Theoretical Computer Science 39 (1985) 297-308
North-Holland

THE INTRACTABILITY OF RESOLUTION

Armin HAKEN
Department of Computer Science, University of Toronto, Toronto, Ontario M5S 1A4, Canada

Communicated by R. M. Karp
Received January 1984
Revised August 1984

Abstract. We prove that, for infinitely many disjunctive normal form propositional calculus tautologies ξ, the length of the shortest resolution proof of ξ cannot be bounded by any polynomial of the length of ξ. The tautologies we use were introduced by Cook and Reckhow (1979) and encode the pigeonhole principle. Extended resolution can furnish polynomial length proofs of these formulas.

1. Definitions and background

1.1. Resolution theorem proving

Theorem proving using resolution was introduced by Robinson [8]. The method is applicable to first-order predicate calculus or to propositional calculus. If the formula to be proved is a consequence of axioms, resolution is used to prove the disjunction of the original formula with the negations of those axioms it depends on. Predicate calculus formulas are reduced to propositional calculus formulas by using quantifier elimination techniques. Furthermore, the formula to be proved is put into *disjunctive normal form* (DNF). The theorem proving task is then reduced to proving that a given DNF propositional calculus formula is a tautology. For the purpose of showing nonpolynomial complexity, we consider only DNF propositional calculus formulas. We use the notation '+' for logical 'or', juxtaposition for 'and', and ''' for negation (x' is 'not x'). An example of a DNF propositional tautology is: $abc'+ab'd+ab'c'd'+a'd+a'c'd'+c$.

To define *resolution* we let ξ be a DNF propositional calculus tautology, and we describe how resolution produces a proof of ξ. The conjunctions of which ξ is a disjunction are called *clauses* and ξ is considered to be a set of clauses. The variables and the negated variables of which a clause is a conjunction are called *literals*. A clause is considered a set of literals. A clause *covers* a truth assignment to the variables in ξ if the truth assignment makes the clause true. The resolution procedure shows ξ to be a tautology by demonstrating that every truth assignment is covered by some clause in ξ. The procedure starts with the original set of clauses in ξ and repeatedly generates new clauses from existing ones. Each new clause is derived from two previously existing clauses and the new one covers only truth assignments

that are already covered by one of the two clauses from which it is derived. The procedure is successful if the empty clause (which covers all truth assignments) is finally generated. Two clauses α and β can be resolved to get a new clause γ if and only if there is exactly one variable x such that the literal x is in one of α or β and the literal x' is in the other clause. The *resolvent* γ is then defined to be the conjunction of all the literals other than x and x' that are contained in α or in β. For example, if α is $uvw'xz'$ and β is $vx'y'z'$, then γ is $uvw'y'z'$. We call this step *eliminating* x from α with β. The clause $ab'c$ cannot be resolved with $a'bc$, nor with the clause cde'.

1.2. Complexity of resolution

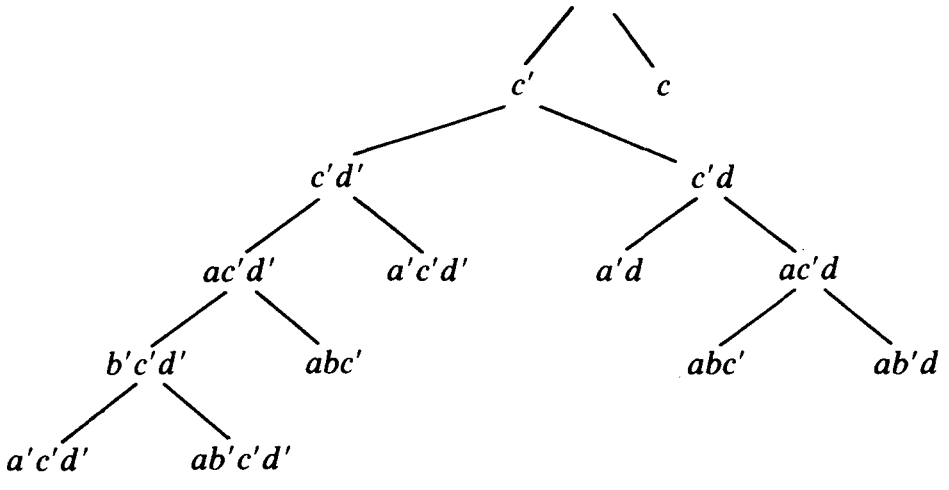
The *complexity* of a resolution proof is taken to be the number of different clauses generated in the course of the proof. We show that, for a certain class of tautologies, there is no polynomial p so that, for every member ξ of the class, the complexity of the shortest resolution proof of ξ can be bounded by $p(c(\xi))$, where $c(\xi)$ is the number of clauses in ξ. Our definition of complexity is adequate for distinguishing between polynomial and non-polynomial lengths of proofs, even if the lengths are counted in terms of how many characters are needed to write out the proofs and formulas. If a DNF tautology ξ is written using N characters, all the variables in any clause appearing in the proof can be written out using less than N characters, so any clause can be written out using at most a constant times N characters.

