THE COMPLEXITY OF RESOLUTION PROCEDURES FOR THEOREM PROVING IN THE PROPOSITIONAL CALCULUS+

Zvi Galil

TR 75-239

May 1975

Department of Computer Science
Cornell University
Ithaca, NY 14853

+This research was partially supported by ONR Grant N00014-67-A-0077-0021.

THE COMPLEXITY OF RESOLUTION PROCEDURES FOR THEOREM PROVING IN THE PROPOSITIONAL CALCULUS

Zvi Galil, Ph.D.
Cornell University 1975

ABSTRACT:

A comparative study on the complexity of various procedures for proving that a set of clauses is contradictory is described. All the procedures either use the resolution rule in some form or are closely related to procedures which do. Among the procedures considered are 1. resolution 2. regular resolution 3. Davis Putnam procedure 4. resolution with extension 5. bounded (and iterated bounded) resolution 6. enumeration procedures 7. semantic trees.

The results include

a. exponential lower bounds for the run-time of most of the procedures,

b. relations between the various procedures,

c. implications to the complexity of integer programming routines.

BIOGRAPHICAL SKETCH

Zvi Galil was born on June 26, 1947 in Tel Aviv, Israel. In June 1970, he received the B.Sc. degree summa cum laude in Applied Mathematics from Tel Aviv University. In September 1971, he received his M.Sc. degree summa cum laude in Applied Mathematics from Tel Aviv University. In September 1972, he entered the graduate school at Cornell University.

TO MY PARENTS

ACKNOWLEDGMENTS

I am deeply indebted to my thesis advisor, Professor John Hopcroft, and to Professor Juris Hartmanis, for their constant help, advice and encouragement.

Thanks are due to Profesors David Gries and William Lucas, for serving on my special committee.

During my years at Cornell, I have enjoyed very much the excellent research environment in the Computer Science Department. I have had many opportunities to talk to the professors of the department and to my fellow students. Besides Professors Hopcroft and Hartmanis, I would like to thank Professors Robert Constable and David Kirkpatrick, as well as Peter van emde Boas, Kurt Mehlhorn, Wolfgang Paul, Janos Simon, and Marvin Solomon.

Thanks are also due to Professor Amir Pnueli at Tel Aviv University in Israel, my first teacher in Computer Science.

Finally, I wish to thank my wife, Belli, for her love, patience, and encouragement, without which this thesis would not have been possible.

TABLE OF CONTENTS

1. Introduction.

1.1. The Underlying Problem.

One important problem in computer science is the "tautology problem": given a Boolean formula in disjunctive normal 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 iff 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 [4]; and

2) There is a nondeterministic polynomial time bounded algorithm for tautology iff NP is closed under complementation [6].

Thus, the tautology problem is closely related to two of the most important open problems in the theory of computational complexity. The dual problem of the tautology problem is: determine if a Boolean formula in conjuctive normal 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 laws. The second is more natural in theorem proving. To prove a theorem from a set of axioms, take the conjuction of all axioms plus the negation of the theorem and prove a contradiction.

A popular procedure to prove that a set of clauses is contradictory is the resolution procedure [26]. 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 [24], uses the consensus rule to generate new conjunctions. The input is a tautology iff the empty conjunction can be obtained. In this thesis we investigate various procedures which check whether their input, which consists of a set of clauses, is contradictory or not. All the results can be restated and will hold automatically for the dual problem. We consider various procedures which either use the resolution rule or are closely related to such procedures, and compare their computing powers. For most of them we will prove exponential lower bounds on their run-time.

1.2. Notation.

We assume that there is an unbounded set $\{x_1,x_2,\dots,y,\dots\}$ of <u>variables</u>. A <u>literal</u> 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 <u>clause</u> is a finite set of literals $(\ell_1,\dots,\ell_k)$ such that no variable appears more than once, and is denoted by $\ell_1 \vee \ell_2 \vee \dots \vee \ell_k$. A <u>formula</u> (in conjunctive normal form (CNF)) is a finite set of clauses. A <u>valuation</u> (or <u>assignment</u>) 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 <u>value</u> (relative to v) of the literal x $(\bar{x})$ is $v(x)$ $(1-v(x))$. The value of the clause $\ell_1 \vee \ell_2 \vee \ldots \vee \ell_k$ 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.

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 CNF with variables $\{x_1,\ldots,x_k\}$ It is also known ([4], [2]) that any formula in CNF of length n can be transformed, using additional variables, into another formula of length $O(n)$ † in CNF with at most three literals per clause, such that the second formula is satisfiable iff the original formula is satisfiable We denote the latter by 3-CNF.

In this thesis we are interested in the following problem: given a formula in CNF (i.e., a set of clauses), is it contradictory?

1.3. Resolution.

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 $C_2 \vee \overline{x}$ clash, then their resolvent is the

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

clause $C_1 \vee C_2$. We will say that $C_1 \vee C_2$ is obtained from $C_1 \vee x$ and $C_2 \vee \overline{x}$ 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 [26] showed that a set of clauses is unsatisfiable iff there is a resolution proof of its unsatisfiability: The "if part' follows from the fact that if an assignment satisfies two clauses (that can be resolved), then it satisfies their resolvent. A proof for the 'only if part' will be indicated later.

We will extensively use binary trees to represent (among others) resolution proofs. A tree T is a directed graph with a unique vertex which is called the root, such that (i) no edge enters to the root, (ii) to every vertex in T there is a unique path from the root, and (iii) there are no cycles in T . The vertices of the tree will be denoted by integers, 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. $i \in T$ means i is a vertex of 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 $T_1 = T$. We sometimes use the same vertex name for two vertices in different trees. For example, the vertex i in T_i is not the same as the vertex i in T'_i. In particular we will use the same name when the trees correspond. No confusion will arise.

A <u>(resolution) proof tree for a clause C (using S)</u> 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.

If the clauses C_1 and C_2 have proof trees T_1 and T_2, and can be resolved to get C, we define the corresponding operation on trees $T = \mathrm{Res}(T_1,T_2)$. T is the proof tree for C whose two main subtrees are T_1 and T_2.

A <u>(resolution) proof tree (using S)</u> 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 $S = \{\bar{u}, u \vee \bar{x}, x \vee y, \bar{y} \vee u\}$. Figure 1.3.1. describes a proof tree using S .

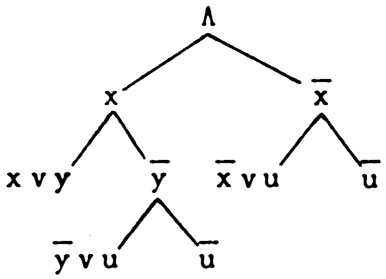


Figure 1.3.1.

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

Let $|S|$ be the number of literals in S and let $\mathcal{T}_S = \{T|T$ 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 and we develop some tools which we hope will be used to prove it. However, so far we could not prove that $\mathrm{Comp}(n)$ 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 is regular iff 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 Figure 1.3.1. is regular. The occurrence of an irregularity is described in Figure 1.4.1.

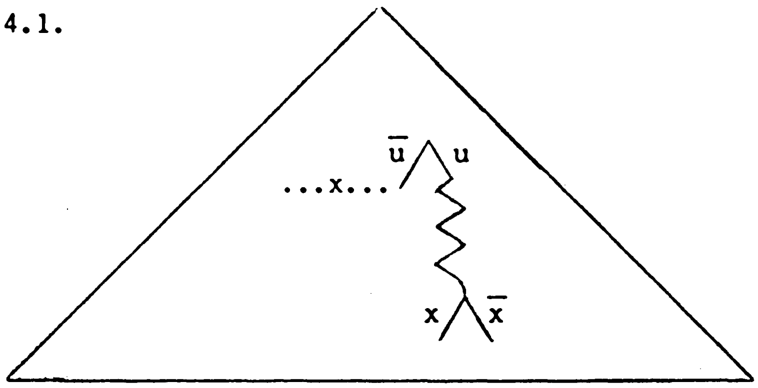


Figure 1.4.1.

Regular resolution is simply constructing a regular resolution proof tree using S . One should note that a set of clauses S is contradictory iff there is a regular proof tree using S . The 'if part' is obvious. We will indicate how to prove the 'only if part' in the sequel. (Thus proving also the 'only if part' for unrestricted resolution.) The complexity of regular resolution, $\mathrm{Comp}_{\mathrm{Reg}}$ is defined as Comp except that the minimum is taken only over regular trees. Tseitin [32] 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. Nevertheless, it was Tseitin's paper which influenced us the most, and most of the results in this thesis were obtained by developing Tseitin's techniques.

His main contribution was an ingenious construction which associates sets of clauses with graphs and then observing that regular resolution corresponds to a process of eliminating edges from the graphs. By choosing appropriate graphs and considering the graphs he was able to prove the exponential lower bound for regular resolution mentioned above.

1.5. The Davis Putnam Procedure.

The Davis Putnam procedure (DPP) [7] is the following routine:

I. 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.

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 <u>eliminating x</u> .

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 ([5], [28]). 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) .

From now on we will assume DPP includes the subsumption rule (between steps II and III). 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 iff 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 [6].)

The other procedures: bounded (and iterated bounded) resolution procedures, extended resolution and enumeration procedures will be defined in the sections which are devoted to them.

1.6. Outline of the Thesis.

Sections 2 and 3 are introductory sections in which we develop some tools which will be used throughout the thesis. Section 2 is devoted to graphs and to the way one can associate sets of clauses with them. In Section 3 we prove some general lemmas about resolution proof trees. Some of the lemmas deal with proof trees using sets of clauses associated with graphs. Although we have not been able to show that the unrestricted resolution procedure has an exponential lower bound, we believe that such lemmas as those in Section 3 might be used to prove this result.

In Section 4 we introduce several versions of bounded resolution and iterated bounded resolution. They all use the resolution rule but allow generation of new clauses only if their size is not larger than some given bound. In the case of iterated bounded resolution the bound is increased whenever the current bound is found to be too small. We prove: (1) no proposed bounded resolution procedure always works correctly; and (2) all iterated bounded resolution procedures are exponential. Our attempts to find straightforward proofs for these results have failed. Our proofs make a nontrivial use of Tseitin's sets of clauses which correspond to graphs.

In Section 5 we give a simplified, hopefully intuitive, and detailed proof for Tseitin's main result, i.e., that regular resolution is exponential. We improve the lower bound and show that $\mathrm{Comp}_{\mathrm{Reg}}(n) \geq 2^{n/72(\log n)^2}$. In this section we also sketch a direct proof to an exponential lower bound for the Davis Putnam procedure. Actually, it follows almost immediately from one of the lemmas. The more general result, the exponential lower bound for regular resolution, follows from a much more involved proof.

Section 6 is devoted to extended resolution, a powerful generalization of the resolution procedure. We show that every proof tree using resolution can be transformed into another proof tree using extended resolution such that (1) the tree is regular, and (2) it contains only clauses of size at most three. The new tree will not use many more clauses than the original one. While (1) is not known to hold for resolution, (2) is shown not to hold for resolution in Section 4. We also include Tseitin's observation that extended resolution gives a polynomial proof for contradictory systems of clauses associated with graphs. The above does not hold for regular resolution and we conjecture that it also fails for unrestircted resolution but we were not able to prove this result.

Section 7 is devoted to enumeration procedures. These procedures enumerate in a special way all possible assignments by constructing enumeration trees. We prove a relationship between these trees and and regular proof trees. Their relationship to semantic trees [6] is described too. We then prove that these procedures have exponential lower bounds. We conclude this section by discussing some implications of our results to the complexity of integer programming routines.

Finally, Section 8 contains a summary and a list of open problems.

1.7. Some General Remarks.

In this section we make several remarks about two important themes in the theory of computational complexity: 1) nondeterminism and 2) lower bounds. We also describe how our results are related to the current state of knowledge in these two areas.

A nondeterministic procedure is a procedure which can have more than one choice for next move. For example the DPP includes the step "choose a variable x." but it does not specify which variable is chosen. Similarly the regular (and unrestricted) resolution does not specify the way in which the proof tree should be constructed. All other procedures that we deal with, except bounded (and iterated bounded) resolution, are also nondeterministic. The run-time of such procedures is defined to be the shortest accepting sequence of moves[†], i.e., the run-time of a hypothetical deterministic procedure (which probably does not exist) which knows how to make the best choice whenever it has to execute a move which is not specified completely. In this light the definition of $\text{Comp}(n)$ $(\text{Comp}_{\text{Reg}}(n))$ can easily be motivated. The complexity is proportional to the run-time of the nondeterministic procedure for constructing a resolution proof (or resolution proof tree without repeating redundant subtrees). The run-time is for the 'worst-case' input of size n $(|S| = n)$.

Today, two basic questions about nondeterminism are widely studied:

1) What is the "cost" of nondeterminism? Since (unfortunately) computers can only execute deterministic programs one would like to know what is the best way to transform a nondeterministic procedure to a deterministic one. For example, the SATISFIABILITY problem (the complement of our underlying problem) can be solved by a nondeterministic procedure which runs in linear time. The procedure just guesses the values of all the variables and then it checks to see if the assignment satisfies the

---

† We consider procedures with 1,0 (Yes-No, accept, reject) output. The run-time is defined only for accepting computations.

given set of clauses. At present, nobody knows how to solve SATISFIABILITY deterministically in less than exponential time. Finding whether SATISFIABILITY can be solved deterministically in polynomial time will settle the 'P=NP?' question since P=NP iff SATISFIABILITY $\in$ P . 'P=NP?' is only one (but probably the most important) instance of question 1) above. For other examples see [30]. The importance of 'P=NP?' comes from two sources: a) It is believed that P is the class of all feasible problems; and b) many combinatorial problems can easily be shown to be in NP , by a procedure which guesses the solution, and then verifies that it is indeed a solution. Some of them also have the property that they belong to P iff P=NP . These are the NP-complete problems ([4], [17] and others).

2) Is there always a good nondeterministic procedure for a problem if its complement has one? One of the instances of this basic question is 'Is NP closed under complementation?', i.e., if a problem can be solved by a nondeterministic procedure in polynomial time can we say the same about its complement? Although SATISFIABILITY can be solved in linear time by a nondeterministic procedure we do not know if there is a nondeterministic procedure which proves that a set of clauses is contradictory and has a polynomial run-time. In fact NP is closed under complementation iff such a procedure exists.

Most of this thesis investigates the run-time of nondeterministic procedures for the underlying problem. There are several theoretical justifications for interest in nondeterministic procedures:

a) As was mentioned above, the existence of a good nondeterministic algorithm for our problem is closely related to an important open problem, namely: 'Is NP closed under complementation?'

b) Assuming one is interested in size of a proof that a set of clauses is contradictory, and he is not interested how this proof has been constructed, the nondeterministic procedures are of great interest, since as was pointed out above, their run-time is closely related to the size of the proof of contradiction. (In this sense NP can be viewed as all theorems which have short proofs.)

There is also a more practical motivation which we hope will satisfy those people who claim (perhaps with some justification) that nondeterministic procedures should not be considered as algorithms.

c) A nondeterministic procedure can be viewed as a family of deterministic algorithms. The different algorithms in the family differ in the heuristic they use for the unspecified steps in the nondeterministic procedure. More specifically, if there are several possible choices for the next step, and the nondeterministic procedure leaves unspecified which one of them is to be executed, then the deterministic procedures in the family may differ in their choice. By proving a lower bound on the run-time of a nondeterministic procedure we automatically prove the same lower bound for all the deterministic procedures in the corresponding family.

The second topic that we discuss is proving <u>lower bounds</u>: Given a computational task, there are two basic approaches. The first approach is to construct upper bounds. This is done by designing good algorithms to solve the problem. The upper bound is thus the run-time of the algorithm and is improved whenever a faster algorithm is constructed. The second approach, in which we are more interested, is determining the inherent complexity of the problem by proving lower bounds on the run-time of all possible algorithms which solve the problem.

One might think that the main goal should be to find the best algorithm for the given problem. However, there are problems such that no algorithm for them can be the best one [3]. Thus, the goal is actually to prove close lower and upper bounds. The closer these bounds are, the better we know the "real" complexity of the given problem.

As for upper bounds, considerable progress has been made during the last five years. Many good algorithms have been constructed and various techniques were developed to assist in designing fast algorithms ([1], [20]). On the other hand, not much is known about lower bounds. One of the earliest results in the theory of computational complexity claims that for a reasonable time bound $T_2(n)$ †, there is a problem which can be solved in time $T_2(n)$ but not in time $T_1(n)$ if $\lim\sup[T_1(n)\log T_1(n)/T_2(n)] = 0$ [13]. Simply stated if $T_2(n)$ is slightly larger than $T_1(n)$, there is a problem which can be solved in time $T_2(n)$ but it requires more time than $T_1(n)$. This was proved by means of diagonalization. Until recently, however, people could not exhibit natural problems that satisfy the above except proving the (trivial) lower bound n by the following (trivial) proof: If the output depends on the entire input, then every algorithm should

† More precisely $T_2(n)$ is time countable (see [14]).

read the complete input; this must take at least n steps... The inability to prove lower bounds comes from the fact that to show that a problem requires time $T(n)$ means to show that every algorithm for the problem must run in time larger than $T(n)$. So we must consider all possible algorithms. This is thus a very hard task, since some algorithms might use some clever tricks from number theory. Nevertheless, there was a recent breakthrough, and nontrivial lower bounds have been proved [21]. Roughly speaking, whenever the problem involves systems with rich expressive power such that a class of computations can be efficiently enconded into instances of the problem, then the problem can be shown not to be in that class. [31] includes many such problems and references to many others. Here too use was made of diagonalization, but in an indirect manner. The lower bounds proved are exponential (2^{cn}), double exponential ($2^{2^{cn}}$) and larger. Thus, if we consider P to be the class of feasible computations, then these problems were proved to be unfeasible. At present we cannot prove any nontrivial lower bound for feasible problems; e.g., we cannot exhibit a natural problem which can be solved in time n^3 but cannot be solved in time n^2 . Nontrivial lower bounds are also not known for those problems which belong to the fuzzy region between feasible problems and nonfeasible ones, namely the NP-complete problems and their complements. Finding tight lower bounds for these will settle 'P=NP?' and may also settle 'Is NP closed under complementation?'. But at present, we cannot even prove much weaker results such as that SATISFIABILITY requires time $n \cdot \log n$. It was recently observed that the problem of proving nontrivial lower bounds for the run-time for computing a function can be reduced to proving lower bound for the size of circuits computing the truncated function†. More specifically, if the circuit must be of size $T(n)\log T(n)$, then the time of the Turing machine must be larger than $T(n)$ [9]. Thus, attention has been devoted to trying to prove nonlinear lower bounds for the size of the corresponding circuits. It is known that most functions require a circuit of exponential size [27], and indeed the truncated versions of the hard problems mentioned above have been shown to require a circuit of exponential size [31]. However, these problems are already known to be unfeasible. For all the others the best lower bound currently known is $2.5n$ [22]..

