower bound for $\hat{N}$.

In order to define the weight function we first check when it is possible that the same clause appears in two different subtrees. The answer appears in Lemma 4.2.1, which is the hardest part of the proof.

First we need to make several observations. For $i \in T$, let $G_i = (V_i, E_i)$ be the connected subgraph of $\bar{G}$ associated with C_i. If i is not a leaf let i_1 and i_2 be its sons and let x_i be the variable annihilated at i. Let $G_i' = (V_i, E_i - x_i)$. By Corollary 3.2.1, there are two possible cases:

Case 1. x_i breaks a cycle in G_i and $G_{i_1} = G_{i_2} = G_i'$.

Case 2. x_i splits G_i and G_{i_1} and G_{i_2} are the two connected components of G_i'.

Thus if i is an ancestor of j in T, G_j is a subgraph of G_i. Also, by the observation following the Correspondence Lemma, if $C_{r_1} = C_{r_2}$, then $G_{r_1} = G_{r_2}$. Hence, in Case 2 none of the clauses in T_{i_1} can be the same as any clause in T_{i_2}. In what follows we will be interested in the question, "When is it possible that T_{i_1} and T_{i_2} contain the same clause C?"; thus, we will be interested in Case 1 only.

4.2 The proof of the exponential lower bound

The proof uses three lemmas, which will be proved later. We now consider a subcase of Case 1 above: Given a subgraph $G = (V, E)$ of G_i we say that x_i *breaks an exterior cycle of* G_i *w.r.t.* G if there is a cycle in $G_i - G$ which contains x_i.

Lemma 4.2.1. *Assume* $C = C_{l_1} = C_{l_2}$, *where* l_1 (l_2) *is in the left* (*right*) *subtree of* $T_i - T_{i_1}$ (T_{i_2}). *Then* x_i *breaks an exterior cycle in* G_i *w.r.t.* G, *where* $G = G_{l_1} = G_{l_2}$.

We denote by $\{D_1, \ldots, D_{\hat{N}}\}$ the distinct clauses in T and let $I_j = \{i \mid C_i = D_j\}$ for $1 \leq j \leq \hat{N}$. For r an ancestor of l in T let $m_{l,r}$ be the number of x_p's, on the path from r to l, that break an exterior cycle in G_p w.r.t. G_l.

Claim. $\sum_{l \in I_j \cap T_r} 2^{-m_{l,r}} \leq 1$ for $1 \leq j \leq \hat{N}$.

Proof. An easy induction on the height of T_r. The induction step follows from Lemma 4.2.1, since whenever D_j appears in both subtrees, $m_{l,r}$ grows by one. □

Thus,

$$\sum_{l \in I_j} 2^{-m_{l,1}} \leq 1 \quad \text{for all } 1 \leq j \leq \hat{N}. \tag{1}$$

Let n_l be the number of x_p's on the path from 1 to l that break a cycle in G_p (case 1 above), and recall s_l as defined in Section 2.

Lemma 4.2.2. $n_l = m_{l,1} + s_l$.

Hence by (1),

$$\sum_{i \in I_j} 2^{-n_i} \cdot 2^{s_i} \leq 1 \quad \text{for all } 1 \leq j \leq \hat{N}. \tag{2}$$

Summing up (2) for all j yields

$$\hat{N} \geq \sum_i 2^{-n_i} \cdot 2^{s_i} \equiv Y(T). \tag{3}$$

A closer look at $Y(T)$ will convince the reader that $Y(T)$ can be defined by the following recursion:

Let $Y(T_i) = 2^{s_i} + \hat{Y}(T_i)$, where

$$\hat{Y}(T_i) = \begin{cases} 0 & \text{if } G_i \text{ consists of a single vertex,} \\ Y(T_{i_1}) + Y(T_{i_2}) & \text{if } x_i \text{ splits } G_i, \\ \frac{1}{2}(Y(T_{i_1}) + Y(T_{i_2})) & \text{otherwise.} \end{cases}$$