The problem of recognizing tautologies is equivalent to recognizing those formulas whose negation is unsatisfiable. Since the satisfiability problem is an NP-complete problem, the set of propositional tautologies is co-NP-complete. If resolution could always give proofs that are bounded polynomially in length, then co-NP would equal NP. That is considered to be unlikely by most people who study the P vs. NP problem. Nevertheless, resolution has not until now been shown to be nonpolynomial. (See [6] for a thorough discussion of the P vs. NP problem.)

1.3. Proof trees

We visualize resolution proofs as binary trees with the nodes labeled by clauses. The root node has the empty clause as a label. The immediate descendants of a node are labeled with the two clauses that the procedure resolved to generate the clause that labels the node. The leaves of the tree are labeled with the original clauses of the tautology whose proof is being represented. Note that many nodes in the tree may be labeled with the same clause, since a clause can be resolved with many other clauses.

Example. A proof tree for $abc' + ab'd + ab'c'd' + a'd + a'c'd' + c$ is given as follows:



1.4. Logical completeness of resolution

The resolution procedure is a nondeterministic generalization of the deterministic Davis–Putnam procedure (DPP) [4]. We show that resolution is logically complete by showing that DPP is logically complete. The DPP resolves clauses in a specified order. To start the DPP, one variable x is chosen and each clause containing the literal x is resolved with each clause containing the literal x', provided x is the only variable at which the two clauses disagree. The clauses that contain x or x' are then discarded. Next, another variable is chosen and that variable is eliminated from all clauses that contain it. The empty clause is ultimately generated if and only if the formula was a tautology. The order in which variables are eliminated can be specified according to which variable is contained in the shortest clause or which variable will produce the fewest new clauses when eliminated.

Lemma 1.1 (Davis and Putnam [4]). *Resolution is logically complete.*

Proof. We show that, using the DPP order of resolving clauses, every tautology has a resolution proof. Let ξ be a propositional DNF tautology with m different variables $v_1, \ldots, v_m$ which are eliminated in that order. Let S_0 be the set of clauses in ξ, and let S_i be the set of clauses that remains after v_i is eliminated, for $i = 1, \ldots, m$. The empty clause is the only clause that S_m might contain. We use induction to show that, for each $i = 1, \ldots, m$, every truth assignment to $v_1, \ldots, v_m$ is covered by some clause from S_i. Therefore, S_m must contain the empty clause. Since ξ is a tautology, all truth assignments are covered by clauses in S_0. Suppose that all truth assignments are covered by clauses in S_k, we show that all truth assignments are also covered by clauses in S_{k+1}. Suppose V is a truth assignment covered by clause α in S_k. The set S_{k+1} is the result of eliminating v_{k+1} from S_k. If α does not contain v_{k+1} or v'_{k+1}, then α is also in S_{k+1}, so a clause from S_{k+1} covers V. Otherwise, let W be the truth assignment that is just like V except it assigns the opposite value to v_{k+1}. Let β be

a clause in S_k that covers W. Either β does not contain v_{k+1} or v'_{k+1}, in which case β is in S_{k+1} and β covers V, or the resolvent of α and β is in S_{k+1} and covers V (also W). In any case there is a clause in S_{k+1} that covers V. □

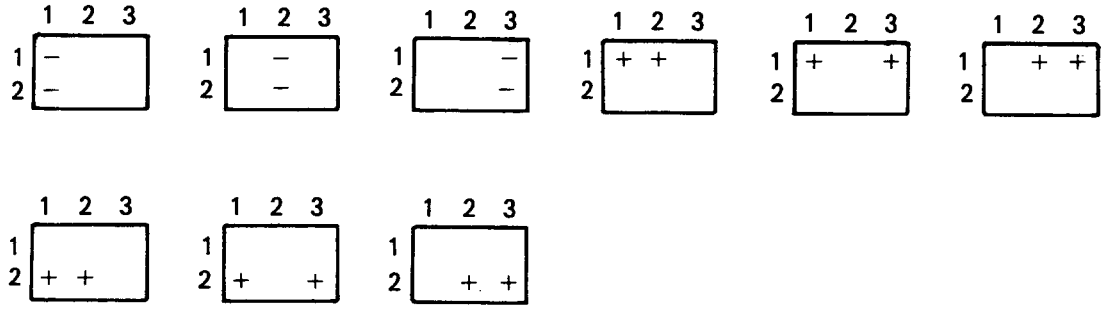
1.5. Pigeonhole formulas

In our proof that resolution has nonpolynomial time complexity, we use a class of tautologies which we call *pigeonhole formulas.* These formulas were defined by Cook and Reckhow [3] who used them to illustrate the efficiency of adding variables as abbreviations for subformulas. We index the pigeonhole formulas as PF_n, where n is simply a convenient quantity related to PF_n. (PF_n encodes the principle that if n pigeons sit in $n+1$ holes, there must be an empty hole.)