Facing this immense difficulty in proving general lower bounds, some researchers have shifted their attention to more special lower bounds. Instead of proving that certain problems require time $T(n)$ on all possible algorithms, they have restricted attention to what they consider natural algorithms for solving the problems. As examples, we can mention (1) Tree programs for selection and sorting problems counting only the number of comparisons, and (2) straight line programs for computing some algebraic expressions over some set of indeterminates, counting the number of basic arithmetic operations in the sequence. For example, matrix multiplication and polynomial evaluation (cf. [3]). Other researchers have considered restricted models, such as on-line Turing machines [11], and monotone circuits [23]. Imposing the restric-

† Given a function on binary strings and an integer n, then the n-th truncation of f is the restriction of f to inputs of size exactly n.

tions above enabled them to prove some nontrivial lower bounds. But many questions are still open even in these restricted models.

Our results are of this second type. We consider various procedures for proving that a given set of clauses is contradictory. Each nondeterministic procedure can be viewed as a collection of deterministic algorithms. Each procedure represents a sort of natural approach to solve the problem. For most of them we prove an exponential lower bound on their complexity.

Considering procedures for proving unsatisfiability of a set of clauses, there are two kinds of results one would like to obtain. The first is to show that a given procedure has a polynomial complexity. This would imply $P=NP$ if the procedure is deterministic and would imply that NP is closed under complementation if it is nondeterministic. So far none of the procedures has been shown to have a polynomial complexity and many researchers conjecture that none exists since they believe that $P \neq NP$ and NP is not closed under complementation.

The second is to show that known procedures have exponential complexity. By doing this to a given procedure, one excludes it from being a candidate for the first kind of results. Although this does not imply anything about the open problems mentioned above, one hopes that obtaining exponential lower bounds for the run-time of specific procedures (which is in some cases not an easy task) will require the development of techniques which will help to analyze the run-time of procedures in general.

2. Graphs and Sets of Clauses Associated with Them.

2.1. Notation.

We will extensively use 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 $\bar{G}$:

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

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

$G = G_1 \cap G_2$ if $V = V_1 \cap V_2$ and $E = E_1 \cap E_2$

$G = G_1 - G_2$ if $V = V_1 - V_2$ and E consists of all edges in E_1 between 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 .

Let $G = (V,E)$ be a subgraph of $\overline{G}$. The <u>boundary of G</u> , F (or F_G when G is not clear from the context) is the set of edges of $\overline{G}$ which are incident with at least one vertex of G and do not belong to E . The <u>exterior boundary</u> (<u>interior boundary</u>) 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 $\overline{G}$ without disconnecting it. Let $\{H_j\}_{j=1}^m$ be the connected components of $\overline{G} - G$ and let k_j , $1 \leq j \leq m$ be the number of edges connecting G and H_j . By definition we have

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

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

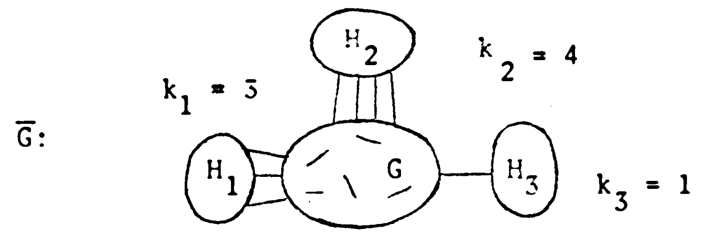


Figure 2.1.1. $s_G = 10$ $(|F^{int}| = 5)$

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

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

The proof follows immediately from (1), (2) and the fact that $k_j \geq 3$ for $1 \leq j \leq m$ since $\overline{G}$ 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 CNF over n variables which contains a clauses of size k such that the variables in any term are either all complemented or all noncomplemented. For every $n \geq 0$ and $k \leq \lceil n/2 \rceil$ $\beta_{n,k}$ 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 princip choose a set of clauses which altogether exclude the 2^n possible assi ments (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. [8] page 45). The statement is, "n objects can occupy all of n+1 holes". Another

---

† $\lceil n/2 \rceil$ ($\lfloor n/2 \rfloor$) is the smallest (largest) integer greater (smaller) th or equal to $n/2$.

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 CNF 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 CNF 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 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. $G(\ell)$ 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 iff

$$C = z_1^{\varepsilon_1} \vee \ldots \vee z_k^{\varepsilon_k}$$

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

$$\sum_{i=1}^{k} (1-\varepsilon_i) \neq \varepsilon_v$$

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 iff

---

† $\sum_{i=1}^{k} \delta_i = \delta_1 + \delta_2 + \ldots + \delta_k$ is the sum modulo 2 of $\delta_1, \delta_2, \ldots, \delta_k$.

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

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

Assume $\sum_{i=1}^{k} \delta_i = \varepsilon_v$ and let $C = z_1^{\varepsilon_1} \vee \ldots \vee z_k^{\varepsilon_k}$ be an arbitrary clause corresponding to v. The only assignment which does not satisfy C is $z_1 = 1 - \varepsilon_1, \ldots, z_k = 1 - \varepsilon_k$. Since $\sum_{i=1}^{k} (1-\varepsilon_i) \neq \varepsilon_v$ it must differ from the assignment δ. Thus, δ must satisfy C. □

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

Claim. $\alpha(G(\ell))$ 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 $\alpha(G(\ell))$ 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 $\varepsilon(G(\ell)) = \sum_{v \in V} \varepsilon_v$.

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

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

Assume $\varepsilon(G(\ell)) = 0$. 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 $1 \le i \le n$ satisfies $\alpha(G(\ell))$. □

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

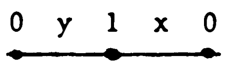
Let T be a proof tree using a contradictory set of clauses which corresponds to a graph, and let i be a vertex in T . We denote by V_i the set of all vertices which correspond to clauses which appear on the leaves of T_i .

The following lemma proves that every proof tree using $\alpha(G(\ell))$ must use at least one clause corresponding to each vertex in G .

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

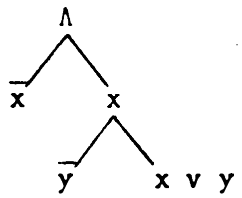
Proof. Assume the claim is wrong and that $v \in V - V_1$. Define a new labelling ℓ' by changing ε_v . By Lemma 2.3.2., $\alpha(G(\ell))$ is contradictory implies $\varepsilon(G(\ell)) = 1 \Rightarrow \varepsilon(G(\ell')) = 0 \Rightarrow \alpha(G(\ell'))$ is satisfiable. But all the clauses on the leaves of T belong also to $\alpha(G(\ell'))$ since none corresponds to v . Thus, we have a proof tree using a satisfiable set of clauses -- a contradiction which proves the lemma. $\square$

We have just proved that if T is a proof tree using $\alpha(G(\ell))$, 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 $\alpha(G(\ell))$ appear on leaves of T ? The answer is negative and is proved by the example in Figure 2.2.1.



The labelled graph $G(\ell)$

$$\alpha(G(\ell)) = \{\bar{x}, \bar{y}, x \vee y, \bar{x} \vee \bar{y}\}$$



A proof tree using $\alpha(G(\ell))$

Figure 2.2.1.

2.3. The Specific Collection of Graphs.

In this subsection we describe a specific graph $\bar{G}$. $\bar{G}$ will depend on a parameter k (or n) , and thus we actually deal with a collection of graphs. This collection was suggested by D. Kirkpatrick [18]. Originally we used the $k \times k$ square grid which Tseitin used. The latter yields slightly weaker results.

Let $\hat{G} = (\hat{V},\hat{E})$ be the k-cube, i.e.,

$$\hat{V} = \{\lambda = (\lambda_1,\ldots,\lambda_k) | \lambda_i \in (0,1)\}$$

$$\hat{E} = \{(\lambda,\mu) | \text{For exactly one } j ,\ \lambda_j \neq \mu_j\}$$

$\hat{G}$ has 2^k vertices and $k2^{k-1}$ edges. Note that the degree of $\hat{G}$ is $k = O(\log n)$. We now transform $\hat{G}$ into a new graph $\bar{G}$ with degree 3:

$$\bar{G} = (\bar{V},\bar{E})$$

$$\bar{V} = \{\lambda^{(i)} | 1 \leq i \leq k,\ \lambda \in \hat{V}\}$$

$$\bar{E} = \{(\lambda^{(i)},\mu^{(i)} | (\lambda,\mu) \in E,\ \lambda_i \neq \mu_i\}$$

$$\cup \{(\lambda^{(i)},\lambda^{(i+1 \bmod k)}) | \lambda \in \hat{V}\} .$$

The transformation is indicated by Figure 2.3.1.

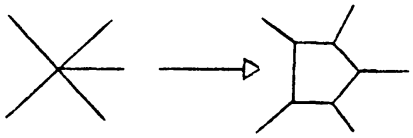
in $\hat{G}$  in $\bar{G}$

Figure 2.3.1. $(k = 5)$

$\bar{G}$ has $n = 1.5k2^k$ edges and $k2^k$ vertices. Let G_ℓ be a subgraph of $\bar{G}$. $\hat{G}_\ell = (\hat{V}_\ell,\hat{E}_\ell)$ is the <u>projection</u> of G_ℓ on the k-cube $\hat{G}$ if

$$|F^{ext}| \geq |F^{ext}_{(0)}| + |F^{ext}_{(1)}| + 2r_0 - r$$

$$\geq 2^{k-1} + 2r_0 - r - |r_0| - |r_1|$$

$$= 2^{k-1} + 2r_0 - r - r_0 + r - r_0$$

$$= 2^{k-1} \geq 2^{k-1} - |r| . \qquad \square$$

<u>Lemma 2.3.3</u>. If $G = (V,E)$ is a subgraph of $\overline{G}$ and $2^{k-2} \leq \#(V) < 2^{k-1}$, then $|F^{ext}_G| \geq 2^{k-2}/k$.

<u>Proof</u>. Let $\tilde{G} = (\tilde{V},\tilde{E})$ be the projection of G on the k-cube $\hat{G}$. $\tilde{G}$ satisfies the hypothesis of Lemma 2.3.2. since $\#(V) = |\tilde{V}|$. Thus, $|F^{ext}_{\tilde{G}}| \geq 2^{k-2}$. But $|F^{ext}_G| \geq |F^{ext}_{\tilde{G}}|/k$, since if any of the k edges incident with λ is in $F^{ext}_{\tilde{G}}$, then at least one of the edges incident with some $\lambda^{(i)}$ is in F^{ext}_G . Hence, $|F^{ext}_G| \geq 2^{k-2}/k$. $\square$

One can easily verify that $\overline{G}$ has edge connectivity 3. Thus, by Lemma 2.1.1. we have

<u>Lemma 2.3.4</u>. If G is a connected subgraph of $\overline{G}$, then $s_G \geq |F^{int}| + \frac{2}{3}\cdot|F^{ext}|$.

From now on we consider a fixed connected graph $\overline{G} = (\overline{V},\overline{E})$ with a fixed labelling ℓ such that $\varepsilon(\overline{G}(\ell)) = 1$. Thus, $\alpha(\overline{G}(\ell))$ is unsatisfiable. We will refer to $\alpha(\overline{G}(\ell))$ as the original set of clauses. We will identify the edges and their labellings if no confustion will arise. Without loss of generality we assume that the edge labels are all variables (i.e., they are noncomplemented). We do not specify $\overline{G}$, since most of the proofs do not depend on the specific graph. But at the end of each proof of a major result we will specify $\overline{G}$ as the graph described above. It might be the case that a better choice for $\overline{G}$ exists, which would slightly improve most of our results.

$$\hat{V}_\ell = \{\lambda \mid \lambda^{(i)} \in \cdot V_\ell \text{ for some } i\}$$

$$\hat{E}_\ell = \{(\lambda,\mu) \mid (\lambda^{(i)},\mu^{(i)}) \in E_\ell \text{ for some } i\} .$$

Similarly we will speak about $\hat{V}_\ell$ as the projection of V_ℓ .

Consider a proof tree T using a contradictory set of clauses which corresponds to $\overline{G}$, and recall V_i as defined in the previous subsection. Let $\#(V_i)$ be the number of vertices of $\hat{V}_i$, the projection of V_i in $\hat{G}$.

<u>Lemma 2.3.1</u>. There is an i_0 such that $2^{k-2} \leq \#(V_{i_0}) < 2^{k-1}$.

<u>Proof</u>. By Lemma 2.2.3., $\#(V_1) = 2^k$ since $V_1 = \overline{V}$. For a leaf, j , $\#(V_j) = 1$, since $|V_j| = 1$. If i_1 and i_2 are the sons of i and $\#(V_{i_1}) \geq \#(V_{i_2})$, then

$$\#(V_{i_1}) \geq \frac{1}{2}\#(V_i) \qquad (V_i = V_{i_1} \cup V_{i_2}) .$$

Hence, by going along a path from the root 1 to a leaf, choosing always i_1 , the value $\#(V_i)$ decreases from 2^k to 1 and never decreases by more than half in any one step. Thus, the existence of i_0 is guaranteed. □

<u>Lemma 2.3.2</u>. If $G = (V,E)$ is a subgraph of the k-cube $\hat{G}$, and $2^{k-2} \leq |V| \leq 2^{k-1}$, then $|F_G^{ext}| > 2^{k-2}$.

Proof. We show by induction on k that for any subgraph $G = (V,E)$ of the k-cube, if $|V| = 2^{k-1} + r$ then $|F^{ext}| \geq 2^{k-1} - |r|$.
This will prove the lemma, since the given graph G satisfies $|V| \leq 2^{k-1}$, i.e., $r \leq 0$ and thus $|F_G^{ext}| \geq 2^{k-1} - |r| = |V| \geq 2^{k-2}$.
Basis is trivial.
Induction step: Let $|V| = 2^{k-1} + r$. Let $G_{(0)}$ be the projection of G onto $\lambda_k = 0$; i.e., $G_{(0)} = (V_{(0)}, E_{(0)})$ and

$$V_{(0)} = \{\lambda \mid \lambda \in V \ , \ \lambda_k = 0\}$$

$$E_{(0)} = \{(\lambda,\mu) \mid (\lambda,\mu) \in E \ , \ \lambda,\mu \in V_{(0)}\} \ .$$

Similarly, define $G_{(1)}$ to be the projection of G on $\lambda_k = 1$.
$|V_{(0)}| = 2^{k-2} + r_0$ and $|V_{(1)}| = 2^{k-2} + r_1$. By the induction hypothes $|F_{(0)}^{ext}| \geq 2^{k-2} - |r_0|$ and $|F_{(1)}^{ext}| \geq 2^{k-2} - |r_1|$. If r_0 and r_1 are of the same sign, then by adding the two inequalities we get $|F_{(1)}^{ext}| \geq |F_{(0)}^{ext}| + |F_{(1)}^{ext}| \geq 2^{k-1} - |r|$. The other cases are, w.l.o.g.

(1) $r > 0$, $r_0 > 0$, $r_1 = r - r_0 < 0$ and

(2) $r < 0$, $r_0 > 0$, $r_1 = r - r_0 < 0$.

In these cases there are at least $r_0 - (r - r_0) = 2r_0 - r$ edges in F^{ext} going from $V_{(0)}$ to $V_{(1)}$. Thus,

3. Some General Lemmas.

3.1. Introduction.

In this section we prove some general lemmas. The most important of them are

1. The Correspondence Lemma, which states the correspondence between operations on a graph G and a regular derivation using the clauses corresponding to G.
2. The Substitution Lemma, which proves the validity of substituting a given tree for a subtree in another given proof tree.
3. The Regularity Lemma, which states that if we can prove a given clause C by a resolution tree, then we can prove a clause C', $C' \subseteq C$, by a regular tree.

These three lemmas will be used in the sequel for proving most of the theorems. We still hope that by using these lemmas one will be able to prove an exponential lower bound for the general resolution procedure.

3.2. The Correspondence Lemma.

Let $G = (V,E)$ be a connected subgraph of $\overline{G}$. 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 $C = z_1^{\varepsilon_1} \vee \ldots \vee z_k^{\varepsilon_k}$, then $\delta(C) = \sum_{z_i \in F^{ext}} (1-\varepsilon_i)$. The <u>parity of G</u>, $\varepsilon(G)$, is defined to be the sum modulo 2 of the vertex labels of G. C is a <u>regular clause associated with G</u> 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 <u>regular clause</u> if there is a connected subgraph G of $\overline{G}$ such that C is a regular clause associated with G.

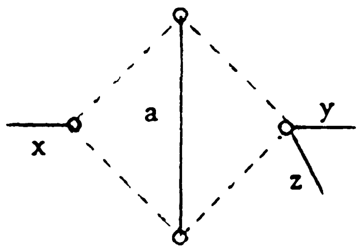


Figure 3.2.1.