Claim. $$Y(T) = Y(T_1). \tag{4}$$

Proof. In the final sum which is obtained from the recursion above, each G_i is represented by 2^{s_i}. The coefficient of 2^{s_i} is 2^{-n_i} since we have a factor of 1/2 any time x_i breaks a cycle in G_i. □

To get a lower bound for $Y(T_1)$, we define

$$Z(G) = 2^{s_G} + \min_{a \in E} Z^a(G),$$

where

$$Z^a(G) = \begin{cases} 0 & \text{if } G \text{ consists of a single vertex,} \\ Z(G') + Z(G'') & \text{if } a \text{ splits } G \text{ into } G' \text{ and } G'', \\ Z(G') & \text{otherwise, where } G' = (V, E - a). \end{cases}$$

Claim. $$Y(T_1) \geq Z(\bar{G}). \tag{5}$$

Proof. $Y(T_i) \geq Z(G_i)$ is easily shown by induction on the height of T_i. □

Thus, by (3), (4) and (5) we get

$$\hat{N} \geq Z(\bar{G}). \tag{6}$$

But,

$$Z(\bar{G}) = \sum_i 2^{s_i}, \tag{7}$$

where $s_{(i)} = s_{G^{(i)}}$ and the $G^{(i)}$'s are the distinct subgraphs which were obtained in the deletion process which is implied by the definition of $Z(\bar{G})$.

We now specify $\bar{G}$ as in Section 2. Of all the $G^{(i)}$'s we can choose a sequence which satisfies the conditions of Lemma 2.3.1. Hence, there is an i such that $|F^{ext}_{(i)}| \geq c'n$. So, by Lemma 2.3.2,

$$s_{(i)} \geq cn. \tag{8}$$

By (6), (7) and (8), $\hat{N} \geq 2^{cn}$ and we have just proved:

Theorem 4.2.1. *$Comp_{Reg}(n) \geq 2^{cn}$ for some $c > 0$.*

Corollary 4.2.1. *DPP with subsumption is exponential.*

The direct proof for Corollary 4.2.1 is much simpler than the proof of Theorem 4.2.1. It does not use Lemma 4.2.1 and Lemma 4.2.2. (See [7] or [9].)

Note that by (6) and (7), $\hat{N} \geq \sum_i 2^{s_{(i)}}$. But actually, $\hat{N} = \sum_i 2^{s_{(i)}}$. The second inequality follows from the following observation: Consider the deletion process implied by the definition of $Z(\bar{G})$. By Lemma 3.3.1 it has a corresponding regular proof tree $\hat{T}$.

Lemma 4.2.3. *The number of distinct clauses on $\hat{T}$ associated with $G^{(i)}$ is $\leq 2^{s_{(i)}}$.*

Hence, $N(\hat{T}) \leq \sum_i 2^{s_{(i)}}$, and $\hat{T}$ must be optimal. Note that $\hat{T}$ is a DP tree. This implies:

Theorem 4.2.2. *For sets of clauses associated with graphs, regular resolution is no more powerful than the Davis–Putnam procedure.*

Note that the observation above implies that there always exists an optimal DP tree. However, one can easily show that an optimal tree is not necessarily a DP tree.

4.3 Proofs of the Lemmas

Proof of Lemma 4.2.1. As we observed above, x_i must break a cycle in G_i. x_i does not belong to the boundary of G, since if it did, then because the tree is regular, the Correspondence Lemma would imply that $x_i \in C_{l_1}$ and $\bar{x}_i \in C_{l_2}$ — a contradiction ($C_{l_1} = C_{l_2}$).

Assume x_i does not break an exterior cycle in G_i w.r.t. G. Then one of the connected components of $G_i - G$ must be split in two by deleting x_i:

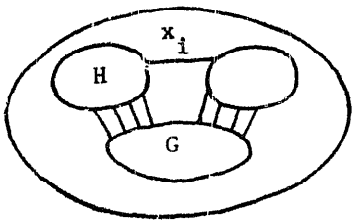

Fig. 4.3.1. G_i.

Let H be one of the two.

For r on the path from i_1 to l_1, let $H_r = H \cap G_r$ and let

$$\delta_r = \sum_{z_m \in F^{ext}_{H_r},\, z_m^{a_m} \in C_r} (1 - a_m),$$

i.e., δ_r is the contribution to the parity of C_r by edges in the exterior boundary of H_r.