For the representation of truth assignments and clauses relating to PF_n it is convenient to refer to variables in PF_n according to their position in (n by $n+1$)-size arrays instead of using double subscripts. We represent a clause by an array as follows: Let x be a variable with an assigned array position. If a clause contains the literal x', then the array representing the clause contains a '$-$' in the position for x. If the clause contains the literal x, the array contains a '$+$' in the x position. Otherwise, the position for x remains blank in the clause array. A truth assignment is represented in a similar way by an array full of '0's and '1's, using '0' for variables assigned false and '1' for variables assigned true. A clause covers a truth assignment if and only if each '$+$' in the clause corresponds to a '1' in the truth assignment, and each '$-$' in the clause corresponds to a '0' in the truth assignment. Blanks in the clause array may correspond to either '0' or '1' in the array of a truth assignment covered by the clause.

We define PF_n in terms of arrays: The formula PF_n is the disjunction of all possible clauses given by n by $n+1$ arrays with exactly one whole column of '$-$'s and blanks everywhere else plus the clauses given by arrays with exactly two '$+$'s, both in the same row, and blanks everywhere else. There are $n+1$ arrays of the first type and $\frac{1}{2}(n^3+n^2)$ arrays of the second type.

Example. PF_2 in array notation is given as follows:



PF_n has n^2+n variables and contains $n+1+\frac{1}{2}(n^3+n^2)$ clauses. For each $n>0$, let comp(PF_n) be the number of clauses generated by the least complex resolution

proof of PF_n. We show that the function comp(PF_n) is exponential in n. Therefore, comp(PF_n) is not bounded by any polynomial of the number of clauses in PF_n.

Lemma 1.2 (Cook and Reckhow [3]). *For all n, PF_n is a tautology.*

Proof. If a truth assignment V for the variables of PF_n has two '1's in some row, then V is covered by a clause with '+'s in those two positions. If V has at most one '1' in each row, then the array for V contains at most n '1's, so one of the $n+1$ columns has only '0's. Thus, V is covered by a clause with '−'s in that column. Therefore, PF_n covers all truth assignments. □

2.Theorem and proof

2.1. Statement of the Theorem, beginning of the proof

Theorem. *There exists a constant c, $c>1$, so that, for sufficiently large n, every resolution proof of PF_n contains at least c^n different clauses.*

Proof. Suppose we have a resolution proof R of PF_n for some particular n. For the whole proof, let k be $\lfloor \frac{1}{4}n \rfloor$. We start with a series of definitions and lemmas.

Let FS1 consist of all sets of k variables from PF_n such that no two variables in the set are on the same row or in the same column (using the array representation). (FS1 stands for 'fixed sets of '1's' for reasons to be seen later.) Let $h(n)$ be the number of elements of FS1. For each set S in FS1 we shall find a corresponding 'highly complex clause' (hcc) in the proof tree of R. We define a function $g(n)$ and show that each hcc can correspond to at most $g(n)$ different members of FS1. Therefore, the proof R contains at least $f(n)=h(n)/g(n)$ different clauses. This function f is shown to be exponential.

2.2. Critical truth assignments

We define CTA, the set of *critical truth assignments* (ctas), as follows: A truth assignment is critical if it has exactly one '1' in every row and exactly one '1' in each column, except for one column which we call the 0-*column* (which consists entirely of '0's). There are $(n+1)!$ ctas. Each cta is covered by only one of the original clauses, the one with '−'s in the cta's 0-column. If V is a cta with a '1' in row r at column c, then we say row r *V-corresponds* to column c and column c V-corresponds to row r. Each row V-corresponds to some column and each column, except the 0-column, V-corresponds to some row.

2.3. Half zero clauses

Define a *half zero clause* (hzc) to be any clause in the proof R that has at least one column with exactly $2k$ '−'s and $n-2k$ blanks. A hzc is said to *present* a cta

if the hzc covers the cta and the 0-column of the cta is a column with exactly $2k$ '$-$'s in the hzc. Note that if α presents V, then α has at most one '$+$' per row or column and no '$+$' in the 0-column of V. We call the set of hzcs HZC.