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

<u>Example</u>. Consider the subgraph $G = (V,E)$ of $\overline{G}$ in Figure 3.2.1. V consists of the circles and E consists of the dotted lines. Assume that all vertex labels are 0, thus $\varepsilon(G) = 0$, $F^{int} = \{a\}$ and $F^{ext} = \{x,y,z\}$. 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 $\overline{G}$, such that C is a regular clause associated with G.

<u>Proof</u>. Assume C is a regular clause associated with G. By deleting all the edges of the boundary of G from $\overline{G}$ 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 $F^{int} = \emptyset$ and $G \cup G_1 = \overline{G}$ (G_1 is the other component), C is also associated with G_1. C cannot be a regular clause associated with G_1 since $\varepsilon(G) + \varepsilon(G_1) = \varepsilon(\overline{G}) = 1$ $\Rightarrow \varepsilon(G_1) \neq \varepsilon(G) \neq \delta(C) \Rightarrow \varepsilon(G_1) = \delta(C)$. □

The second observation is obvious:

ii) C is one of the original clauses which corresponds to a vertex v in $\overline{G}$ iff C is a regular clause associated with the subgraph $G = (v,\emptyset)$ consisting of an isolated vertex†.

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.

<u>Lemma 3.2.1. (The Correspondence Lemma)</u>. 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 $\overline{G}$ where

---

† We identify the vertex v and the singleton $\{v\}$.

$$V = \{v \mid v \in \overline{V} \text{ and a clause corresponding to } v \text{ appears on a leaf of } T\}$$

and

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

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

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

T:

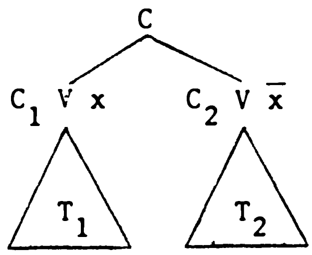


Figure 3.2.2.

Without loss of generality there are four cases:

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

---

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

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, $x \in F_1^{int} = F_2^{int}$. 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 C. Since $\delta(C_1 \vee x) \neq \varepsilon(G_1)$ and $\delta(C_2 \vee \bar{x}) \neq \varepsilon(G_2)$,

$$\delta(C_1 \vee x) + \delta(C_2 \vee \bar{x}) = \varepsilon(G_1) + \varepsilon(G_2) = \varepsilon(G) .$$

We show below that

$$\delta(C_1 \vee x) + \delta(C_2 \vee \bar{x}) \neq \delta(C) \qquad (1)$$

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

To prove (1), recall that if $C = z_1^{\varepsilon_1} \vee \dots \vee z_r^{\varepsilon_r}$, then $\delta(C) = \sum_{z_i \in F^{ext}} (1-\varepsilon_i)$. But $F_1^{ext} \cap F_2^{ext} \equiv x \cup F_{12}$ and $F_{12} \subseteq F^{int}$. (See Figure 3.2.3.) In $\delta(C_1 \vee x) + \delta(C_2 \vee \bar{x})$ 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 $\bar{x}$, $\delta(C_1 \vee x) + \delta(C_2 \vee \bar{x}) = \delta(C) + 0 + 1$ and (1) follows. $\square$

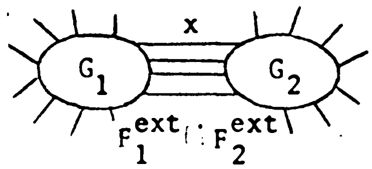


Figure 3.2.3.

Corollary 3.2.1. Let T be the regular proof tree of Figure 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 [5] Cook observed that the order of elimination of variables in DPP is important. A restatement of his result is

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 Figure 3.2.4. and let $S_n = \alpha(\hat{G}(\ell))$. Obviously all the clauses are of size 3.

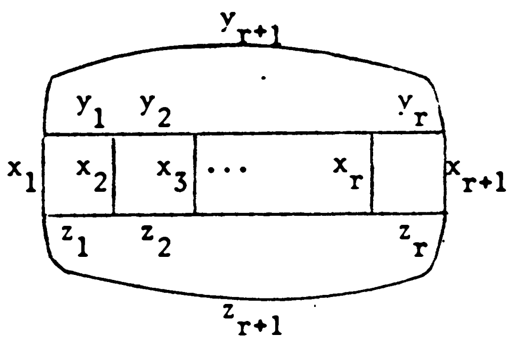


Figure 3.2.4. $\hat{G}(\ell)$, ℓ is a fixed labelling with $\varepsilon(\hat{G}(\ell)) = 1$

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 eliminating $x_2,\ldots,x_{r+1}$. One can easily verify that all 2^r clauses associated with G are generated at this point. Thus, a above follows since $n = 3r + 3$. On the other hand, Order 1 yields clauses associated with 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 iff 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.

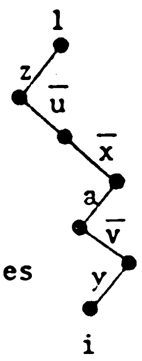
We use an additional labelling: If x is chosen at vertex i we label one of the edges $(i,i_1),(i,i_2)$ by x and one by $\bar{x}$. Given the labels from the root to i we take C_i to be the unique clause associated with G_i , such that the complements are assigned according to the edges in the path from 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 lables 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 one of the edges $(i,i_1),(i,i_2)$ with x and the other with $\bar{x}$, 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. $\square$

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

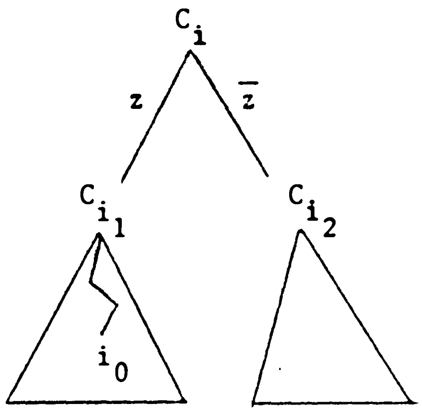
(u and v label edges which are not incident with vertices of G .)



3.4. Substitution.

We will need an operation called <u>substitution</u> which we now define. Let T be a proof tree and i_0 a vertex in the tree. Assume we are given another tree T'_{i_0} with $C'_{i_0} \subseteq C_{i_0}$ where C'_{i_0} is the clause at the root of T'_{i_0}. By the statement "T' is obtained from T by substituting T'_{i_0} for T_{i_0} in T." we mean that T' is constructed as follows: Start with T'_{i_0}, go along the path from i_0 to the root of $T - -1$, and perform all possible resolutions with subtrees which are incident with this path.

More precisely construct T' by the following inductive process: T'_{i_0} is the given tree.

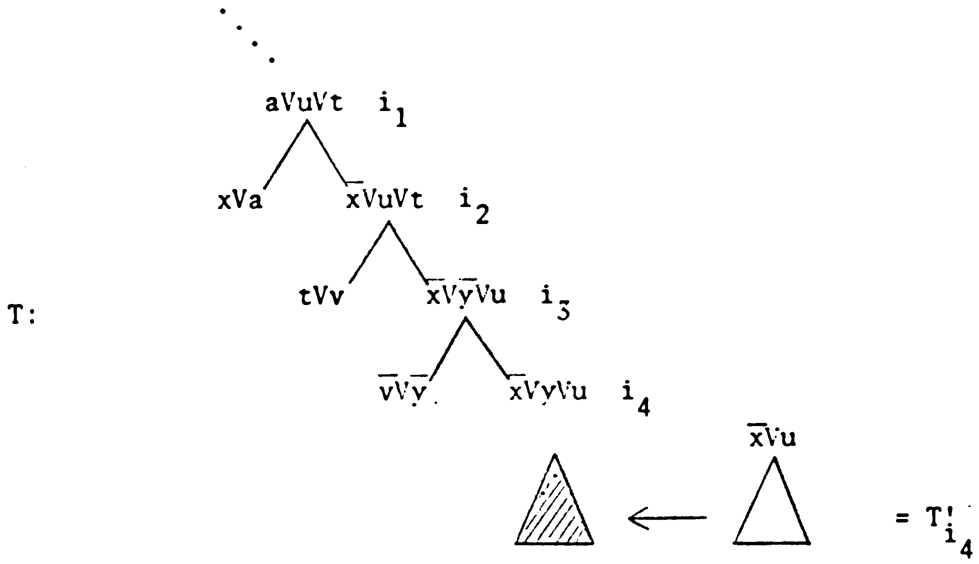


Assume i_1 is on the path, i is its father, i_2 is its brother and z is annihilated at i. Assume that T'_{i_1} has been defined. Then

$$T'_i = \begin{cases} \text{if } \; z \notin C'_{i_1} \; \text{ then } \; T'_{i_1} \\ \\ \text{else } \; \mathrm{Res}(T'_{i_1}, T_{i_2}) \end{cases}$$

Define $T' = T'_1$.

Example. The Figure below describes substituting T'_{i_4} for T_{i_4} in T .

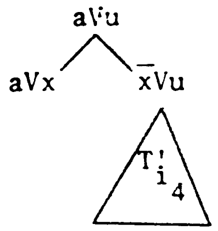


T'_{i_4} - given

$T'_{i_3} = T'_{i_4}$ $\qquad (y \notin C'_{i_4} = \bar{x}Vu)$

$T'_{i_2} = T'_{i_3} = T'_{i_4}$ $\qquad (\bar{v} \notin C'_{i_3} = \bar{x}Vu)$

$T'_{i_1} =$



.

Lemma 3.4.1. (The Substitution Lemma). Assume T is a proof tree, T'_{i_0} is a proof tree for $C'_{i_0} \subseteq C_{i_0}$ and T' is obtained from T by substituting T'_{i_0} for T_{i_0} in T. Then, T' is a proof tree.

Proof. For all i on the path from i_0 to the root, $C'_i \subseteq C_i$, where C'_i is the clause at the root of T'_i constructed above. This is proved by induction on the distance of i from i_0. The claim of the lemma follows from the case $i = 1$. □

From the proof above we have also

Corollary 3.4.1. If j is on the path from i_0 to the root of T', then there is a vertex s on the path from i_0 to the root of T such that $C'_j \subseteq C_s$.

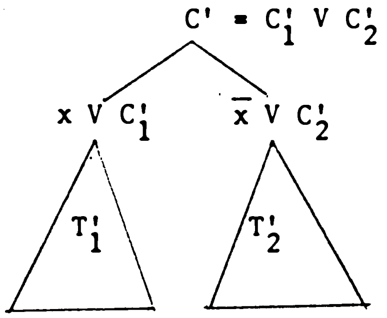
3.5. The Regularity Lemma.

Lemma 3.5.1. (The Regularity Lemma). Let T' be a proof tree for the clause C'. Then there is a $C \subseteq C'$ and a regular proof tree T for C, such that $\alpha \subseteq \alpha'$ where α (α') is the set of clauses on the leaves of T (T').

Proof. Induction on the height of T'.

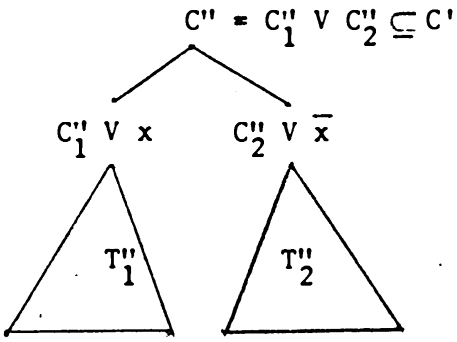
Basis - trivial

Induction step: Let T' be the tree



Apply the induction hypothesis to T_1' (T_2') to obtain a regular tree T_1'' (T_2'') which proves $\hat{C}_1$ $(\hat{C}_2)$ with $\hat{C}_1 \subseteq x \vee C_1'$ $(\hat{C}_2 \subseteq \bar{x} \vee C_2'')$.

If $x \notin \hat{C}_1$ $(\bar{x} \notin \hat{C}_2)$ then $T = T_1''$ $(T = T_2'')$ and $C = \hat{C}_1$ $(C = \hat{C}_2)$ satisfy the hypothesis of the lemma. Otherwise, let $\hat{C}_1 = C_1'' \vee x$ and $\hat{C}_2 = C_2'' \vee \bar{x}$ and let T'' be the tree



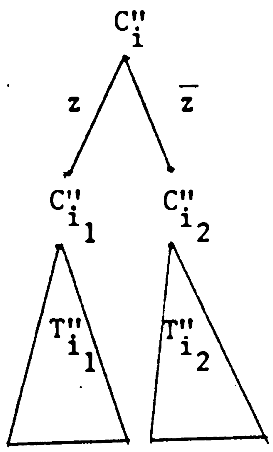
T'' needs not be regular because a literal which appears in $C_1'' \vee x$ $(C_2'' \vee \bar{x})$ might be annihilated in T_2'' (T_1''). So let

$$P = \{z \mid z \in C_1'' \text{ , } z \text{ is annihilated in } T_2''\}$$

$$\cup \{z \mid z \in C_2'' \text{ , } z \text{ is annihilated in } T_1''\} .$$

If $P = \emptyset$ then T'' is regular and $T = T''$ and $C = C''$ will do. Otherwise, we eliminate in T'' all annihilations of z's in P by deleting corresponding subtrees. T'' decomposes into a collection of separate subtrees and we construct from them a new tree T by performing all possible annihilations.

More precisely we construct T by the following inductive process: If T''_i consists of a single vertex, then $T_i = T''_i$. If T''_i is the tree



then

(1) If $z \in P$ $(\bar{z} \in P)$, then $T_i = T_{i_1}$ $(T_i = T_{i_2})$, i.e., the other subtree is deleted to prevent annihilation; otherwise

(2) If $z \notin C_{i_1}$ $(\bar{z} \notin C_{i_2})$, then $T_i = T_{i_1}$ $(T_i = T_{i_2})$, i.e., the other subtree is deleted since we do not need to annihilate z in the new tree; otherwise

(3) $T_i \leftarrow \mathrm{Res}(T_{i_1}, T_{i_2})$.

Let $T = T_1$ (where 1 is the root) and $C = C_1$. For i in T, an easy induction on the height of T_i proves that

(a) T_i is regular

(b) $C_i \subseteq C''_i \vee P$. (Recall that a clause is a set of literals.)

Thus T is regular and $C \subseteq C'' \subseteq C'$, since $P \subseteq C''$. Hence T and C satisfy the required properites. In all cases $\alpha \subseteq \alpha'$ since T is obtained from T' by using only subtrees of T' . This completes the proof. $\square$

Example. Figure 3.5.1. illustrates the induction step in the proof of Lemma 3.5. Although the two main subtrees of T' are regular T' itself is not regular $(P \neq \emptyset)$. The circled subtrees are eliminated to prevent annihilation of literals in P ((1) above). The third subtree is eliminated because y does not appear in the corresponding clause ((2) above). As expected (a) and (b) above hold. Note that $\bar{x} \vee \bar{u} \vee z$ is longer than its corresponding clause in T' $(\bar{x} \vee \bar{u})$.

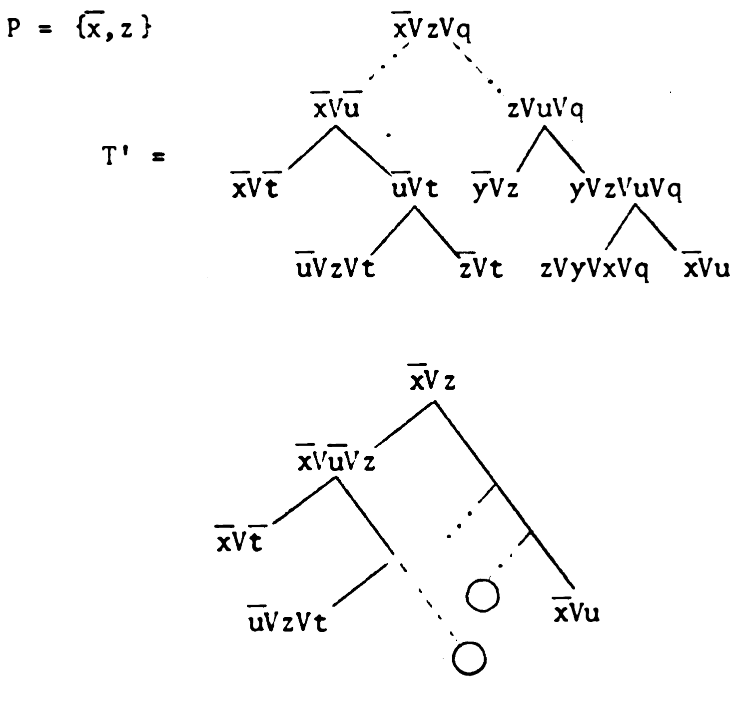


Figure 3.5.1.

4. On the Validity and Complexity of Bounded Resolution.

4.1. Introduction.

In this section we consider certain proposed procedures based on (not necessarily regular) resolution for checking whether a formula in 3-CNF is unsatisfiable. We get the following results:

1) The proposed procedures which are forced to run in polynomial time do not identify all unsatisfiable formulas.

2) The proposed procedure which identify all unsatisfiable formulas require exponential time in the worst case.

The exponential lower bounds for these procedures do not follow from Tseitin's lower bound for regular resolution since these procedures also allow nonregular resolution trees.