Claim. $$\delta_{i_1} = \delta_{l_1} \oplus \varepsilon(H). \tag{9}$$

Similarly, $\delta_{i_2} = \delta_{l_2} \oplus \varepsilon(H)$. But $\delta_{l_1} = \delta_{l_2}$ and $\delta_{i_1} \neq \delta_{i_2}$. ($C_{i_1} = C_i \vee x_i$, $C_{i_2} = C_i \vee \bar{x}_i$ and $x_i \in F^{ext}_{H_{i_1}} = F^{ext}_{H_{i_2}}$). This contradiction proves the lemma.

To prove the claim we show by induction on the distance of r from i_1 that

$$\delta_{i_1} = \delta_r \oplus \varepsilon(H - H_r). \tag{10}$$

This is obviously true for $r = i_1$. Assume that the result holds for r, and let r_1, r_2 be the sons of r, r_1 is on the path to l_1. The only case in which δ_r (or $\varepsilon(H - H_r)$) can be different from δ_{r_1} (or $\varepsilon(H - H_{r_1})$, respectively) is when the deletion of x_r from G_r splits it into G_{r_1} and $G_{r_2} \subseteq H_r$.

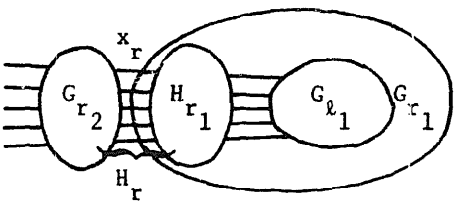

Fig. 4.3.2. G_r.

δ_r has some terms deleted and some new terms added. Since in mod 2 sum "subtracting" is the same as "adding",

$$\delta_{r_1} = \delta_r \oplus \underbrace{\text{terms added} \oplus \text{terms deleted}}_{\lambda_r}. \tag{11}$$

But,

$$\lambda_r = \varepsilon(G_{r_2}). \tag{12}$$

This follows from

(a) $\lambda_r = \sum_{x_j \in F^{ext}_{G_{r_2}}} (1 - a'_j)$,

(b) $\varepsilon(G_{r_2}) \neq \delta(C_{r_2}) = \sum_{x_j \in F^{ext}_{G_{r_2}}} (1 - a_j)$, and

(c) $a'_r \neq a_r$ and $a'_j = a_j$ for all $j \neq r$.

also, $H - H_{r_1} = (H - H_r) \cup G_{r_2}$ and the two subgraphs are vertex disjoint. Thus,

$$\varepsilon(H - H_{r_1}) = \varepsilon(H - H_r) \oplus \varepsilon(G_{r_2}). \tag{13}$$

Substituting (11)–(13) into (10) yields (10) for r_1. □

Proof of Lemma 4.2.2. Let $G = G_l$ and let $\{H_j\}_{j=1}^m$, $\{k_j\}_{j=1}^m$, s_l, F_l^{int} be as defined in Section 2. Recall that all of these were defined by considering G as a subgraph of $\bar{G}$. In this proof only we consider G as a subgraph of G_i for each i, an ancestor of l. We will add the index i to get the corresponding parameter, e.g., $F_{l,i}^{int}$ is the interior boundary of G_l w.r.t. G_i, etc.

By definition of $s_{l,i}$

$$s_{l,i} = |F_{l,i}^{int}| + \sum_{j=1}^{m_i} (k_{j,i} - 1). \tag{14}$$

Claim. $s_{l,i}$ is the number of variables x_j from i to l that break a cycle in G_j, but not an exterior cycle with respect to G_l.

The claim proves the lemma since for $i = 1$, $G_1 = \bar{G}$, and hence the number of times x_j breaks a cycle in G_j but not an exterior cycle w.r.t. G_l is $n_l - m_{l,1} = s_{l,1} = s_l$.