Lemma 2.1. *Every cta is presented by at least one hzc.*

Proof. Let V be a cta and let p be the 0-column of V. The cta V is covered by the root clause of the tree for proof R. Whenever V is covered by a clause γ, where γ is the resolvent of clauses α and β, one of α or β covers V also. Thus, we can follow a path down the tree to a leaf so that every clause along the path covers V. The leaf clause at the end of the path must have column p full of '$-$'s, since that is the only kind of original clause that covers V. As we move up the path from the leaf to the root, at each step one variable is eliminated by the resolution action and possibly some literals are added to the clause on the path by the other immediate descendant. So, at each step on the path at most one '$-$' disappears from column p and some '$-$'s might be added to column p. Column p remains free of '$+$'s for all clauses on the path, since all clauses on the path cover V and p is the 0-column of V. Thus, the number of '$-$'s in column p starts at n and ends at 0 and never goes down by more than one. At some point the number of '$-$'s in column p is exactly $2k$. That clause is a hzc and presents V. □

Lemma 2.2. *If a clause α, present in proof R, covers a cta V with 0-column p, then either α has at least $2k$ '$-$'s in column p, or there is a hzc β among the descendants of α in the proof tree such that β presents V.*

Proof. Suppose α has fewer than $2k$ '$-$'s in column p. The argument is essentially the same as the argument proving Lemma 2.1. There is a path from α to a leaf that covers V such that every clause on the path covers V. A node at which the number of '$-$'s in column p becomes $2k$ is the β we want. □

2.4. Neighboring ctas

Let V and W be two ctas and let p be the 0-column of V, q the 0-column of W. We call V and W *neighbors* if W is the result of switching columns p and q in V. We say W is the *q-neighbor* of V and V is the p-neighbor of W.

Lemma 2.3. *Let a hzc α present a cta V with 0-column p. Let q be a column different from p, and let r be the row that V-corresponds to column q. Suppose α does not have a '$-$' in row r at column p and α does not have a '$+$' in row r at column q. Then α covers the q-neighbor of V.*

Proof. Let W be the q-neighbor of V. The truth assignments V and W differ only at row r, columns p and q. Since $\alpha_{rp} \neq$ '$-$', $\alpha_{rq} \neq$ '$+$', $V_{rp} =$ '0', $V_{rq} =$ '1' and α covers

V, we must have α_{rp} = blank and α_{rq} = blank. Thus, α has blanks wherever W differs from V. Since α covers V, it follows that α covers W. □

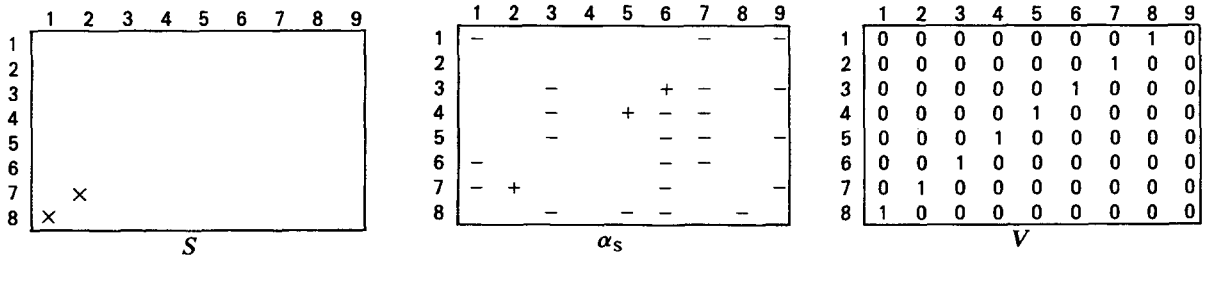
2.5. Highly complex clauses

We now define HCC, the set of hccs (*highly complex clauses*), as a subset of HZC. Recall that FS1 is the set of all sets S of k variables such that no two variables in S are in the same row or column. Now, we assign a certain hzc α_S to each S in FS1. First, define S-CTA to be the set of ctas that assign '1' to each variable in S. (The name FS1 comes from the fact that each member S of FS1 defines a fixed set of '1's for the ctas in S-CTA.) Next, let S-HZC be the set of all hzcs α such that α presents at least one member of S-CTA. By Lemma 2.1, the set S-HZC is nonempty. The resolution procedure generates new clauses one by one, so, let α_S be the first member of S-HZC generated by the resolution procedure in the proof R. Note that, in the proof tree, no S-cta is presented by any descendant of α_S. HCC is the set of clauses α_S for S in FS1.