We consider three versions of bounded resolution (BR): we assume that there is an integer k such that the size of each clause generated is bounded by k. All three versions use an operation RESOLVE(k) which adds to the current set of clauses α all those clauses of size $\leq k$ obtained from two of the clauses in α by the resolution rule. Two of the versions use the operation EXPAND(k) which replaces each clause C of size less than k by all terms $\hat{C}$ of size k such that $C \subseteq \hat{C}$. Let ALL(k) be a predicate which is true iff all terms of size k are present. The three routines are:†

---

† The numbers 1,2,3 denote positions in the programs. We will refer to them in the sequel.

```
BR1(k):  repeat
            EXPAND(k)
            RESOLVE (k)
         until no new clauses are generated
     1
         if ALL(k) then output "unsatisfiable"
                   else output "satisfiable"

BR2(k):  repeat
            RESOLVE(k)
         until no new clauses are generated
     2
         EXPAND(k)
     3
         if ALL(k) then output "unsatisfiable"
                   else output "satisfiable"

BR3(k):  repeat
            RESOLVE(k)
         until no new clauses are generated
         if the empty clause is present then output "unsatisfiable"
                                        else output "satisfiable"
```

BR1 has been suggested by J. Hopcroft. It actually works only with clauses of size exactly k . BR3 is similar to the (unrestricted) resolution procedure. It (implicitly) generates resolution trees with clauses of size at most k .

These procedures generate $O(m^k)$ distinct clauses, where m is the number of variables. Hence, they all run in polynomial time. For each of these procedures we would like to know:

(1) Does the procedure work correctly for some constant integer k ? and if not,

(2) Does it work correctly for $k = k(n)$, some slowly growing function of n , where n is the size of the input? (Obviously $k = m$ will do.)

A positive answer to (1) would imply $P = NP$. A positive answer to (2) would yield a subexponential algorithm for the tautology problem (and hence to all NP-complete problems) provided $k(n)$ is asymptotically slower than n^{ε} for every ε . That question (1) has a negative answer seems obvious, but the simplest proof we could obtain for this result is distinctly nontrivial!

If k is not large enough, these procedures might give a wrong answer -- they might decide that the input is satisfiable while it actually is unsatisfiable. The possibility of such an error can be detected. Obviously if upon termination of BR1 (BR2, BR3) all terms of size k are present (the empty clause is present) the input is unsatisfiable. Otherwise, if a new term can be generated (by the resolution rule) which is larger than k and does not contain any of the present clauses, one cannot conclude that the input is satisfiable, but rather that the bound k is too small. This leads to the <u>iterated bounded resolution</u> (IBR): IBR1 (IBR2, IBR3) is simply iterating BR1(k) (BR2(k), BR3(k)) with increasing k if the case mentioned above occurs. These iterated bounded resolution procedures always give the correct answer. The question about them is:

(3) What is the time bound for the iterated bounded resolution procedures?

Note that we restricted the inputs to formulas in 3-CNF; otherwise question (1) is meaningless.

Example. Recall $\beta_{n,k}$ defined in Subsection 2.1. and consider

$$\beta_{5,3} = \{(x_1 \vee x_2 \vee x_3), (x_1 \vee x_2 \vee x_4), \ldots, (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3), \ldots, (\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5)\} .$$

One can see immediately that BR(3) would not work and that BR(4) will work (where BR is any one of the versions). (This example is due to S. Even.)

Our attempts to answer questions (1) - (3) by first considering simple examples have failed. The seemingly simple question, "Is the bound $k = 4$ sufficient for $m = 6$?" (where m is the number of variables) was answered by J. Simon [29]. However, the proof is complicated and it is not clear how to generalize it. Obviously, one can use the idea in the example above and consider $\beta_{m,\ell}$, $\ell \leq \lceil m/2 \rceil$. Obviously a bound $k \geq 2\ell-2$ is necessary for BR(k) to work for $\beta_{m,\ell}$. However, after using the usual trick of converting $\beta_{m,\ell}$ to 3-CNF it is no longer clear that a smaller bound would not suffice. Even if this method worked it would have settled question (1), but not questions (2) and (3), since the input is of length $n = O(\binom{m}{\ell})$ and the number of distinct clauses generated is $< 2^{2\ell}\binom{m}{2\ell-2} < n^3$ since one can show that the bound $k = 2\ell-2$ suffices.

In Theorem 4.1.1. we prove the equivalence of BR1, BR2 and BR3. This will enable us to speak about BR which will stand for each one of them.

<u>Theorem 4.1.1.</u> For a fixed k all proposed bounded resolution versions are equivalent, i.e., they give the same answer to any given input.

<u>Proof.</u> First we show that BR1 and BR2 are equivalent. Let $term_1$, $term_2$ and $term_3$ be the clauses present at points 1, 2 and 3 in BR1 and BR2. We show that (1) for every $\hat{C}$ in $term_1$ there is $C \in term_2$ such that $C \subseteq \hat{C}$; and (2) for every C in $term_2$, every $\hat{C}$ of size k such that $C \subseteq \hat{C}$ is in $term_1$. This will imply (by definition of EXPAND(k)) that $term_1 = term_3$ and will complete the proof.

(1) Proof by induction on the level of resolution in BR1. For level 0 this follows from the definition of EXPAND(k). Assume $\hat{C} \in term_1$. There must be $\hat{\hat{C}} \subseteq \hat{C}$ which was obtained from $\hat{C}_1$ and $\hat{C}_2$:

$$\begin{tikzcd} & \hat{C} \arrow[dl, dash, "x"'] \arrow[dr, dash, "\bar{x}"] & \\ \hat{C}_1 & & \hat{C}_2 \end{tikzcd}$$

By induction hypothesis there are $C_1, C_2 \in term_2$, $C_1 \subseteq \hat{C}_1$ and $C_2 \subseteq \hat{C}_2$. If $x \notin C_1$ ($\bar{x} \notin C_2$) then $C_1 \in term_2$ ($C_2 \in term_2$) and $C_1 \subseteq \hat{C}$ ($C_2 \subseteq \hat{C}$) . Otherwise, BR2 generates C , the resolvent of C_1 and C_2 . $C \in term_2$ and $C \subseteq \hat{C}$.

(2) Proof by induction on the level of resolution in BR2. For level 0 it follows again by definition of EXPAND(k). Assume $C \in term_2$ and

is obtained from C_1 and C_2 :

$$\begin{tikzcd} & C \arrow[dl, dash, "x"'] \arrow[dr, dash, "\bar{x}"] & \\ C_1 & & C_2 \end{tikzcd}$$

Let $\hat{C} \supseteq C$ be of size k . We show that $\hat{C} \in \text{term}_1$. Note that $|C_1 \cap C_2| < k$ $(x \in C_1, \bar{x} \in C_2)$. There are two possibilities:

(a) $|C_1 \cap C_2| = k - 1$. Thus $|C_1| = |C_2| = k$ and by the induction hypothesis they belong to term_1. Thus C will be generated in BR1 by RESOLVE(k) and $\hat{C}$ will be generated by the immediately following EXPAND(k).

(b) $|C_1 \cap C_2| \leq k - 2$. Thus $(k-|C_1|) + (k-|C_2|) = 2k - |C_1| - |C_2|$ $= 2k - |C_1| - |C_1 \cap C_2| - |C_2 - C_1| \geq k + 2 - |C_1| - |C_2 - C_1| = k + 2$ $- |C_1 \cup C_2| = k - |C|$. Following the inequality proved above, since $|\hat{C} - C| = k - |C|$ we can write $\hat{C} - C = D_1 \cup D_2$ such that $|D_i| \leq$ $k - |C_i|$ for $i = 1,2$. Thus $\hat{\hat{C}}_i = C_i \cup D_i$ satisfies $|\hat{\hat{C}}_i| \leq k$. Let $\hat{C}_i$ be a clause of size k obtained from $\hat{\hat{C}}_i$ by adding to it enough literals from $\hat{C} - \hat{\hat{C}}_i$. By induction hypothesis, $\hat{C}_1$ and $\hat{C}_2$ belong to term_1. So C , which is obtained from them by the resolution rule, belongs to term_1.

Now we show that BR2 and BR3 are equivalent. We prove that $\Lambda \in \text{term}_2$ iff ALL(k) is true for term_3. Since up to point 2, BR2 and BR3 are the same, this completes the proof.

If $\Lambda \in \text{term}_2$, then ALL(k) is true for term_3 by definition of EXPAND(k). To prove the converse we need the following definition:

A set of terms satisfies ALL'(ℓ) if, for every clause $\hat{C}$ of size ℓ, there is a term C in the set such that $C \subseteq \hat{C}$. If ALL(k) is true for term$_3$, then ALL'(k) is true for term$_2$. We show below that if ALL'(i), $i > 0$, is true for a set of clauses, then ALL'(i-1) is true for the set obtained by applying RESOLVE(k). But RESOLVE(k) does not change term$_2$. Thus, ALL'(0) must be true for term$_2$, i.e., $\Lambda \in$ term$_2$. Let α be a set of clauses which satisfies ALL'(i), $i > 0$, and let β be the set obtained by applying RESOLVE(k) to α. Given an arbitrary clause $\hat{C}$ of size $\ell - 1$, we show that there is a clause $C \in \beta$ with $C \subseteq \hat{C}$. Let x be a variable such that x and $\bar{x}$ do not belong to $\hat{C}$. Since ALL'(i) holds for α, there are C_1 and C_2 in α, $C_1 \subseteq \hat{C} \vee x$ and $C_2 \subseteq \hat{C} \vee \bar{x}$. If $x \notin C_1$ $(\bar{x} \notin C_2)$ then $C = C_1$ $(C = C_2)$ will do. Otherwise, let C be the resolvent of C_1 and C_2. Obviously, $C \in \beta$ and $C \subseteq \hat{C}$. □

4.2. Answers to the Questions.

Theorem 4.2.1. For infinitely many $n \geq 1$, there are unsatisfiable formulas α_n in 3-CNF over n variables, $|\alpha_n| = 8n$, such that every resolution tree using α_n contains clauses of size $\geq n/6(\log n)^2$.

Proof. Let α_n be the set of clauses which corresponds to the graph $\bar{G}$ of Section 2. We show that every resolution tree using α_n contains clauses of size $2^{k-2}/k$. The claim above follows since $n = 1.5k2^k$.

Suppose, to the contrary, that there is a resolution tree $\hat{T}$ using α_n with $\max_{i \in \hat{T}} |C_i| = \ell < 2^{k-2}/k$. For every proof tree T

using α_n we will assign a positive integer $\mu(T)$ as described below. Starting with $T^{(0)} = \hat{T}$ we show how to transform $T^{(m)}$ into a new proof tree $T^{(m+1)}$ using α_n, such that $\mu(T^{(m+1)}) < \mu(T^{(m)})$. This will establish the contradiction.

Definition of $\mu(T)$. Recall V_i and $\#(V_i)$ as defined in Section 2. We say that a vertex i in T satisfies property P (or $P(i)$ is true) iff $2^{k-2} < \#(V_i) < 2^{k-1}$ and for all $j \neq i$ in T_i, $\#(\overline{V}_j) < 2^{k-2}$.

Let ℓ_i be the distance of i from the root of T and let

$$\mu(T) = \sum_{P(i)=\text{true}} \ell_i .$$

By Lemma 2.3.1. there is an i_0 such that $P(i_0)$ is true. Since $P(1)$ is false, $\mu(T) > 0$.

The transformation from $T \equiv T^{(m)}$ to $T' \equiv T^{(m+1)}$

Induction hypothesis. Assume all vertices i with $P(i)$ true and their ancestors in T contain clauses of size $\leq \ell$. (This is obviously true for $T^{(0)}$.)

Let i be a vertex in T with $P(i)$ true. By the assumption above, $|C_i| \leq \ell$. By the Regularity Lemma, C_i contains a clause C_i' which has a regular proof tree T_i', such that $V_i' \subseteq V_i$ (where V_i' has the obvious meaning). We will prove below:

Proposition. For every vertex j in T_i', $P(j)$ is false.

T' is obtained from T by substituting T_i' for T_i in T. Consider all subtrees attached to the path from i to the root of T. They contain all j's such that $P(j)$ = true except for i. During the substitution some of them may be deleted. Those which are not can only get closer to the root. They satisfy P because their subtrees are not changed. By the proposition, for every j in T_i', $P(j)$ is false. There can be at most one vertex j with $P(j)$ false in T and $P(j)$ true in T'. If such a vertex exists it must be a proper ancestor of i. Thus, $\mu(T^{(m+1)}) < \mu(T^{(m)})$.

We need to show that the induction hypothesis holds for T'. Consider a vertex j which is an ancestor of a vertex which satisfies P. If j is not on the path from i to the root of T', then C_j' also appeared in T on an ancestor of a vertex which satisfied P. (Its whole subtree is also a subtree of T.) Thus, $|C_j'| \leq \ell$ by the induction hypothesis. Otherwise, by Corollary 3.4.1., $C_j' \subseteq C_s$ for some s, an ancestor of i. Thus, $|C_j'| \leq |C_s| \leq \ell$ and the induction hypothesis holds for T'. □

<u>Proof of the Proposition</u>. $V_i' \subseteq V_i$ implies that $\#(V_i') < 2^{k-1}$. By the Correspondence Lemma, C_i' is a regular clause associated with some $G_i' = (V_i', E_i')$. Also, $|F_i^{ext,}| \leq \ell$ since

$$|F_i^{ext,}| \leq |F_i'| = |C_i'| \leq |C_i| \leq \ell .$$

This implies that $\#(V_i') < 2^{k-2}$, since otherwise $2^{k-2} \leq \#(V_i') < 2^{k-1}$, and so

$$|F_i^{ext,}| \geq 2^{k-2}/k > \ell$$

by Lemma 2.3.3. -- a contradiction.

Let j be a vertex in T_i'. $\#(V_j') \leq \#(V_i') < 2^{k-2}$ since $V_j' \subseteq V_i'$. Thus, j does not satisfy P. □

Theorem 4.2.1. answers questions (1) and (2) about the bounded resolution procedures:

Corollary 4.2.1. BR(k) works for α_n only if $k \geq n/6(\log n)^2$.

It also answers question (3) about the iterated bounded resolution procedures:

Corollary 4.2.2. The run-times of IBR1, IBR2 and IBR3 are all greater than $2^{n/6(\log n)^2}$.

Proof. Corollary 4.2.1. for BR3 follows from Theorem 4.2.1. But by Theorem 4.1.1., the same holds for BR1 and BR2. Corollary 4.2.2. is obvious for IBR1 and IBR2 since $2^{n/6(\log n)^2}$ is a lower bound for the number of distinct clauses of size $n/6(\log n)^2$. Also, it is easy to see that if IBR3 generates a regular clause associated with some subgraph G, it must generate all regular clauses associated with G. Since a G with $|F_G^{ext}| > n/6(\log n)^2$ must exist, at least $2^{n/6(\log n)^2}$ distinct clauses are generated by IBR3. □

Note that a theorem corresponding to Theorem 4.2.1. with the same lower bound on the size of clauses in regular trees using α_n is trivial. It follows immediately from the Correspondence Lemma and Lemma 2.3.1. Note also that by applying the Regularity Lemma to a given proof tree we obtain a regular proof tree. But the latter can contain longer clauses (as in the example in Subsection 3.5) and thus Theorem 4.2.1. does not follow directly from the Regularity Lemma.

5. Exponential Lower Bound for Regular Resolution.

5.1. Introduction.

In this section we give a simplified version of Tseitin's proof which establishes an exponential lower bound for regular resolution. As was observed in [6], this implies an exponential lower bound for the Davis Putnam Procedure (DPP). However, an exponential lower bound for DPP may be obtained directly by a much simpler argument. We will sketch this argument at the end of this section.

We use again a contradictory system of clauses which corresponds to a graph $\overline{G}$. 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, $\sum w(C) \leq 1$, where the sum is taken over all occurrences of C in T. This will imply $\sum_{i \in T} w(C_i) \leq N(T) = \hat{N}$. 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 5.2.1. which is the hardest part of the proof.

First we need to make several observations. Let i be a vertex on T and let $G_i = (V_i, E_i)$ be the connected subgraph of $\overline{G}$ 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:

<u>case 1</u> 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.

5.2. 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 5.2.1. Assume $C = C_{\ell_1} = C_{\ell_2}$, 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 $G = G_{\ell_1} = G_{\ell_2}$.

We denote by $\{D_1,\ldots,D_{\hat{N}}\}$ the distinct clauses in T and let $I_j = \{i | C_i = D_j\}$ for $1 \le j \le \hat{N}$. For r an ancestor of ℓ in T let $m_{\ell,r}$ 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. $\sum_{\ell \in I_j \cap T_r} 2^{-m_{\ell,r}} \le 1$ for $1 \le j \le \hat{N}$.

Proof. An easy induction on the height of T_r. The induction step follows from Lemma 5.2.1., since whenever D_j appears in both subtrees, $m_{\ell,r}$ grows by one. □

Thus,

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

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 5.2.2. $n_\ell = m_{\ell,1} + s_\ell$.

Hence by (1),

$$\sum_{\ell \in I_j} 2^{-n_\ell} \cdot 2^{s_\ell} \leq 1 \quad \text{for all} \quad 1 \leq j \leq \hat{N} . \tag{2}$$

Summing up (2) for all j yields

$$\hat{N} \geq \sum_{\ell} 2^{-n_\ell} \cdot 2^{s_\ell} \equiv Y(T) . \tag{3}$$

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

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

<u>Claim.</u> $$Y(T) = Y(T_1)\ . \tag{4}$$

<u>Proof.</u> 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_ℓ} is 2^{-n_ℓ} 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

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

<u>Claim.</u> $$Y(T_1) \geq Z(\overline{G}). \tag{5}$$

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

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

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

But,

$$Z(\overline{G}) = \sum_i 2^{s^{(i)}} \tag{7}$$

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

We now specify $\overline{G}$ as in Section 2. By Lemma 2.3.1. and Lemma 2.3.3 there is i such that $|F^{ext}_{(i)}| \geq n/6(\log n)^2$. Also, by Lemma 2.3.4.,

$$s^{(i)} \geq 2/3|F^{ext}_{(i)}| \geq n/9(\log n)^2 . \tag{8}$$

By (6), (7) and (8), $\hat{N} \geq 2^{n/9(\log n)^2}$. Since $|\alpha(\overline{G})| = 8n$ we have just proved:

<u>Theorem 5.2.1.</u> $\mathrm{Comp}_{Reg}(n) \geq 2^{n/72(\log n)^2}$.