The claim is easily proved by induction on the distance of i from l. For $i = l$, $0 = 0$. Assume it holds for i_1 the son of i. If x_i splits G_i, all the terms for i_1 in (14) are the same as for i (also if x_i is in the boundary of G, since the corresponding k_{j,i_1} is 1 and does not contribute anything to the sum in (14)). If x_i breaks an exterior cycle in G_i w.r.t. G_l nothing changes either. Otherwise,

(a) if $x_i \in F^{int}$, then $|F_{l,i}^{int}| = |F_{l,i_1}^{int}| + 1$

(b) if $x_i \in F^{ext}$, then

$$\sum_{j=1}^{m_i = m_{i_1}} (k_{j,i} - 1) = \sum_{j=1}^{m_{i_1}} (k_{j,i_1} - 1) + 1$$

and

(c) if x_i is as in Figure 4.3.1 (the remaining case), then

$$\sum_{j=1}^{m_i = m_{i_1} - 1} (k_{j,i} - 1) = \sum_{j=1}^{m_{i_1}} (k_{j,i_1} - 1) + 1.$$

In (b) one of the k_{j,i_1}'s increases by 1 and in (c) one of the $k_{j,i}$'s splits into two: k_{j_1,i_1} and k_{j_2,i_1}, and the one is subtracted twice. Thus, the claim holds for i. □

Proof of Lemma 4.2.3. It is implicit in the proof of Lemma 4.2.1 that if i is an ancestor of l in a proof tree, then the parity of the number of complemented variables corresponding to edges which connect G_l with any one of the connected components of $G_i - G_l$ is determined given G_i and G_l and does not depend on the path from i to l. (This is exactly the meaning of (9).) Thus, taking $i = 1$, G_l can have at most

$$2^{|F_l^{\text{int}}|+\Sigma_j(k_j-1)} = 2^{s_l}$$

distinct clauses associated with it. (2^{k_j-1} is exactly half of the ways to assign complements to the edges which connect G_l and H_j. Indeed, exactly half of them satisfy the parity mentioned above.) □

4.3 Comparison to Tseitin's proof

Our method of proof is essentially indentical to that of Tseitin. Tseitin introduced the way of associating sets of clauses with graphs, and lemmas 2.3.1, 2.3.2 and 2.3.3 are due to him. Tseitin then introduced, what we call here, deletion process and showed that given a regular proof tree there is a deletion process which corresponds to it. To get the lower bound, Tseitin defined the function $Z(G)$ and then showed that $\hat{N} \geq Z(\bar{G})$. This last part, which is the hardest in the proof, is where our proof proceeds differently. Thus, neither Lemma 4.2.1 nor Lemma 4.2.2 can be found in Tseitin's proof. The comparison at this point is difficult due to the fact that almost no details are given by Tseitin.

Our improvement over Tseitin's result is due to the choice of the graphs. Except for the last stage, the proof does not depend on the specific graphs. The graphs are chosen so that a large subgraph G should have large s_G, which is achieved in our case by Lemmas 2.3.1 and 2.3.2. The bound depends essentially on how large s_G can be. Originally Tseitin chose the $k \times k$ square grids. In that case, $n = O(k^2)$, and using similar techniques he could show that there is an i such that $s_{(i)} \geq ck$ (instead of (8)). As a result he obtained a $2^{c\sqrt{n}}$ lower bound. In his direct proof for an exponential lower bound for the DPP, Kirkpatrick [9] considered other graphs which are obtained by a simple transformation from the k-dimensional cubes for $k \geq 3$. We adopted these graphs in [7] to increase the bound to $2^{cn/(\log n)^2}$. Finally, the choice of Margulis' graphs enable us to improve the lower bound to 2^{cn}.

Although we have improved Tseitin's result, we feel that the main contribution of this paper is clarifying and simplifying an important result which would be almost inaccessible otherwise.

5. Conclusion and open problems

We saw that both the Davis–Putnam procedure and regular resolution must generate an exponential number of distinct clauses when the inputs are sets of

clauses which correspond to some graphs. However, these inputs are not inherently hard. In fact Tseitin observed:

Theorem 5.1. *For every set of clauses α which correspond to a graph there is a proof tree T, which uses extension, such that the number of distinct clauses on T is polynomial in $|\alpha|$.*

In fact there is a proof tree with $O(|\alpha| \log |\alpha|)$ distinct clauses. For proof see [7] or [9].