Lemma 2.4. *Each hcc has at least $k+1$ columns c such that column c contains a '+' or at least $2k$ '−'s.*

Proof. Let S be in FS1 and α_S be the appropriate hcc. Let V be an S-cta that is presented by α_S, and let p be the 0-column of V. At least k of the rows in α_S have no '−' in column p and contain no variable from S: Of the n rows in α_S, $2k$ rows contain '−' in column p and k rows contain variables from S. That leaves at least $n-3k$ rows, or at least k rows, since $n \geq 4k$. Each such row with no '−' in column p and no variables from S V-corresponds to a column. Call this set of columns GC (good columns). Let column q be from GC and let row r be the row that V-corresponds to q. Either α_S has a '+' in column q, or, by Lemma 2.3, α_S covers the q-neighbor of V. Since V is an S-cta, any variable in S must have a '1' in V. Therefore, column p has no element of S. Column q also has no element of S, because row r was chosen to contain no variable from S. So, the q-neighbor of V is also an S-cta. Thus, the q-neighbor is not presented by any descendant of α_S. By Lemma 2.2, column q must contain at least $2k$ '−'s in α_S. Column p together with GC supplies the promised $k+1$ columns. □

We illustrate Lemma 2.4 in the following diagram ($n=8$, $p=9$, GC = {3, 5, 7}):



2.6. *The function $f(n)$*

Each of the $h(n)$ members of FS1 supplies a hcc, but this mapping need not be one to one. To get a lower bound on the number of different hccs we may divide $h(n)$ by an upper bound $g(n)$ on the number of sets in FS1 that could possibly supply a given hcc. Let α be in HCC. If α is α_S for a particular S in FS1, then α has '+'s or blanks in the positions for variables from S. That is true because α_S covers an S-cta. Also, α_S has at most one '+' per column. We use the complexity of α to bound the number of ways one could choose S in FS1 such that α could be α_S. To get $g(n)$ we count the number of ways one could assign k '1's to the variables of PF_n, so that no two variables are in the same row or column, there is no conflict with '−'s in α, and if a column of α contains a '+', that is the only place to assign a '1' in that column.

Let A be a set of $k+1$ columns that have a '+' or at least $2k$ '−'s in α. We can find A by Lemma 2.4. For each column in A, there are at most $n-2k$ places where we can assign a '1'. Whenever we assign k '1's, some number i of them is in the $k+1$ columns from A, and the remaining $k-i$ of them are in the $n-k$ columns outside of A. The number i can be any integer from 0 to k. Given i, there are d_i ways to choose the columns with '1's, where $d_i = \binom{k+1}{i}\binom{n-k}{k-i}$. For the i columns from A we get at most $(n-2k)^i$ ways to choose the rows of the '1's in those columns. In the $k-i$ columns from outside of A we get at most $(n-i)!/(n-k)!$ choices for the rows for the '1's, since each row can only be used once. So,

$$g(n) = \sum_{i=0}^{k} d_i (n-2k)^i \frac{(n-i)!}{(n-k)!}.$$

We find $h(n)$, the size of FS1, in the same way, except we have more freedom to choose the rows for variables that are in columns from A:

$$h(n) = \sum_{i=0}^{k} d_i \frac{n!}{(n-i)!} \frac{(n-i)!}{(n-k)!}.$$

(Note that $h(n)$ can also be written as $\binom{n+1}{k}\, n!/(n-k)!$.)

The function $f(n)$ is $h(n)/g(n)$ and $\mathrm{comp}(PF_n) \geqslant f(n)$.

2.7. *The function $f(n)$ is exponential, conclusion of the proof*

Now let $n \geqslant 200$.

$$f(n) = \frac{h(n)}{g(n)} = \sum_{i=0}^{k} d_i \Big/ \sum_{i=0}^{k} d_i \frac{(n-2k)^i (n-i)!}{n!}$$

$$> \sum_{i=0}^{k} d_i \Big/ \sum_{i=0}^{k} \frac{d_i}{(1.49)^i},$$

since $(n-2k)/(n-i) < 1/1.49$ (for $i \leqslant k$ and $n > 40$). Let $m = \lfloor \frac{1}{50} n \rfloor$.

We continue with the chain of inequalities:

$$f(n) > \sum_{i=m}^{k} d_i \Big/ \left(2 \sum_{i=m}^{k} \frac{d_i}{(1.49)^i}\right) > (1.49)^m/2 > (1.49^{0.01})^n \quad (n \geq 200),$$

because:

$$\sum_{i=0}^{m-1} \frac{d_i}{(1.49)^i} < \sum_{i=m}^{2m-1} \frac{d_i}{(1.49)^i},$$

since

$$\frac{d_i}{(1.49)^i} < \frac{d_{i+1}}{(1.49)^{i+1}} \quad \text{for } i+1 \leq \tfrac{1}{25}n$$

$$\left(\frac{d_{i+1}}{d_i} = \frac{(k+1-i)(k-i)}{(i+1)(n-2k+i+1)} > \frac{(n/5)(n/5)}{(n/25)(0.6n)} > 1.5\right).$$