<u>Corollary 5.2.1.</u> DPP with subsumption is exponential.

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

<u>Lemma 5.2.3.</u> 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. Thus, for sets of clauses associated with graphs, regular resolution is no more powerful than the Davis Putnam Procedure. Nevertheless, the direct proof for Corollary 5.2.1. is much simpler than the proof of Theorem 5.2.1. We give an outline of this proof (exponential lower bound for DPP) at the end of this section.

5.3. Proofs of the Lemmas.

Proof of Lemma 5.2.1. 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 $x_i \in C_{\ell_1}$ and $\overline{x}_i \in C_{\ell_2}$ -- a contradiction $(C_{\ell_1} = C_{\ell_2})$.

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 :

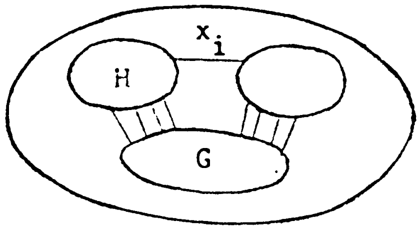


Figure 5.3.1. G_i

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 $\delta_r = \sum_{z_m \in F^{ext}_{H_r}, z_m \in C_r} \varepsilon_m (1-\varepsilon_m)$, i.e., δ_r is the contribution to the parity of C_r by edges in the exterior boundary of H_r .

Claim. $$\delta_{i_1} = \delta_{\ell_1} + \varepsilon(H) . \qquad (9)$$

Similarly, $\delta_{i_2} = \delta_{\ell_2} + \varepsilon(H)$. But $\delta_{\ell_1} = \delta_{\ell_2}$ and $\delta_{i_1} \neq \delta_{i_2}$. $(C_{i_1} = C_i \vee x_i$, $C_{i_2} = C_i \vee \overline{x}_i$ 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

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

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 $G_{r_2} \subseteq H_r$.

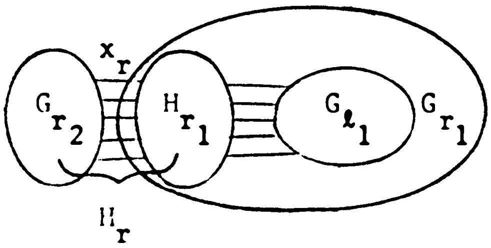


Figure 5.3.2. G_r

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

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

But,

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

This follows from

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

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

c) $\varepsilon'_r \neq \varepsilon_r$ and $\varepsilon'_j = \varepsilon_j$ for all $j \neq r$.

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

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

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

Proof of Lemma 5.2.2. Let $G = G_\ell$ and let $\{H_j\}_{j=1}^m$, $\{k_j\}_{j=1}^m$, s_ℓ , F_ℓ^{int} as defined in Section 2. Recall that all of these were defined by considering G as a subgraph of $\overline{G}$. In this proof only we will 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., $F_{\ell,i}^{int}$ is the interior boundary of G_ℓ w.r.t. G_i , etc.

By definition of $s_{\ell,i}$

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

<u>Claim.</u> $s_{\ell,i}$ is the number of variables x_j from i to ℓ that break a cycle in G_j, which is not an exterior cycle with respect to G_ℓ.

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

The claim is easily proved by induction on the distance of i from ℓ. For $i = \ell$, $0 = 0$. 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 $|F^{int}_{\ell,i}| = |F^{int}_{\ell,i_1}| + 1$

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 5.3.1. (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. $\square$

Proof of Lemma 5.2.3. It is implicit in the proof of Lemma 5.2.1. 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 $G_i - G_\ell$ 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

$$2^{|F_\ell^{int}|+\sum_j (k_j-1)} = 2^{s_\ell}$$

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.) $\square$

5.4. A Direct Proof for Corollary 5.2.1.

We now outline a direct proof of an exponential lower bound for the Davis Putnam procedure. It goes as follows:

1. Proof of the Correspondence Lemma.
2. Showing the existence of G_i, such that $s_i \geq n/9(\log n)^2$.
3. Showing that at least 2^{s_i} distinct regular clauses associated with G_i must be present.

1 and 2 are the same as for regular resolution. For 3 we need only a few observations:

a) All clauses generated by DPP are regular;

b) Without subsumption, if a clause C is generated and it is associated with the subgraph G, then all regular clauses associated with G are generated at the same stage;

c) To count how many clauses which correspond to G_i are generated, we may assume w.l.o.g. that we use DPP without subsumption and count all the clauses C which correspond to G_i and are not subsumed by any other clause at the time they are all generated. (b and c are proved by a straightforward induction.)

d) If C corresponds to G_i and is subsumed by C' when it is generated, then C' is a regular clause associated with a connected component of $G - G_i$.

<u>Proof</u>. By a) C' corresponds to some G_i'. If G_i' is not a connected component of $G - G_i$, since $C' \subsetneq C$, G_i' must be G_i plus some edges in F_i^{int}. But such edges cannot exist since $x \in E_i' \Rightarrow x$ had been eliminated before C' was generated, and $x \in F_i^{int} \Rightarrow x$ has not been eliminated when C is generated.

e) If $\{H_j\}_{j=1}^m$ are the connected components of $\overline{G} - G_i$ and k_j is the number of edges connecting G_i and H_j, then

$$s_i = |F_i^{int}| + \sum_{j=1}^{m} (k_j - 1) .$$

f) If C is a clause associated with G_i such that for each j the unique clause $C_j \subseteq C$ which is associated with H_j satisfies $\delta(C_j) = \varepsilon(H_j)$, then C is a regular clause associated with G_i and is not subsumed by any clause when it is generated. (By d , C can be subsumed only by some C_j . But by a none of the C_j is generated. C is regular by a simple parity argument.)

g) The number of such C's is

$$2^{|F_i^{int}| + \sum_{j=1}^{m} (k_j - 1)} = 2^{s_i} . \qquad \square$$

The proof of an exponential lower bound for the DPP is much simpler than the proof of the same result for regular resolution, since the latter is much stronger. This is because 1) DPP is a special case of regular resolution, and 2) in the case of regular resolution we count only distinct clauses which appear on the proof tree. It is not at all clear from the direct proof outlined above how many of the distinct clauses generated by the DPP must appear in a DP tree.

6. Extended Resolution.

6.1. Introduction.

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 $z \equiv f(x,y)$. One can verify that $\alpha(f,z,x,y)$ exists for all 16 possible choices of f .

Example.

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

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

$$\alpha(U_1,z,x,y) = \{(\bar{z}\vee x),(\bar{x}\vee z)\} \quad \text{where} \quad U_1(x,y) = x .$$

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 extensioı rule is similar to the use of additional variables in formal algebraic manipulation to denote repeating subexpressions. (This helps sometimes to reduce the lengths of the arithmetic computations by using clever substitutions.)

In the preceding sections we considered several restricted versioı of resolution. We saw that

1) Allowing only clauses of size up to a fixed constant bound causes the procedure not to recognize certain unsatisfiable sets of clauses.

2) Allowing only regular resolution enabled us to prove an exponential lower bound.

The question which naturally arises is: what happens if we impose these restrictions on extended resolution? The following theorems answer this question.

Let T be a resolution proof tree (with or without extension). Let m and $N = N(T)$ be the number of variables and distinct clauses on T respectively.

<u>Theorem 6.1.1.</u> Using extension, we can transform T into a regular proof tree T' which contains at most $O(N^2m)$ distinct clauses.

If the original set of clauses has at most ℓ literals per clause, then

<u>Theorem 6.1.2.</u> Using extension, we can transform T into a proof tree T' with clauses of size $\leq \ell$. T contains at most $O(Nm)$ distinct clauses.

<u>6.2. Proofs of the Theorems.</u>

<u>Proof of Theorem 6.1.1.</u> In order to transform T into a regular tree we can eliminate all irregularities using <u>renaming</u>, i.e., we prevent reintroducing x by substituting z for it. (See Figure 6.2.1.) We do the same whenever x is reintroduced between 1 and 2 (e.g., at 3). Later, at the point where x is annihilated again (at 2) we substitute $\overline{z}$ for $\overline{x}$ and annihilate z .

At first glance it is not at all clear that

1) By using renaming we can eliminate all irregularities.

2) That in case renaming succeeds it does not introduce too many new clauses. (The number of new clauses might be $O(N)$; i.e., one renaming

might double the number of distinct clauses.) We will use renaming, but in a careful manner.

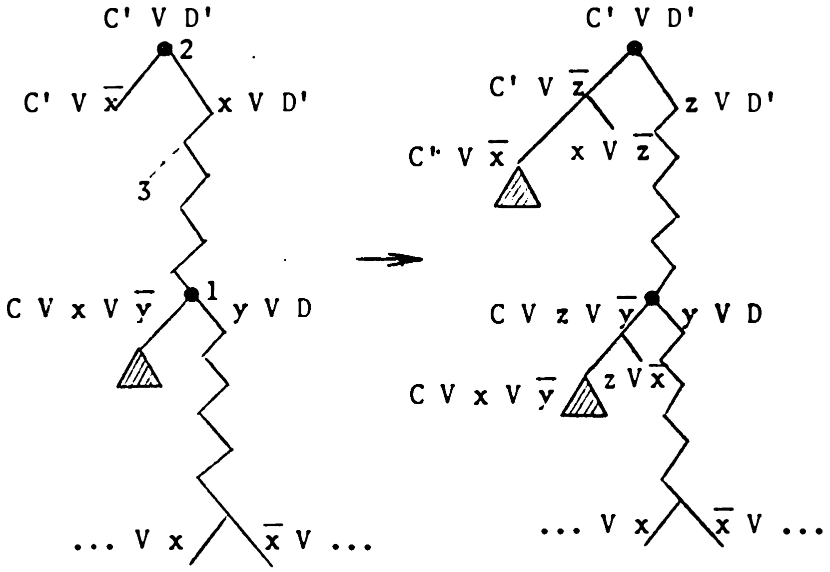


Figure 6.2.1.

W.l.o.g. assume that each distinct clause in T is derived by the same subtree in each of its occurrences in T. This implies that the same clause cannot appear twice on any path from the root to one of the leaves. Define the following partial order on the clauses on T : $C < D$ iff D appears at a root of a subtree which contains C. Our assumption on T implies that $<$ is indeed a partial order. Thus, we can number the clauses $C_1, C_2, \ldots, C_N$, so that $C_i < C_j \Rightarrow i < j$ (topological sort).

We use Nm variables: for each original variable x we use $x^{(1)}, \ldots, x^{(N)}$ where $x^{(1)}$ is identical to x. $x^{(i)}$ is said to be the i-th level of x. A clause is at level i if all its literals are at level i . The new clauses formalize $x^{(i+1)} \equiv x^{(i)}$ for $1 \leq i \leq N-1$. If $C = x^{(i)} \vee D$ we *rename $x^{(i)}$ in C* by

$$
\begin{tikzcd}[column sep=small]
 & x^{(i+1)} \vee D \arrow[dl, dash] \arrow[dr, dash] & \\
x^{(i)} \vee D & & \bar{x}^{(i)} \vee x^{(i+1)}
\end{tikzcd}
$$

Similarly, we can rename $\bar{x}^{(i)}$ in C . We *rename C at level i to level $i+1$* if

$$C = {x_{i_1}^{(i)}}^{\varepsilon_1} \vee \ldots \vee {x_{i_k}^{(i)}}^{\varepsilon_k} \quad , \quad 1 \leq i_1 < i_2 < \ldots < i_k \leq m$$

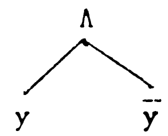
and we rename ${x_{i_1}^{(i)}}^{\varepsilon_1}$, then ${x_{i_2}^{(i)}}^{\varepsilon_2}$ and so forth. We obtain

$$C' = {x_{i_1}^{(i+1)}}^{\varepsilon_1} \ldots {x_{i_k}^{(i+1)}}^{\varepsilon_k}$$

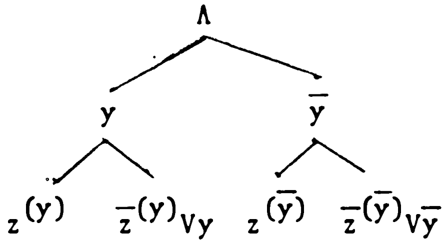
at level $i+1$. Note that C is renamed in a fixed order. We *rename α from level i to level $j > i$* by renaming it to level $i+1, i+2, \ldots, j$.

All the original clauses are at level 1. We use the following inductive construction: if C_i is on a leaf of T , i.e., it is an original clause, then we rename C_i from level 1 to level i. If C_i is derived from C_j and C_k in T , then $j < i$ and $k < i$ and we rename C_j and C_k from levels j and k respectively to level i. Then, we resolve them to get C_i at level i. The new tree T' is regular and contains $\leq O(N^2m)$ distinct clauses. $\square$

Proof of Theorem 6.1.2. W.l.o.g. $\ell \geq 3$. For every clause C in T we have a new variable $z^{(C)}$ such that $z^{(C)} \equiv C$. Using additional variables, we show that if E is derived from C and D in T, then given trees for $z^{(C)}$ and $z^{(D)}$, we can derive $z^{(E)}$. We also indicate how to derive $z^{(C)}$ for every input clause C . If the top of T looks like



then it follows by induction from the claim above that we can derive $z^{(y)}$ and $z^{(\bar{y})}$. But $z^{(y)} \equiv y$ and $z^{\bar{y}} \equiv \bar{y}$, and using the available clauses expressing these facts we get the top of T' :



Since we will use only clauses of size $\leq \ell$ this will complete the construction of T' .

Let $x_1, x_2, \ldots, x_m$ be the variables used in T . Assume $C =$ $y_{i_1} \vee \ldots \vee y_{i_r}$ where $1 \leq i_1 \leq \ldots \leq i_r \leq m$. Let $C_j \equiv y_{i_1} \vee \ldots \vee$ y_{i_j} $(C_r = C)$. Thus,

$$z^{(C_1)} \equiv y_{i_1} \tag{(1)a}$$

and

$$z^{(C_j)} \equiv z^{(C_{j-1})} \vee y_{i_j} \ , \ 1 \le j \le r \ . \qquad (1)b$$

(For the corresponding set of clauses use the example in the beginning of the section.)

We now show how to obtain $z^{(E)}$ from $z^{(C)}$ and $z^{(D)}$. Let $C = u_1 \vee \ldots \vee u_r$, $D = v_1 \vee \ldots \vee v_s$, x is u_p, $\bar{x}$ is v_q. We denote by $C \vee z \to C \vee z'$ the subtree

$$\begin{array}{c} C \vee z' \\ \diagup \quad \diagdown \\ \bar{z} \vee z' \qquad C \vee z \end{array}$$

and let $\overset{*}{\to}$ be the transitive closure of $\to$. Note that if $w \in E$, then $w \vee C' \overset{*}{\to} z^{(E)} \vee C'$: If w is the j-th literal in E then,

$$x \vee C' \to z^{(E_j)} \vee C' \to z^{(E_{j+1})} \vee C' \to \ldots \to z^{(E)} \vee C' \ .$$

(The clauses $\bar{w} \vee z^{(E_j)}, \bar{z}^{(E_j)} \vee z^{(E_{j+1})}, \ldots$ are available from (1).)

We decompose $z^{(C)}$ (i.e., $z^{(C_r)}$) as follows:

$$\begin{array}{l} z^{(C_r)} \qquad \bar{z}^{(C_r)} \vee u_r \vee z^{(C_{r-1})} \\ \quad \diagdown \quad \diagup \\ \quad z^{(C_{r-1})} \vee u_r \overset{*}{\to} z^{(C_{r-1})} \vee z^{(E)} \qquad \bar{z}^{(C_{r-1})} \vee u_{r-1} \vee z^{(C_{r-2})} \\ \qquad\qquad\qquad\qquad\qquad \diagdown \quad \diagup \\ \qquad\qquad\qquad\qquad\qquad z^{(C_{r-2})} \vee z^{(E)} \vee u_{r-1} \overset{*}{\to} z^{(C_{r-2})} \vee z^{(E)} \end{array}$$

$$\cdots$$

since $u_r, u_{r-1} \ldots \in E$.

Thus, by repeating the above we derive $z^{(C_{p-1})} \vee z^{(E)} \vee x$ and $z^{(D_{q-1})} \vee z^{(E)} \vee \bar{x}$. We resolve them to get $z^{(C_{p-1})} \vee z^{(D_{q-1})} \vee z^{(E)}$. If we continued as before we would get clauses of size 4. So, we define a new variable $\hat{z} \equiv z^{(D_{q-1})} \vee z^{(E)}$. We use the new clauses to derive $z^{(C_{p-1})} \vee z^{(D_{q-1})} \vee z^{(E)} \overset{*}{\to} z^{(C_{p-1})} \vee \hat{z}$. Then as before we continue the decomposition to get $z^{(C_{p-2})} \vee z^{(E)} \vee \hat{z}$; then "substitutin back $\hat{z}$ we get

$$\begin{array}{ccc} z^{(C_{p-2})} \vee z^{(E)} \vee \hat{z} & & \bar{\hat{z}} \vee z^{(E)} \vee z^{(D_{q-1})} \\ & z^{(C_{p-2})} \vee z^{(E)} \vee z^{(D_{q-1})} . & \end{array}$$

This process (using new z's) is repeated. The reader can easily see that $z^{(E)}$ is obtained at the end.

For an input clause C one can easily verify that $Z^{(C)}$ can be derived using C and the clauses implied by (1). $(C \overset{*}{\to} Z^{(C)})$. □

6.3. More on the Power of Extended Resolution.

Another property which shows that extended resolution is very powerful was first observed by Tseitin:

Theorem 6.3.1. For any set of clauses which corresponds to a graph there is a resolution proof tree T , which uses extension, such that the number of distinct clauses on T is polynomial.