As we mentioned in the introduction, the complexity of resolution is still open. Although we feel that it must be exponential on the graph inputs, we could not even show that $\text{Comp}(n) \geq cn$ for all c. One possible way to settle this question might be to find a way to transform every resolution tree T to a regular tree T', so that $N(T') \leq p(N(T))$ where p is a polynomial. One can easily do it by using extension ([7]):

Theorem 5.2. *Using extension, we can transform every resolution tree T into a regular proof tree T', so that $N(T') = O((N(T)^3))$.*

But without extension we do not know how to do it. On the other hand we could not give examples for an input α such that there exists a non-regular resolution tree T using α with complexity smaller than the complexity of all regular trees using α.

Another interesting problem is whether regular resolution is more powerful than the DPP. Theorem 4.1.2 indicates that if the answer is affirmative, then one will need new contradictory sets of clauses, other than those corresponding to graphs to prove this.

The last problem which seems to be the hardest and is probably the most important is to determine the complexity of extended resolution. In fact we believe that the methods described in Section 2 to generate contradictory formulas cannot be used to show that extended resolution is exponential. Moreover, we feel that this open problem will be settled only when the two open problems involving P and NP are solved.

Acknowledgments

I am indebted to Professor John Hopcroft, and to Professor Juris Hartmanis, for their constant help, advice and encouragement. I would also like to thank Peter van emde Boas, David Kirkpatrick, Wolfgang Paul, Janos Simon, and Marvin Solomon for many helpful discussions. Finally, I would like to thank Nick Pippenger who advised me on Margulis' paper and helped me to improve the previous $2^{cn/(\log n)^2}$ lower bound for regular resolution to 2^{cn}.

References

[1] M. Bauer, D. Brand, M.J. Fischer, A.R. Meyer and M.S. Paterson, A note on disjunctive form tautologies, *SIGACT News* (April 1973) 17–20.

[2] S.A. Cook, The complexity of theorem proving procedures, *Proc. 3rd ACM STOC* (1971) 151–158.

[3] S.A. Cook, Examples for the Davis–Putnam procedure, unpublished manuscript (June 1971).

[4] S.A. Cook and R.A. Reckhow, On the length of proofs in the propositional calculus, *Proc. 6th ACM STOC*, (1974) 135–148.

[5] M. Davis and H. Putnam, A computing procedure for quantification theory, *J. ACM* **1** (1960) 201–215.

[6] B. Dunham and H. Wang, Toward feasible solutions of the tautology problem, IBM Thomas J. Watson Research Center RC 4924 (July 1974).

[7] Z. Galil, The Complexity of Resolution Procedures for Theorem Proving in the Propositional Calculus, Ph.D. Thesis, TR 75–239, Dept. of Computer Science, Cornell University (May 1975).

[8] R.M. Karp, Reducibility among combinatorial problems in: R.E. Miller and J.W. Thatcher eds., *Complexity of Computer Computations* (Plenum Press, New York, 1972) 85–104.

[9] D.G. Kirkpatrick, Topics in the complexity of combinatorial algorithms, Ph.D. Thesis, Technical Report No. 74, Dept. of Computer Science, University of Toronto (December 1974).

[10] G.A. Margulis, Explicit construction of concentrators (translated from Russian), *Probl. Inf. Transm.* **9** (1973) 325–332.

[11] W.V. Quine, A way to simplify truth functions, *Amer. Math. Monthly* **62** (1956) 627–631.

[12] J.A. Robinson, A machine oriented logic based on the resolution principle *J. ACM* **12** (1965) 23–41.

[13] I. Simon, On the time required by the Davis–Putnam tautology recognition algorithm, Research Report CSRR–2050, Dept. of Computer Science, University of Waterloo (June 1971).

[14] G.S. Tseitin, On the complexity of derivations in the propositional calculus, in: A.O. Slisenko ed., *Structures in Constructive Mathematics and Mathematical Logic, Part II* (translated from Russian) (1968) 115–125.