So, $f(n) > c^n$ with $c = 1.49^{0.01}$ for $n > 200$. □

3. Extensions

3.1. Extended resolution and regular resolution

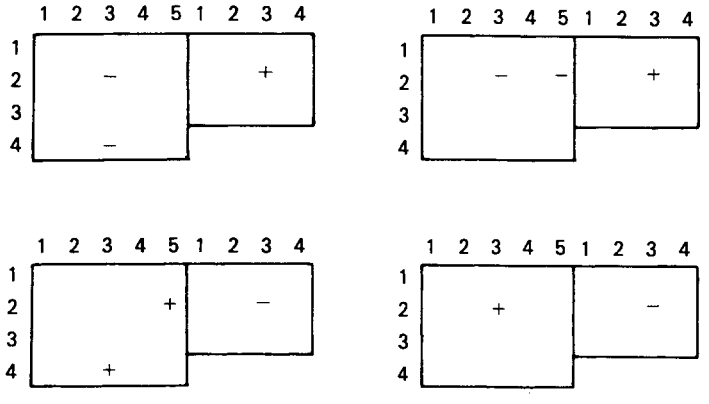
Regular resolution is resolution with the following restriction: if clause α contains variable v and clause β does not contain v, but some descendant of β in the proof tree does contain v, then α and β may not be resolved. If one follows a branch in the proof tree, a variable may not be eliminated and then later be contained again in a clause on the branch. Tseitin [9] has shown that regular resolution is nonpolynomial (see also [5, 7]). Tseitin also invented extended resolution and found that extended resolution would yield short proofs for his examples. It turns out that extended resolution also yields proofs of PF_n with complexity on the order of n^4.

In *extended resolution* we can add variables that abbreviate propositional formulas on the variables we already have. For instance, if we are proving θ and the variable x does not appear in θ, then we can add clauses that are logically equivalent to the negation of $x \equiv \lambda(u, v, w, y, z)$ where λ is a logical function of five variables and u, v, w, y, and z appear in θ. Such an extension by adding a definition does not change the status of θ as a tautology or a nontautology.

The pigeonhole formulas are given by Cook and Reekhow as an example of how the introduction of definitions can shorten a proof in a proof system with slightly different syntax. We describe how extended resolution can produce a polynomial length proof of PF_n (following Cook's and Reckhow's argument [3]).

The basic idea is to use induction: reduce PF_i to PF_{i-1} while only generating polynomial complexity. When this reduction is carried out $n-1$ times, the only thing left is to prove PF_1 which takes four clauses. We first go from PF_n to PF_{n-1}: Let the variables in PF_n be $y_{i,j}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, n+1$. Introduce variables $x_{i,j}$ for $i = 1, \ldots, n-1$ and $j = 1, \ldots, n$. The definition of $x_{i,j}$ is $y_{i,j} + y_{i,n+1}y_{n,j}$. In terms of adding clauses that represent the negation of that definition we add:

$x_{i,j}y'_{i,j}y'_{n,j}$ and $x_{i,j}y'_{i,j}y'_{i,n+1}$ and $x'_{i,j}y_{n,j}y_{i,n+1}$ and $x'_{i,j}y_{i,j}$. In array notation for $x_{2,3}$ we add $(n=4)$:



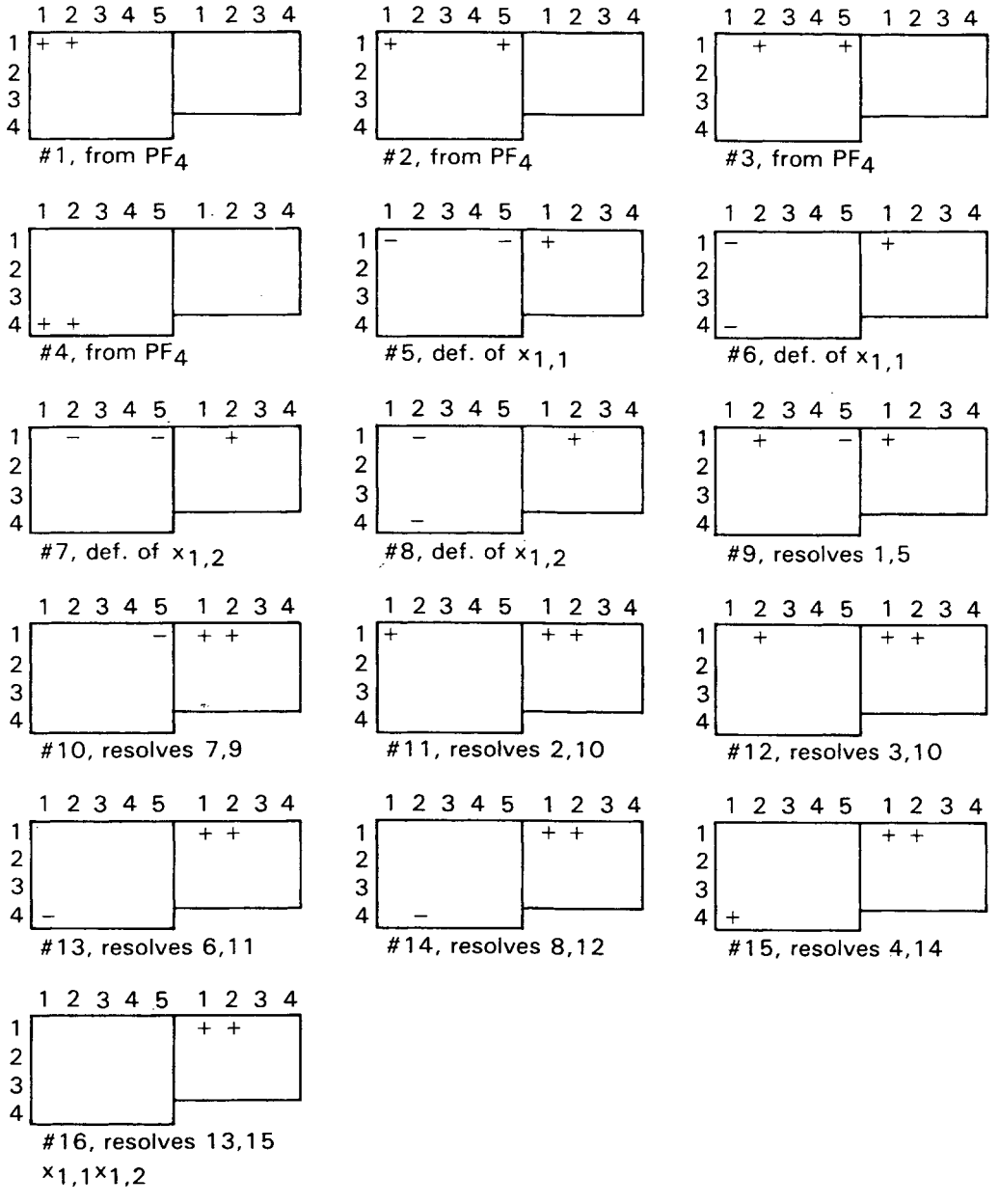


Fig. 1. Generation of $x_{1,1}x_{1,2}$ using extended resolution.