One way to prove it is to consider a deletion process and to choose one subgraph for every set of subgraphs which have the same exterior boundary. We denote by $G_{\tilde{i}}$ the subgraph wich represents G_i and introduce a new variable $z_{\tilde{i}}$, where $z_{\tilde{i}} \equiv$ sum modulo 2 of the edge labels of $F_{\tilde{i}}^{ext}$. One can show that using the tricks of Theorem 6.1.2. with $+$ instead of V, there is a proof tree for $z_{\tilde{i}}^{\varepsilon_{\tilde{i}}}$ where $\varepsilon_{\tilde{i}}$ is the parity of G_i ; i.e., if x_r splits G_r into G_{r_1} and G_{r_2} then one can construct a proof tree for $z_{\tilde{r}}^{\varepsilon_{\tilde{r}}}$ using $z_{\tilde{r}_1}^{\varepsilon_{\tilde{r}_1}}$ and $z_{\tilde{r}_2}^{\varepsilon_{\tilde{r}_2}}$. The contradiction is derived from the decomposition of z_p and $\bar{z}_q$ where $G_p \cup G_q$ is the whole graph and G_p (G_q) has parity 1 (0) . If the graph has $O(n)$ vertices and edges this technique yields a tree with $O(n^2)$ distinct clauses. A proof tree with $O(n \log n)$ clauses is obtained† if we use graphs which are not necessarily connected. This is achieved by taking a full binary tree, G_1 = the given graph and if i_1 and i_2 are the sons of i , then G_{i_1} (G_{i_2}) consists of exactly half the vertices of G_i and all the edges in $\bar{G}$ among them.□

So, although the sets of clauses associated with graphs defeat regular resolution, bounded resolution (and as will turn out enumeration procedures as well), they are by no means hard inputs. The reason for this is that resolution in some sense is analogous to operating with AND and OR and the graph examples are typically sum mod 2 which is exponential in AND and OR.

---

† This was independently observed by Kirkpatrick [19].

Similarly, Tseitin also observed that using extension one can transform any Boolean formula α into a formula β in CNF such that $|\beta| \leq O(|\alpha|)$. (In fact Tseitin gave a transformation into 3-CNF.) On the other hand, we know that without introducing new variables there are formulas α such that the equivalent β in CNF must be of length exponential in $|\alpha|$ (e.g., $x_1 + x_2 + \ldots + x_n$).

7. Enumeration Procedures.

7.1. Introduction.

So far we discussed only resolution procedures. They all use a "bottom up" approach: They construct a tree from the bottom up by starting with clauses at the leaves and using the resolution rule. In this section we consider procedures which use a "top down" approach. They start with the given set of clauses at the root and construct a tree from the root down, in a way which will be described below. The tree will represent (implicitly) an enumeration of all 2^n possible assignments. However, its size need not be 2^n since if, for example, one of the clauses is $x_1 \vee \bar{x}_2$, then each assignment with $x_1 = 0$, $x_2 = 1$ does not satisfy the given set of clauses.

The outline of this section is as follows: a We introduce two classes of enumeration procedures. b We then describe a close relationship between these procedures and regular resolution. c Then we consider the issue of the complexity of these procedures. It will turn out that although they have complexity not larger than regular resolution, and in some cases much smaller, they are still exponential. d The relationship between our results and the results of Cook and Reckhow on the complexity of semantic trees will be briefly discussed. e We will conclude the section by describing some implications of our results to the complexity of integer programming routines.

7.2. The Two Procedures.

A partial assignment (complete assignment) is an assignment which specifies the values of some (all) of the variables. There exists an obvious partial order between partial assignments: $\delta_1 \preceq \delta_2$ if all the variables which are assigned by δ_1 are also assigned by δ_2 and to the same values.

Given a set of clauses and a partial assignment δ we obtain another set of clauses S^δ by substituting all values given by δ : $S^\delta = \{C' \mid \exists\, C \in S , C' \subseteq C , C'$ contains all literals in C which are not assigned by δ , and all the literals in $C - C'$ are assigned 0 by $\delta\}$. Note that whenever a literal in $C \in S$ is assigned the value 1 by δ , C is dropped since its value is 1 for every complete assignment $\delta' \succeq \delta$. If δ assigns a value to exactly one variable, say $x = a$, we will write $S^{x=a}$ for S^δ .

A simple enumeration tree (s.e. tree) using a given set of clauses S is a binary tree T such that 1) the root of T contains S ; 2) every leaf of T contains a set of clauses one of which is Λ ; and 3) if S_i labels an interior vertex $i \in T$, then a) $\Lambda \notin S_i$ and b) there is a variable x which appears in S_i such that $S_i^{x=0}$ and $S_i^{x=1}$ label the sons of i . We will say that the variable x is chosen at i . Note that we can associate with each vertex i in an s.e. tree a partial assignment δ_i , which is implied by the path from the root to i . The set of clauses at i is denoted by S_i . Obviously $S_i = S^{\delta_i}$.

Theorem 7.2.1. S is contradictory iff there is an s.e. tree using S.

Proof. An easy induction: if S contains Λ the tree consists of one vertex and vice versa. Otherwise, let x be a variable in S. Then S is contradictory iff $S^{x=0}$ and $S^{x=1}$ are contradictory iff (by induction hypothesis) there are s.e. trees for $S^{x=0}$ and $S^{x=1}$ iff there is an s.e. tree for S. □

The complexity of an s.e. tree T, $N(T)$, is the number of distinct labels on its vertices. Obviously, there is no benefit in repetition of the proof that $S_i = S_j$ is contradictory. One can also define the straight line version (like in the case of resolution) in which such repetition does not occur.

A simple enumeration procedure (s.e.p.) is simply constructing an s.e. tree using the given (contradictory) set of clauses S. The complexity of s.e.p. is defined by $\mathrm{Comp}_{sep}(n) = \max_{|S|=n} \min N(T)$, where the minimum is taken over all s.e. trees using S.

The definition of an enumeration tree (e. tree) T using S is very similar to the definition of s.e. tree. The only differences are a the root contains a subset of S, and b if i_1, i_2 are the sons of i, then $S_{i_1} \subseteq S_i^{x=0}$ and $S_{i_2} \subseteq S_i^{x=1}$. (In s.e. trees, $S_{i_1} = S_i^{x=0}$ etc.)

Theorem 7.2.2. S is contradictory iff there is an e. tree using S.

Proof. Obvious. (To show that S is contradictory it suffices to show that $S' \subseteq S$ is contradictory.) □

The complexity $N(T)$ of an e. tree T, enumeration procedure (e.p.), and $\mathrm{Comp}_{ep}(n)$ are defined in the obvious way. Obviously, every s.e. tree is an e. tree. Thus, $\mathrm{Comp}_{sep}(n) \geq \mathrm{Comp}_{ep}(n)$.

Given an e. tree T we can obtain an s.e. tree $\hat{T}$ by choosing the same variables as in T . Obviously for $i \in T$, $S_i \subseteq \hat{S}_i$. $\hat{T}$ might be even smaller than T : in $\hat{T}$ a vertex can be closed (since one of the clauses in it becomes empty), while the corresponding vertex in T is not closed. (Since that clause was deleted in one of the ancestors of the vertex. Recall that in e. tree $S_{i_1} \subseteq S_i^{x=0}$, i.e., some clauses might be deleted.) But although the size of T is not smaller than the size of $\hat{T}$, $N(T)$ may be much smaller than $N(\hat{T})$. For example, if I is a set of vertices in T , we may have $\hat{S}_i = \bar{S} \cup S_i'$ and $S_i = \bar{S}$ for $i \in I$; i.e., in the s.e. tree they all differ, while in the e. tree they are all the same.

7.3. Enumeration Procedures and Regular Resolution.

A regular proof tree and an e. tree have the same structure if the two trees without the labelling are the same, and if x is chosen at a vertex in the one, then x is annihilated at the corresponding vertex in the other.

Theorem 7.3.1. If T is a regular proof tree using S , then there is an e. tree $\tilde{T}$ using S , such that T and $\tilde{T}$ have the same structure.

Proof. Let α_i be the set of clauses on the leaves of T_i , let A_i be the set of literals annihilated on the path from the root of T to

$\hat{S}_{i_1}$ and $\hat{S}_{i_2}$ are nonempty. Again by minimality there must be at least one clause in S_{i_1} (S_{i_2}) which originated in a clause in $\hat{S}_{i_1}$ ($\hat{S}_{i_2}$). In the leaves of $\tilde{T}_{i_1}$ ($\tilde{T}_{i_2}$), replace the clauses of $\hat{S}_{i_1}^{x=0}$ ($\hat{S}_{i_2}^{x=1}$) by the corresponding clauses in $\hat{S}_{i_1}$ ($\hat{S}_{i_2}$). The new tree $\tilde{T}'_{i_1}$ ($\tilde{T}'_{i_2}$) is a regular proof tree for x ($\bar{x}$), using S_i. $\tilde{T}_i = \mathrm{Res}(\tilde{T}'_{i_1}, \tilde{T}'_{i_2})$ is a regular proof tree using S_i and have the same structure as T_i. □

Note that if T is the e. tree of Figure 7.3.1., then $\tilde{T}$ is the regular proof tree of Figure 1.3.1. Corollary 7.3.1. follows immediately from Theorem 7.3.2. and Theorem 7.2.2:

Corollary 7.3.1. If S is contradictory, then there is a regular proof tree using S.

The requirement that T is minimal is probably not crucial and can be replaced by another assumption. However, it cannot be dropped, since there are e. trees that do not have corresponding regular trees of the same structure. A trivial example for such an e. tree appears in Figure 7.3.2.

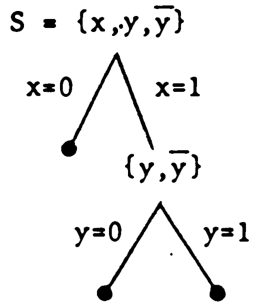


Figure 7.3.2.

7.4. The Complexity of Enumeration Procedures.

Theorem 7.4.1. If T is a regular tree using S, then there is an e. tree $\tilde{T}$ using S such that $N(\tilde{T}) \leq N(T)$.

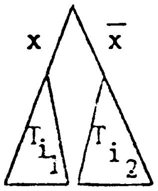
Proof. W.l.o.g. if a clause appears more than once in T, then it is derived by the same subtree. We show that $\tilde{T}$, the e. tree which corresponds to T by Theorem 7.3.1., satisfies the above. We assume that $C_i = C_j$ in T and show that $\tilde{S}_i = \tilde{S}_j$ in $\tilde{T}$. This will imply that $N(\tilde{T}) \leq N(T)$ and will complete the proof.

Recall that $S_i = \alpha_i^{\delta_i}$ and $S_j = \alpha_j^{\delta_j}$. By our assumption on T, $T_i = T_j$ and $\alpha_i = \alpha_j$. Let $Var^{(\ell)} = Var_1^{(\ell)} \cup Var_2^{(\ell)}$, where $Var^{(\ell)}$ $(Var_1^{(\ell)}, Var_2^{(\ell)})$ are the variables in α_ℓ (the variables annihilated in T_ℓ, and the variables in C_ℓ respectively). It follows that the three sets for $\ell = i$ are the same as the corresponding sets for $\ell = j$. Also, δ_i (δ_j) assigns values to the variables in $Var_2^{(i)} = Var_2^{(j)}$. Moreover, they both assign the same values, since by the construction $x = 0$ $(x = 1)$ in $\tilde{T}$ correspond to x $(\bar{x})$ in T and since $C_i = C_j$ each variable in C_i (C_j) appears as the same literal when it is annihilated in an ancestor of i (j). Thus $\tilde{S}_i = \tilde{S}_j$. □

The converse of Theorem 7.4.1. is not true. The s.e. tree T in Figure 7.4.1. satisfies $N(T) = 3$, but every regular proof tree $\tilde{T}$ using the same set of clauses must satisfy $N(\tilde{T}) \geq 7$.

i , and let δ_i be a partial assignment which assigns all the literals in A_i the value 0 and does not specify the others. $\tilde{T}$ is obtained from T by relabelling: for each vertex i let $\tilde{S}_i \equiv \alpha_i^{\delta_i}$.

We show by induction on the height of $\tilde{T}_i$ that $\tilde{T}_i$ is an e. tree using $\tilde{S}_i$: if i is a leaf, then since the literals of C_i are annihilated on the path to the root $\tilde{S}_i = \alpha_i^{\delta_i} = \{C_i\}^{\delta_i} = \Lambda$ and T_i is an e. tree using $\tilde{S}_i$. If

$T_i =$ 

,

then since $\alpha_{i_1} \subseteq \alpha_i$,

$$\tilde{S}_{i_1} = \alpha_{i_1}^{\delta_{i_1}} \subseteq \alpha_i^{\delta_{i_1}} = (\alpha_i^{\delta_i})^{x=0} = \tilde{S}_i^{x=0} .$$

Similarly $\tilde{S}_{i_2} \subseteq \tilde{S}_i^{x=1}$. □

Obviously, if T is nonregular, there cannot be an e. tree with the same structure as T . Figure 7.3.1. describes the s.e. tree $\tilde{T}$ which corresponds to the regular proof tree of Figure 1.3.1. An e. tree is <u>minimal</u> if every set of clauses is minimally inconsistent, i.e., it is contradictory but any one of its proper subsets is satisfiable. Note that if a set of clauses S is minimally inconsistent and a variable z appears in S , then both z and $\bar{z}$ must appear in S .

$\alpha_i = \{x \vee y, \bar{y} \vee u, \bar{u}\}$

$A_i = \{x\}$

$\delta_i = \{x=0\}$

$\hat{S}_i = \alpha_i^{\delta_i} = \{y, \bar{y} \vee u, \bar{u}\}$

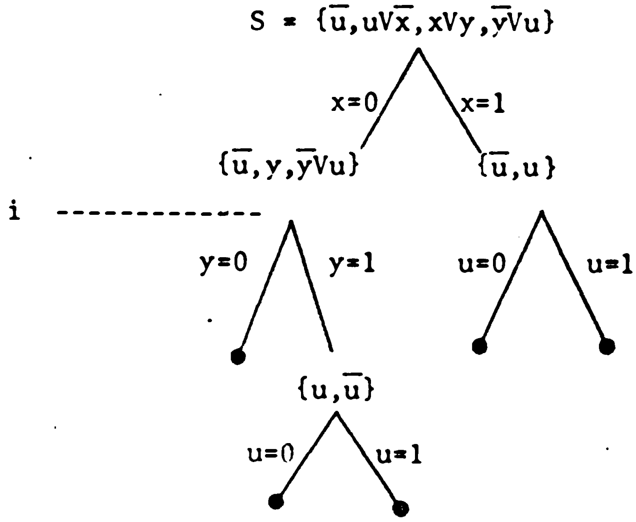
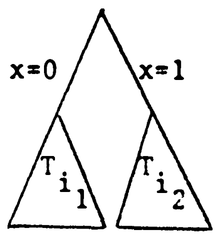


Figure 7.3.1.

Theorem 7.3.2. If T is a minimal e. tree using S, then there is a regular proof tree $\tilde{T}$ using S, such that T and $\tilde{T}$ have the same structure.

Proof. Induction on the height of T. Basis is trivial. Consider the tree

$T_i =$ 

By definition $S_{i_1} \subseteq S_i^{x=0}$ and $S_{i_2} \subseteq S_i^{x=1}$. By induction hypothesis there is a regular proof tree $\tilde{T}_{i_1}$ $(\tilde{T}_{i_2})$ using S_{i_1} (S_{i_2}) with the same structure as T_{i_1} (T_{i_2}). Let $\hat{S}_{i_1}$ $(\hat{S}_{i_2})$ be the set of clauses in S_i which contain x $(\bar{x})$. Since S_i is minimally inconsistent

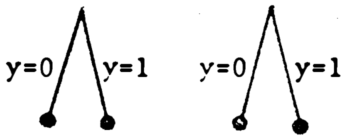


Figure 7.4.1.

In the sequel we will give an e. tree T such that its corresponding regular tree (by Theorem 7.3.2.) has a complexity much larger than that of T. Nevertheless, enumeration procedures are exponential.

Theorem 7.4.2. $\mathrm{Comp}_{ep}(n) \geq 2^{n/288(\log n)^2}$.

Corollary 7.4.1. $\mathrm{Comp}_{sep}(n) \geq 2^{n/288(\log n)^2}$.

To prove Theorem 7.4.2. we will show that if T is an e. tree using a contradictory set of clauses S which corresponds to the graph $\overline{G}$ of Section 2, then $N(T) \geq 2^{n/36(\log n)^2}$ $(|S| = 8n)$.

We now define a labelled deletion process, which is similar to a deletion process. It generates a tree and a set of graphs $\{G_i\}$. In addition, each graph G_i has a labelling ℓ_i, such that by considering the tree and $\{\alpha(G_i(\ell_i))\}$ we obtain an e. tree.

Let $G = (V,E)$ be a labelled graph, let $z \in E$ and $a \in \{0,1\}$. Assume we delete z from E and do not change the labelling except possibly the endpoints of z by adding to them (mod 2) $1 - a$. We

involved.

† This stronger statement was suggested by S. Cook in a private communication.

Lemma 7.4.1. $\alpha(G-z^a) = \alpha(G)^{z=1-a}$,

Proof. The set of vertices is the same in both graphs. For $v \in V$ which is not an endpoint of z nothing changes. So its corresponding clauses appear in both sets of clauses. Let v be an endpoint of z and assume it has degree k . There are 2^{k-1} clauses in $\alpha(G)$ which correspond to v . 2^{k-2} of them contain z^a (z^{1-a}) . By substituting $z = 1 - a$, those which contain z^{1-a} drop and those which contain z^a yield 2^{k-2} distinct clauses. Let $C = z^a \vee z_1^{\varepsilon_1} \vee \dots \vee z_{k-1}^{\varepsilon_{k-1}}$, and assume it satisfies the parity condition w.r.t. v , i.e., $(1-a) + \sum_{i=1}^{k-1} (1-\varepsilon_i) \neq \varepsilon_v$. Thus, $\sum_{i=1}^{k-1} (1-\varepsilon_i) \neq \varepsilon_v + (1-a)$. But this means that the clause obtained from C by substituting $z = 1 - a$ satisfies the parity condition w.r.t. v in $G - z^a$. ($\varepsilon_v + (1-a)$ is the new label of v .) Hence exactly all 2^{k-2} clauses which corresponds to v in $G - z^a$ appear in $\alpha(G)^{z=1-a}$. $\square$

The labelled deletion process on $\bar{G}$ starts by constructing a root and labelling it by $\alpha(\bar{G})$. Each vertex i in the tree will contain a labelled subgraph of $\bar{G}$ $G_i(\ell_i)$, such that $S_i = \alpha(G_i(\ell_i))$ is contradictory. Assume i has been constructed. If G_i is a single vertex, then i is a leaf. Otherwise, choose x , an edge in G_i .

programming problem given by $\max\{x_1 : 2x_1+2x_2+\dots+2x_n=n\}$. Then every

† In this section only + stands for regular sum and not for mod 2 sum.

ℓ_{i_1} (ℓ_{i_2}) is the labelling implicit in $G_i - x$ $(G_i - \bar{x})$. Now define G_{i_1} (G_{i_2}) so that $S_{i_1} = \alpha(G_{i_1}(\ell_{i_1}))$ $(S_{i_2} = \alpha(G_{i_2}(\ell_{i_2})))$ is contradictory:

<u>case 1.</u> x does not split G_i. Define $G_{i_1} = G_i - x$ $(G_{i_2} = G_i - \bar{x})$. S_{i_1} is contradictory since $\varepsilon(G_{i_1}(\ell_{i_1})) = \varepsilon(G_i(\ell_i)) = 1$. Similarly S_{i_2} is contradictory.

<u>case 2.</u> x splits G_i. Since $\varepsilon(G_i(\ell_{i_1})) = \varepsilon(G_i(\ell_i)) = 1$ exactly one of the components must have parity 1 under ℓ_{i_1}. Denote it by G_{i_1} and denote the second one by G_{i_2}. Since $\varepsilon(G_{i_j}(\ell_{i_1})) \neq \varepsilon(G_{i_j}(\ell_{i_2}))$ for $j = 1,2$ we must have $\varepsilon(G_{i_2}(\ell_{i_2})) = 1$ and $\varepsilon(G_{i_2}(\ell_{i_1})) = 0$. Hence, S_{i_1} and S_{i_2} are contradictory.

By Lemma 7.4.1., and since G_{i_1} is a subgraph of $G_i - x$, $S_{i_1} = \alpha(G_{i_1}(\ell_{i_1})) \subseteq \alpha(G_i - x) = \alpha(G_i(\ell_i))^{x=0} = S_i^{x=0}$; i.e., $S_{i_1} \subseteq S_i^{x=0}$.

tree generated by implicit enumeration procedure must have $2^{\lceil n/2 \rceil}$ vertices. Since the proof is so simple we give it here. (Our proof is simpler than the original proof which uses an unnecessary weight argument.)

<u>Proof.</u> Each vertex in the tree whose distance from the root $\leq \lfloor n/2 \rfloor$ cannot be fathemed. (The corresponding linear programming is still feasible.) Thus, the tree must contain a full binary tree of depth $\lceil n/2 \rceil$. $\square$

Proof. In Section 3 we proved a one to one correspondence between (unlabelled) deletion processes and regular proof trees. The one to one correspondence between (unlabelled) deletion processes and labelled deletion processes is implicit above. (When G_i splits G_{i_1} and G_{i_2} are determined uniquely.) $\square$

Note that the corresponding trees have the same structure and that if x $(\bar{x})$ is annihilated on an edge in the one, then x is set to 0 (1) on the corresponding edge of the other. The latter follows from the fact that when G_i splits, then (1) in the case of regular proof tree (Lemma 3.3.1.) G_{i_1} (G_{i_2}) is determined so that the complements of the variables in its exterior boundary have parity opposite to its parity, i.e., so that C_{i_1} (C_{i_2}) is regular; (2) in the case of regular e. tree G_{i_1} (G_{i_2}) is determined so that $\varepsilon(G_{i_1}(\ell_{i_1})) = 1$ $(\varepsilon(G_{i_2}(\ell_{i_2})) = 1)$. But this happens when the parity of the number of variables in its exterior boundary set to 1 is opposite to its original parity.

The reason is that CNF is a "lower level language" in comparison to systems of equations. To express the unique equation of Theorem 7.6.1. (7.6.2.) requires an exponential number of clauses (without using additional variables).

We included Corollary 7.6.1. although it can be proved directly using a much simpler proof. The reason for doing it is that we conjecture that the problem which appears there (i.e., the zero-one programmin problem obtained using the set of clauses which corresponds to a graph)

Fact 1. If $S'_i \subseteq S_i$ is contradictory, then it contains at least one clause corresponding to every vertex of G_i. Fact 1 implies

Fact 2. If $S'_i \subseteq S_i$ and $S'_j \subseteq S_j$ are both contradictory and $S'_i = S'_j \neq \{\Lambda\}$, then $G_i = G_j$ and $\ell_i = \ell_j$.

Let $G = (V,E)$ be a connected subgraph of $\overline{G}$. A set of clauses S is G-contradictory under (labelling) ℓ if (1) $S = S' \cup S''$; (2) S'' is satisfiable and its variables label edges which do not have any endpoint in G; and (3) S' is a contradictory subset of $\alpha(G(\ell))$.

Fact 3. If S is G-contracidtory under ℓ and $\overline{S} \subseteq S$ is contradictory, then $\overline{S}$ is G-contradictory under ℓ.

Assume that (1) $S_i = S'_i \cup S''_i$ is G_i-contradictory under ℓ_i (S'_i, S''_i as above), (2) S_i is the set of clauses at a vertex i in an e. tree, and (3) x is chosen at i. There are three cases:

Case 1'. x labels an edge in G_i but does not split G_i.

Case 2'. x splits G_i.

Case 3'. x appears in S''_i.

Case 1' (2') is similar to case 1 (2) in the definition of labelled deletion process. Defining ℓ_{i_1} (ℓ_{i_2}) and G_{i_1} (G_{i_2}) as we did there, using a similar argument plus Fact 3 implies that S_{i_1} is G_{i_1}-contradictory under ℓ_{i_1} (S_{i_2} is G_{i_2}-contradictory under ℓ_{i_2}). In Case 3', since S''_i is satisfiable, then at least one of $\{S''^{x=0}_i, S''^{x=1}_i\}$

is satisfiable. Thus, by Fact 3 at least one of $\{S_{i_1}, S_{i_2}\}$ is G_i-contradictory under ℓ_i. If both are G_i-contradictory we choose (arbitrarily) $j = i_1$, otherwise, we select the only $j \in \{i_1, i_2)$ such that S_j is G_i-contradictory under ℓ_i. We call j the important son of i.

Proof of Lemma 7.4.3. Given the e. tree T we construct the regular e. tree from the root down. Assume the set of clauses at vertex j in T ($\tilde{T}$) is S_j ($\tilde{S}_j$). Each vertex i in $\tilde{T}$ will have a unique corresponding vertex j in T such that

(1) S_j is G_i-contradictory under labelling ℓ_i and

(2) $\tilde{S}_i = \alpha(G_i(\ell_i))$.

We start with the root of $\tilde{T}$ corresponding to the root of T and let $\tilde{S}_1 = S_1 = S = \alpha(\overline{G}(\ell))$. Thus, (1) and (2) above hold for $i = j = 1$ Assume i and j satisfy (1) and (2). If G_i is a single vertex, then i is a leaf of $\tilde{T}$. Otherwise define

$$\text{representative } (j) = \begin{cases} j & \text{if the variable chosen at } j \text{ labels an edge in } G_i \\ \text{representative (important son of } j \text{) otherwise.} \end{cases}$$

Let $j' =$ representative (j). By Case 3' above j' exists and S_j is G_i-contradictory under ℓ_i. Let x be the variable chosen at j'. By definition of representative (j), x labels an edge in G_i. We choose x at i and construct i_1 (i_2) as the son of i which corresponds to the substitution $x = 0$ ($x = 1$). The vertex in T

which corresponds to i_1 (i_2) is the son of j' -- j_1 (j_2) which corresponds to the same substitution. By cases 1' and 2' above S_{j_1} (S_{j_2}) is G_{i_1} (G_{i_2}) - contradictory under ℓ_{i_1} (ℓ_{i_2}), which is obtained from ℓ_i by the labelled deletion of x $(\bar{x})$ from G_i. We define $\tilde{S}_{i_1}$ $(\tilde{S}_{i_2})$ so that they satisfy (2). Hence, (1) and (2) hold for i_1,j_1 and i_2,j_2 respectively. Thus, $\tilde{T}$ is a regular e. tree using S.

To show that $N(\tilde{T}) \leq N(T)$ it suffices to show that if $j',j'' \in T$ and corresponds to $i',i'' \in T$, then if $S_{j'} = S_{j''}$ then $\tilde{S}_{i'} = \tilde{S}_{i''}$. So assume $S_{j'} = S_{j''}$. Since $S_{j'}$ is $G_{i'}$ - contradictory under $\ell_{i'}$ and $S_{j''}$ is $G_{i''}$ - contradictory under $\ell_{i''}$, then by Fact 2 $G_{i'} = G_{i''}$ and $\ell_{i'} = \ell_{i''}$. Hence, $\tilde{S}_{i'} = \alpha(G_{i'},(\ell_{i'})) = \alpha(G_{i''},(\ell_{i''})) = \tilde{S}_{i''}$. Therefore, $N(\tilde{T}) \leq N(T)$. □

Let T be a regular e. tree using $S = \alpha(\bar{G})$, where $\bar{G}$ is of degree 3, and let T' be the regular proof tree which corresponds to it. For a vertex i in both trees let G_i be the corresponding subgraph, C_i' the clause at i in T' and S_i the set of clauses at i in T. We want to find out what is the maximum number of distinct clauses $\{C_j'\}$ such that $S_j = S_i$. Obviously if $S_i = S_j$, then $G_i = G_j$. We say that C_i' and C_j' <u>differ in x</u> if one of them contains x and the other contains $\bar{x}$.

<u>Lemma 7.4.4</u>. Assume $C_i' \neq C_j'$ and $S_i = S_j \neq \{\Lambda\}$. Let x,y,z be the edge labels of the edges incident with a vertex v in $\bar{G}$, such that

C_i' and C_j' differ in x. Then C_i' and C_j' differ also in exactly one of $\{y,z\}$.

Proof. Since $G_i = G_j$, C_i' and C_j' contain the same variables. Obviously, all the three cannot appear in C_i' (C_j') since G_i is connected. (Recall, that the literals in C_i' correspond to the edges in the boundary of G_i.) By Fact 2 $\ell_i = \ell_j$ (the labellings in the regular e. tree). W.l.o.g. $x \in C_i'$ $(\overline{x} \in C_j')$ which implies that x was set to 0 (1) at an ancestor of i (j) in T (by the correspondence of Lemma 7.4.2.), i.e., at that ancestor of i (j) there was a labelled deletion of x $(\overline{x})$. If C_i' and C_j' do not differ in one of $\{y,z\}$, then the vertex label of v in ℓ_i must differ from that in ℓ_j - contradiction. $\square$

Lemma 7.4.5. Assume (1) $C_i' \neq C_j'$ and $S_i = S_j \neq \{\Lambda\}$, (2) $x \in F_i^{ext}$ whose endpoint in G_i is u and (3) C_i' and C_j' differ in x. Then (1) there is $y \in F_i^{ext}$ whose endpoint in G_i is v, (2) there is a path $x_1, x_2, \ldots, x_r$ from u to v, $x_j \in F_i^{int}$ $1 \leq j \leq r$ and (3) C_i' and C_j' differ in y and in x_k for all $1 \leq k \leq r$.

Proof. Let $x \equiv x_0$ and $y \equiv x_{r+1}$. The existence of x_{k+1} follows from the existence of x_k by Lemma 7.4.4. In constructing the path one cannot get a circle since by Lemma 7.4.4. C_i' and C_j' cannot differ in all three variables incident with the same vertex. The only way the path can terminate is when $x_{r+1} \in F_i^{ext}$. $\square$

$C_i' = x \vee \bar{x}_1 \vee \bar{x}_2 \vee x_3 \vee \bar{x}_4 \vee y \vee \ldots$

$C_j' = \bar{x} \vee x_1 \vee x_2 \vee \bar{x}_3 \vee x_4 \vee \bar{y} \vee \ldots$

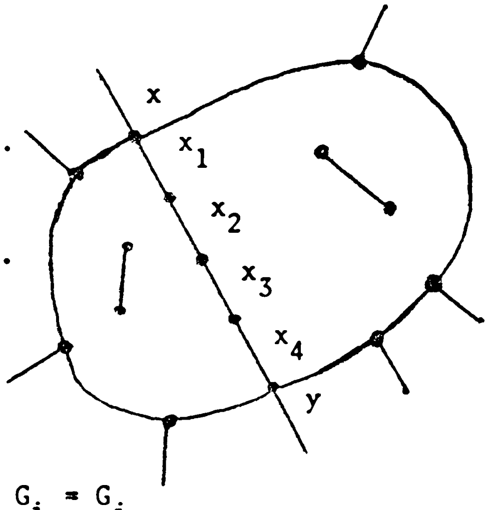


Figure 7.4.1. $G_i = G_j$

<u>Lemma 7.4.6</u>. For every interior vertex $i \in T$ there are at most $2^{1/2|F_i^{ext}|+|F_i^{int}|}$ distinct clauses C_j' such that $S_i = S_j$.

<u>Proof</u>. Assume $S_i = S_j$. If C_i' and C_j' differ in $x \in F_i^{ext}$, then by Lemma 7.4.5. they must differ in $y \in F_i^{ext}$ which is determined uniquely. So we can pair (x,y) and infer that C_i' and C_j' either agree in x and y or they differ in x and y. If (x_1,y_1) is another pair, then $x \neq x_1$ and $y \neq y_1$ since by the proof of the previous lemma the path between the endpoints of x and y cannot split. There are at most $1/2|F_i^{ext}|$ pairs, and by the above argument there are at most two ways to assign complements to each pair. Thus, the above upper bound follows. (To obtain the bound we do not restrict the way to assign complements in the interior boundary.) $\square$

Before proving Theorem 7.4.2. we sketch an example for a graph $\hat{G}$, an e. tree T using $\alpha(\hat{G})$ with $N(T) \leq cn$ (c is a constant) such that its corresponding regular proof tree $\tilde{T}$ (by Theorem 7.3.2.) which satisfies $N(\tilde{T}) \geq 2^n$: Let $\hat{G}$ be the graph in Figure 7.4.2. It has n vertices (edges) on a circle and $2n$ vertices ($3n$ edges) out of

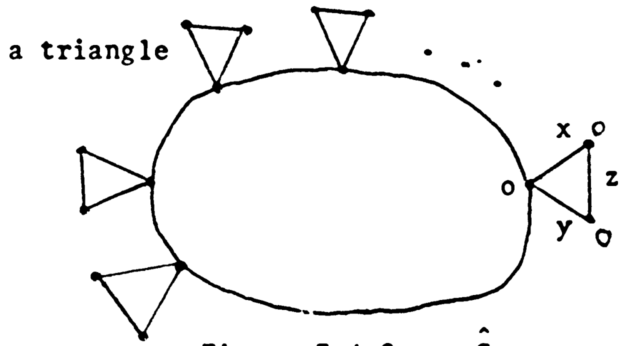


Figure 7.4.2. $\hat{G}$

the circle. Consider an (unlabelled) deletion process which always deletes first edges not in the circle triangle by triangle. Let T ($\tilde{T}$) be the corresponding e. tree (regular proof tree). Let G_i be the circle. Both T and $\tilde{T}$ have $2^{s_i} = 2^n$ vertices labelled by G_i. In T all the clauses on these vertices are distinct, e.g., in the distinguished triangle either x,y or $\overline{x},\overline{y}$ can be in the clause (by a parity argument). But in $\tilde{T}$ setting both x,y to 0 or both to 1 yields the same set of clauses which correspond to G_i. One can verify that indeed $N(\tilde{T}) \leq cn$ for some constant c. (In this example the upper bound of Lemma 7.4.6. is achieved. Note that the paths mentioned in Lemma 7.4.5. are all of zero length.)