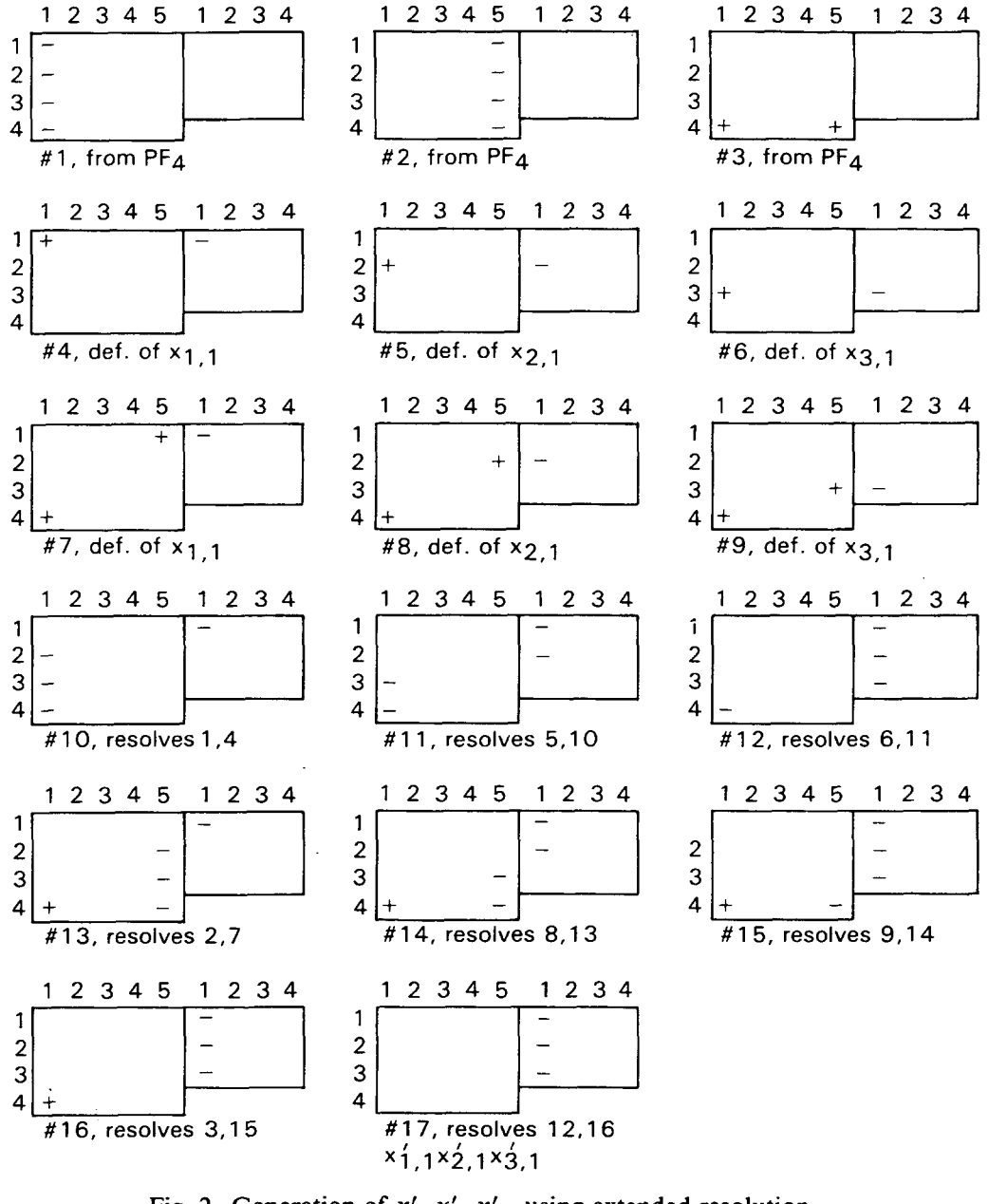


Fig. 2. Generation of $x'_{1,1}x'_{2,1}x'_{3,1}$ using extended resolution.

Each of the original 'two '+'s in a row' clauses for PF_{n-1} can be generated together with only seven other new clauses by resolution. (See Fig. 1 for an example with $n=4$.) Each of the original 'column of '−'s' clauses for PF_{n-1} can be generated together with $2(n-1)+1$ other new clauses using resolution. (See Fig. 2 for an example with $n=4$.) To prove the instance of PF_{n-1}, more new variables are introduced so that the problem can be reduced to PF_{n-2}, etc. The whole process needs only on the order of n^4 new clauses.

3.2. Further related research

The most obvious further questions have to do with extended resolution. One goal is to prove that extended resolution is also nonpolynomial. The task seems very difficult since using the extension rule we are able to simulate meta-arguments

that a given formula is a tautology. Another goal would be to prove that extended resolution or some extended Frege system as defined in [3] can polynomially simulate a Turing machine that recognizes tautologies. This task also seems very difficult. Possibly, the question of the complexity of extended resolution will only be solved when the NP vs. co-NP problem is resolved. Finally, it may be possible to show that Frege systems without the extension rule also generate nonpolynomial complexity on the pigeonhole formulas.

Acknowledgment

This work is part of the requirements for the degree of Ph.D. in Mathematics at the University of Illinois in Urbana-Champaign. The author wishes to thank his thesis advisor Prof. Gaisi Takeuti and his thesis committee for their help. The author thanks the referees for their helpful comments and particularly for suggesting the much shorter argument in Section 2.7. Also, the author thanks the University of Illinois and the National Science Foundation for fellowship support.

References

[1] A. Barr, P. Cohen and E.A. Feigenbaum, *Handbook of Artificial Intelligence, Vol. III* (Kaufmann, Los Altos, CA, 1983) Chapter 7.

[2] S.A. Cook and R.A. Reckhow, On the lengths of proofs in the propositional calculus, Preliminary version, *Proc. 6th Ann. ACM Symp. on the Theory of Computing* (1974) 135–148. Final version in: *ACM Sigact News* **6** (1974) 15–22.

[3] S.A. Cook and R.A. Reckhow, The relative efficiency of propositional proof systems, *J. Symbolic Logic* **44**(1) (1979) 36–50.

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

[5] Z. Galil, On the complexity of regular resolution and the Davis–Putnam procedure, *Theoret. Comput. Sci.* **4**(1) (1977) 23–46.

[6] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, CA, 1979).

[7] K. Justen, A note on regular resolution, *Computing* **26**(1) (1981) 87–89.

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

[9] G.S. Tseitin, On the complexity of derivations in propositional calculus, in: A.O. Slisenko, ed., *Studies in Mathematics and Mathematical Logic, Part II* (Consultants Bureau, New York/London, 1970) 115–125.