Proof of Theorem 7.4.2. Let T be an optimal regular e. tree using $S = \alpha(\overline{G})$ and let T' be the regular proof tree which corresponds to it. Let $R_1,R_2,\ldots,R_{\hat{N}}$ ($\hat{N} = N(T)$) be the distinct labels on T and let $D_1,D_2,\ldots,D_{N(T')}$ be the distinct clauses on T'. Let $I_j =$

$\{\ell | C'_\ell = D_j\}$ and $J_j = \{\ell | S_\ell = R_j\}$ for the corresponding j's . We showed in the proof of Theorem 5.2.1. (cf. formula (2) of §5.2) that

$$\sum_{\ell \in I_j} 2^{-n_\ell + s_\ell} \leq 1 . \tag{1}$$

Since if $C'_{\ell_1} = C'_{\ell_2}$, then $S_{\ell_1} = S_{\ell_2}$, it follows that

$$J_i = \bigcup_{r=1}^{n_i} I_{i_r} . \tag{2}$$

By Lemma 7.4.5., $n_i \leq 2^{1/2|F_i^{ext}| + |F_i^{int}|}$, so by (1) and (2) we derive

$$\sum_{\ell \in J_i} 2^{-n_\ell + s_\ell} \leq 2^{1/2|F_i^{ext}| + |F_i^{int}|} .$$

But by Lemma 2.3.4. and the fact that $G_\ell = G_i$ for every $\ell \in J_i$ $s_\ell \geq 2/3|F_\ell^{ext}| + |F_\ell^{int}| = 2/3|F_i^{ext}| + |F_i^{int}|$. Hence,

$$\sum_{\ell \in J_i} 2^{-n_\ell + 2/3|F_i^{ext}| + |F_i^{int}|} \leq 2^{1/2|F_i^{ext}| + |F_i^{int}|} ,$$

i.e., $\sum_{\ell \in J_i} 2^{-n_\ell + 1/6|F_\ell^{ext}|} \leq 1$ and summing it up for $i = 1,2,\ldots,\hat{N}$ yields

$$\hat{N} \geq \sum_\ell 2^{-n_\ell} \cdot 2^{1/6|F_\ell^{ext}|} . \tag{3}$$

From this point on the proof proceeds as in Theorem 5.2.1. Formula (3) above has the same role as formula (3) there. The only difference is that $1/6|F_\ell^{ext}|$ replaces s_ℓ . Hence, for $\overline{G}$ the graph of Section 2 $\hat{N} \geq 2^{n/36(\log n)^2}$. By Lemma 7.4.3. and since $|S| = 8n$, $\mathrm{Comp}_{\mathrm{Reg}}(n) \geq 2^{n/288(\log n)^2}$. □

Note that Theorem 5.2.1. (the exponential lower bound for regular resolution) follows from Theorem 7.4.1. and Theorem 7.4.2. On the other hand, Theorem 7.4.2. does not follow from Tseitin's result. We included the proof for the latter since in that case the lower bound was achieved. This implied that DPP is as good as regular resolution for sets of clauses which correspond to graphs.

7.5. Enumeration Trees and Semantic Trees.

In [6] the authors considered a special version of semantic trees as a procedure for proving that a set of clauses is contradictory: A semantic tree for a set of clauses S is a binary tree T such that for each interior vertex $i \in T$ the two edges leaving i are labelled one with x and the other with $\overline{x}$. The label x $(\overline{x})$ stands for the assignment $x = 0$ $(x = 1)$. A branch is closed for S iff one of the clauses in S has value 0 under the partial assignment determined by the branch. The tree is closed for S iff every branch is closed. Consequently, the authors stated without proof

Theorem. Every closed semantic tree for a set of clauses S can be converted to a resolution tree of the same size and shape by an appropria relabelling, and conversely.

<u>Corollary</u>. Semantic trees are exponential in size.

First note that semantic trees are actually e. trees without the labels. Also, note that the theorem as stated is wrong: (a) We cannot relabel a nonregular proof tree to get a semantic tree (e. tree) and (b) an example of an e. tree (semantic tree) which cannot be relabelled to get a resolution tree appeared in Figure 7.3.2. Nevertheless, the corollary is correct, since one can prove

<u>Theorem</u>. Every closed semantic tree (resolution proof tree) T using a set of clauses S can be converted to a resolution proof tree (closed semantic tree) $\tilde{T}$ of size not larger than T. Moreover $\tilde{T}$ is obtained from T by prunning.†

<u>Proof</u>. The transformation from a resolution proof tree to a semantic tree follows from the Regularity Lemma and Theorem 7.3.1. The transformation from a semantic tree to a resolution tree is achieved by first transforming the e. tree into a minimal e. tree of not larger size and then applying Theorem 7.3.2. We omit the details of the transformation to a minimal e. tree. □

The above corollary follows from this theorem. However, it concerns only the size of the tree and not the number of distinct labels on the tree. This explains why the proof of Theorem 7.4.2. is much more involved.

---

† This stronger statement was suggested by S. Cook in a private communication.

7.6. Some Implications to the Complexity of Integer Programming Routines.[†]

We assume that the reader is familiar with implicit enumeration procedures. Our reference for terminology concerning enumerative approaches is [10, Chapter 4]. Following Theorem 7.4.2. we obtain

Corollary 7.6.1. Implicit enumeration is exponential.

Proof. One can easily transform the set of clauses into a system of inequality with 0-1 valued variables (e.g., $x \vee \bar{y} \vee z$ is transformed into $x + (1-y) + z \geq 1$). The e. tree corresponds to the tree generated by the implicit enumeration. The only difference is that in the latter we may fathem earlier, since we fathem also when the corresponding linear programming problem is unfeasible. One can verify that the problem is feasible at i iff the subgraph G_i contains a cycle. In Theorem 7.4.2. we saw that there must be at least $2^{cn/(\log n)^2}$ distinct labels S_i. One can show that there must be at least this number of distinct labels S_i such that G_i contains a cycle, i.e., i is not fathemed due to the unfeasibility of the linear programming problem. $\boxdot$

Corollary 7.6.1. is not new [15]:

Theorem 7.6.1. Let n be an odd integer, and consider the zero-one programming problem given by $\max\{x_1 : 2x_1 + 2x_2 + \ldots + 2x_n = n\}$. Then every

---

† In this section only $+$ stands for regular sum and not for mod 2 sum.

tree generated by implicit enumeration procedure must have $2^{\lceil n/2 \rceil}$ vertices. Since the proof is so simple we give it here. (Our proof is simpler than the original proof which uses an unnecessary weight argument.)

Proof. Each vertex in the tree whose distance from the root $\leq \lfloor n/2 \rfloor$ cannot be fathemed. (The corresponding linear programming is still feasible.) Thus, the tree must contain a full binary tree of depth $\lceil n/2 \rceil$ . . $\square$

Note that if we do not solve the same problem twice, (i.e., fathem when the problem at a vertex has already appeared), then by always choosing the variables in a fixed order, say $x_n, x_{n-1}, \ldots$, the only distinct problems that we get are of the form $2x_1 + \cdots 2x_r = m$, $1 \leq r, m \leq n$. So there are at most n^2 distinct problems. Here again we notice the difference between the size of the tree and the number of distinct labels on its vertices.

Theorem 7.6.2. Let m be an integer, $N = \sum_{i=1}^{m} 2^{i+m} - 1$, $H(x,y) = \sum_{i=1}^{m} [2^{i+m}x_i + (2^{i+m}+2^i)y_i]$, and consider the zero-one programming problem given by $\max\{x_1 : H(x,y)=N\}$. Then every tree generated by implicit enumeration procedure must have $2^{\lceil m/2 \rceil}$ vertices, which contain distinct problems.

The proof of Theorem 7.6.2. is more involved than that of Theorem 7.6.1. Yet it is fairly simple and we do not give it here. One would like to try and prove Theorem 7.4.2. from Theorem 7.6.2. (i.e., to prove the theorem from its corollary). This does not seem to work.

The reason is that CNF is a "lower level language" in comparison to systems of equations. To express the unique equation of Theorem 7.6.1. (7.6.2.) requires an exponential number of clauses (without using additional variables).

We included Corollary 7.6.1. although it can be proved directly using a much simpler proof. The reason for doing it is that we conjecture that the problem which appears there (i.e., the zero-one programming problem obtained using the set of clauses which corresponds to a graph) defeats several other integer programming routines.

8. Conclusion and Open Problems.

We considered several procedures for determining whether a set of clauses is contradictory (a formula in DNF is a tautology). Four of them are nondeterministic and use the resolution rule: (1) The David Putnam procedure; (2) regular resolution; (3) unrestricted resolution; (4) extended resolution. These procedures have increasing computational power; i.e., each procedure is a special case of the next. They are nondeterministic since in the case of DPP the choice of the eliminated variable is not specified and in the other cases the way to construct the proof tree is not specified. We showed that the DPP and regular resolution have exponential complexity. It still remains to determine:

1) what is the complexity of extended resolution?, and

2) what is the complexity of unrestricted resolution?

As a first step one might try to prove a nonlinear lower bound for these procedures. The fact that these procedures are nondeterministic should not matter: as was pointed out in the introduction, if we found a nondeterministic procedure of polynomial complexity for the tautology problem, then this would settle an important open problem (namely: is NP closed under complementation?).

To prove most of our results we exploited Tseitin's ingenious examples, i.e., the sets of clauses corresponding to graphs. We found out that for these sets of clauses, the best regular tree is a DP tree.

One might ask:

3) are there examples for which DPP generates many more distinct clauses than regular resolution?

The same questions are also open for

4) regular resolution versus unrestricted resolution; and for

5) unrestricted resolution versus extended resolution.

We only know that the answer is positive for regular resolution versus extended resolution. However, we conjecture that this is the case for questions 3) and 5). To answer question 3) one will need a new contradictory set of clauses, other than those corresponding to graphs. It is interesting to note that 5) is open although extended resolution was shown in Section 6 to be much more powerful than unrestricted resolution in many aspects. We feel that the techniques we used, especially the lemmas in Section 3, could be pushed further to show that unrestricted resolution must be exponential for Tseitin's examples. This would settle questions 2) and 5), but not necessarily question 4). It should be noted that for small examples, introducing nonregular derivations does not shorten the proofs. However, as in most combinatorial problems, small examples do not mean anything and large examples are too complicated to analyze completely. The open problem which seems the hardest is problem 1). Extended resolution was shown to be very powerful. Note that the proofs in Section 6 indicate that by adding the extension rule one gets a powerful programming language. In fact Theorem 6.3.1., which shows that extended resolution can prove very efficiently that a set of clauses associated with a graph is contradictory, suggests more than that: We conjecture that any family of contradictory sets of clauses, which is constructed in a direct way similar to that of the graph example cannot be used to show that extended resolution is exponential. This is because if we construct such a family, we must know that each member in the family is contradictory, i.e., we can prove it. We believe that extended resolution can simulate efficiently our proof as is the case with the graph examples. Since extended resolution is so powerful, we feel that question 1) will be settled only when the two open problems involving P and NP are solved.

In addition to the four nondeterministic procedures discussed above we considered three versions of bounded resolution. All these procedures are deterministic. We concluded that several attempts to restrict the size of the clauses generated (by the resolution rule) must fail. We also showed that the corresponding versions of iterated bounded resolution are exponential. To prove the results about bounded resolution we proved an $O(n/(\log n)^2)$ lower bound on the size of clauses in resolution proof trees. Figure 8.1. describes the relative difficulty of some of the problems that we considered. It indicates why we have not been able to settle so far question 2) above.

We also considered (simple) enumeration procedures. They do not use the resolution rule at all, but they are closely related to regular resolution. We proved that they too are exponential. One can define

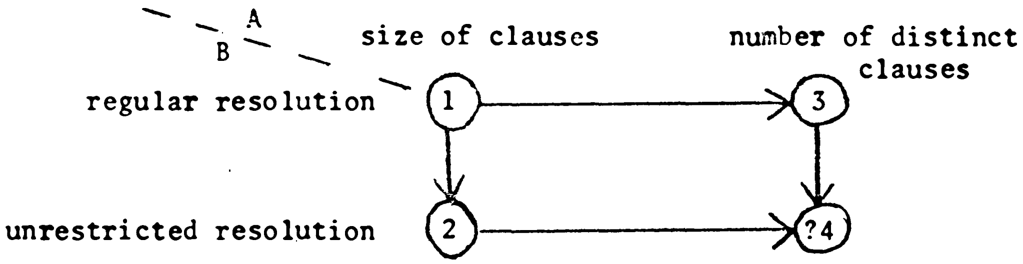


Figure 8.1. Lower bounds on A using B .
An arrow points to the harder problem

1. an easy observation at the end of Section 4;
2. Theorem 4.2.1.;
3. Theorem 5.2.1.;
4. Question 1).

also extended enumeration procedures by using the enumeration procedures plus the extension rule. It is then easy to show that extended enumeration and extended resolution are equal in power. (Each one can efficiently simulate the other.) We briefly discussed some implications to the complexity of integer programming. Several questions are naturally raised:

6) Consider the zero-one integer programming problems which are obtained from contradictory sets of clauses. What is the run-times of the popular integer programming routines on these problems?

The large difference in difficulty between the proof of Theorem 7.6.1. (7.6.2.) and Theorem 7.4.1. is interesting. Although all NP-complete problems are in some sense equivalent, things are less unified when we consider procedures for NP-complete problems. Considering

REFERENCES

[1] Aho, A.V., J.E. Hopcroft and J.D. Ullman. The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading, Mass. (1974).

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

[3] Borodin, A.B. Computational complexity: theory and practice. In Currents in the Theory of Computing, A.V. Aho (ed.), Prentice-Hall, (1973).

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

[5] Cook, S.A. Examples for the Davis-Putnam procedure. Unpublished manuscript, (June 1971).

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

[7] Davis, M. and H. Putnam. A computing procedure for quantification theory. JACM 1, (1960), 201-215.

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

[9] Fischer, M.J. and N. Pippenger. To appear.

[10] Garfinkel, R.S. and G.L. Nemhauser. Integer Programming, Wiley, New York, (1972).

[11] Hennie, F.C. On line Turing machine computations. IEEE Trans. on Electronic Computers, vol. EC-15, (February 1966), 35-44.

[12] Harris, R.V. A polynomial bound on the complexity of the Davis-Putnam procedure as applied to symmetrizable propositions. PhD. Thesis, Dept. of Computer Science Cornell University, (August 1972).

[13] Hartmanis, J. and R.E. Stearns. On the computational complexity of algorithms. Trans. of AMS 117, (May 1965), 285-306.

[14] Hopcroft, J.E. and J.D. Ullman. Formal Languages and their Relation to Automata. Addison-Wesley, Reading, Mass. (1968).

[15] Jeroslow, R.G. Trivial integer programs unsolvable by branch-and-bound. Math. Programming 6, (1974), 105-109.

[16] Johnson, D.S. Approximation algorithms for combinatorial probl Proc. 5th ACM STOC, (1973), 38-49.

[17] Karp, R.M. Reducibility among combinatorial problems. In Complexity of Computer Computations, R.E. Miller and J.W. Thatc (eds.), Plenum Press, New York, (1972), 85-104.

[18] Kirkpatrick, D.G. Personal communication.

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

[20] Knuth, D.E. The Art of Computer Programming: Vol. 3, Sorting a Searching. Addison-Wesley, (1973).

[21] Meyer, A.R. and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. Proc. 13th SWAT, (1972), 125-129.

[22] Paul, W.T. A 2.5N lower bound for the combinational complexity of Boolean functions. Proc. 7th ACM STOC, (1975), 27-36.

[23] Pratt, V.R. The power of negative thinking in multiplying Bool matrices. Proc. 6th ACM STOC, (1974), 80-83.

[24] Quine, W.V. A way to simplify truth functions. American Mathematical Monthly 62, (1956), 627-631.

[25] Rabin, M.O. Theoretical impediments to artificial intelligence Proc. IFIP Mathematical Aspects of Information Processing, (197 615-619.

[26] Robinson, J.A. A machine oriented logic based on the resolutio principle. JACM 12, (1965), 23-41.

[27] Shannon, C.E. The synthesis of two-terminal switching circuits Bell System Tech. J. 28, (1949), 59-98.

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

[29] Simon, J. Personal communication.

[30] Simon, J. On some central problems in computational complexity Ph.D. Thesis, TR 75-224, Dept. of Computer Science Cornell University, (January 1975).

they do not deserve more investigation. If $P \neq NP$, which many researchers believe is the case, then no procedure for the tautology problem can run in polynomial time. In this case, as was pointed out by M. Rabin [25], we must be satisfied with procedures which sometimes give incorrect answers.

In an article called "Toward feasible solutions of the tautology problem" [8], the authors give numerous examples how to check efficiently if the input is a tautology in many special cases. They conclude by expressing the hope that there will be found a way to combine their tricks to construct a general algorithm with polynomial run-time. Unfortunately, they do not give the slightest hint how one can do it ... Before trying to combine the known procedures to construct a good algorithm (or a good approximate algorithm), it might be worthwhile to do the following:

a) For each procedure, to identify some large nontrivial class of inputs on which the procedure runs in polynomial time. (In [12], such class was found for DPP.)

b) For each procedure, to identify some large nontrivial class of inputs on which the procedure runs in exponential time. Our results are a first step in this direction.

approximate solutions to NP-complete problems, it is known that using similar approaches yielded different results for different NP-complete problems [16]. Thus, it is not surprising that it was almost straight forward to show that enumeration procedure is exponential for zero-one programming problems, while it was not trivial at all to show the same for the tautology problem. In fact we could not show that enumeration procedures are exponential for the vertex covering problem. On the other hand, given a procedure one would not like to check it separately for every NP-complete problem. So one may ask

7) Are enumeration procedures exponential for all NP-complete problems?, and

8) Can one prove the above by a general theorem?†

It is interesting to note that the exponential lower bounds we proved for various procedures were of the form $2^{cn/(\log n)^2}$ for some constant c. This means that these procedures are not much more efficient (in the worst case) than the procedure which explicitly checks all 2^n possible assignments. All these lower bounds were obtained using $\alpha(\overline{G})$, where $\overline{G}$ is the graph of Section 2. It is possible that some other graphs will yield higher lower bounds. So, a final open problem is

9) Can the lower bounds be improved?

The fact that some of the procedures are of exponential complexity and that bounded resolution does not always work does not imply that

---

† Obviously, proving that $P \neq NP$ will do.

[31] Stockmeyer, L.J. The complexity of decision problems in automata theory and logic. Ph.D. Thesis, Project MAC TR 133, (July 1974).

[32] Tseitin, G.S. On the complexity of derivations in the propositional calculus. Structures in Constructive Mathematics and Mathematical Logic, Part II. A.O. Silenko (ed.), (1968), 115-125, (Translated from Russian).