ANGEWANDTE MATHEMATIK UND
INFORMATIK
UNIVERSITÄT ZU KÖLN

Report No. 95–189

The Complexity of the Falsifiability Problem for Pure Implicational Formulas

by
P. Heusch

1995

Accepted for MFCS'95

Institut für Informatik
Universität zu Köln
Pohligstr. 1
D–50969 Köln

The Complexity of the Falsifiability Problem for Pure Implicational Formulas

Peter Heusch[1]

Abstract

Since it is unlikely that any NP-complete problem will ever be efficiently solvable, one is interested in identifying those special cases that can be solved in polynomial time. We deal with the special case of Boolean formulas where the logical implication $\rightarrow$ is the only operator and any variable (except one) occurs at most twice. For these formulas we show that an infinite hierarchy $S_1 \subseteq S_2 \cdots$ exists such that we can test any formula from S_i for falsifiability in time $O(n^i)$, where n is the number of variables in the formula. We describe an algorithm that finds a falsifying assignment, if one exists. Furthermore we show that the falsifiability problem for $\bigcup_{i=1}^{\infty} S_i$ is NP-complete by reducing the SAT-Problem. In contrast to the hierarchy described by Gallo and Scutella for Boolean formulas in CNF, where the test for membership in the k-th level of the hierarchy needs time $O(n^k)$, our hierarchy permits a linear time membership test. Finally we show that S_1 is neither a sub- nor a superset of some commonly known classes of Boolean formulas, for which the SAT-Problem has linear time complexity (Horn formulas, 2-SAT, nested satisfiability).

Subject classification: algorithms and data structures, logic in computer science.

1 Introduction

The satisfiability problem (SAT) for Boolean formulas in conjunctive normal form (CNF) was the first problem that was shown to be NP–complete, [1]. For this reason, its complexity has been the subject of quite a number of studies. However, CNF–SAT shows a sort of threshold behaviour, yielding the effect that for many input restrictions for which the problem is solvable in polynomial time the problem becomes NP–complete even if the set of inputs is only slightly extended.

[1] Universität zu Köln, Pohligstr.1, D–50969 Köln, email: heusch@informatik.uni-koeln.de, Fax: (02 21) 4 70 - 53 87

One example of such a class is the class of CNF–formulas where every variable may occur at most twice, for inputs from this class the SAT–problem is solvable in linear time, if however three occurrences of a variable are allowed in the input formulas, the satisfiability problem is NP–complete. There are also classes C_i of formulas where for any $F \in C_i$ the satisfiability problem is solvable in time $O(|F|^i)$, $|F|$ denoting the number of variables in the formula, for example the classes C_i where every formula in C_i is satisfiable by setting at most i variables to true, but this classification is quite unsatisfactory in the sense that the test whether a given F belongs to some class C_k may need up to $O(|F|^k)$ steps. Important classes showing this behaviour are the classes Γ_i defined by Gallo and Scutella, [2].

We will present a new hierarchy $S_1 \subseteq S_2 \subseteq \cdots$ with the property that for every $F \in S_i$ the falsifiability can be solved in time $O(|F|^i)$, while the test whether $F \in S_i$ can be solved in linear time. Furthermore, we will prove that every SAT–problem is polynomially reducible to some problem in $\bigcup_i S_i$, hence the falsifiability problem for $\bigcup_i S_i$ is NP–complete.

The remaining part of this paper is organized in the following way: the rest of this chapter contains the definitions needed, in chapter 2 we prepare our main result while chapter 3 contains the main result. In the last chapter we give a relationship between the class of formulas solvable in linear time by our algorithm and other classes for which satisfiability is solvable in linear time.

A Boolean formula $F = C_1 \wedge C_2 \wedge \ldots \wedge C_r$ in conjunctive normal form (CNF) over n Variables $v_1, \ldots, v_n$ is a conjunction of clauses $C_1, \ldots, C_r$, where each clause C_l is a disjunction of literals $x_{i_1}, \ldots, x_{i_k}$, a literal is either stands for a variable (positive literals) or its complement (negative literals). A Boolean formula is in pure implicational form (PIF), iff it contains only positive literals and the only connective being used is the logical implication. For any implication $A \rightarrow B$ we call A the implicant and B the consequence of the implication. Since the implication is a nonassociative connective, we define $A \rightarrow B \rightarrow C$ to be read as $A \rightarrow (B \rightarrow C)$. An assignment $t : \{v_1, \ldots, v_n\} \mapsto \{true, false\}$ satisfies a Boolean formula F, iff F evaluates to $true$ when every variable v is replaced by $t(v)$ and the usual evaluation rules for Boolean operators are applied, t falsifies F iff F evaluates to $false$. A partial assignment is a function $t : \{v_1, \ldots, v_n\} \mapsto \{true, false, undef\}$, a

(partial) assignment t' extends a partial assignment t, iff

$$t(v) \neq undef \Rightarrow t'(v) = t(v).$$

An assignment t' is called 1–extension of a partial assignment t, if t' extends t and $t(v) = undef$ implies $t'(v) = true$.

For any Boolean formula $F = F_1 \rightarrow F_2$ and for any subformula F' of F we define

$$\mathcal{D}_l(F', F) = \begin{cases} 0 & \text{if } F = F', \\ 1 + \mathcal{D}_l(F', F_1) & \text{if } F' \text{ lies in the implicant of } F, \\ \mathcal{D}_l(F', F_2) & \text{if } F' \text{ lies in the consequence of } F. \end{cases}$$

If F is represented by a tree then $D_l(F', F)$ denotes the number of left edges we have to pass on the path from the root of F to the root of F'. The set

$$\mathcal{B}(F) = \{F' | F' \text{ is subformula of } F, D_l(F', F) = 0\}$$

is called the backbone of F, those subformulas F' of F that have $D_l(F', F) = 1$ are called the backbone implicants of F. The backbone of F contains exactly one subformula that is a variable, this variable is the rightmost variable $V_r(F)$. The set of Boolean formulas in PIF where every variable except the rightmost variable occurs at most twice is called 2–PIF. A backbone implicant F' of F is called a critical subformula, w.r.t a partial assignment t, iff $t(V_r(F')) = false$. We will see that the number of critical subformulas plays an important role in the analysis of the falsifying algorithm. If a subformula F' of F is critical and F'' is a backbone implicant of F', we call F'' compensating (w.r.t a partial assignment t), if t falsifies F''. This is due to the fact that a formula F in PIF can be satisfied by setting $V_r(F)$ to $true$ or by falsifying at least one of the backbone implicants, hence to falsify F, $V_r(F)$ must be set to $false$ and all backbone implicants have to be satisfied.

2 A hierarchy for pure implicational formulas

We will now define the formula subsets that subdivide 2–PIF and prove some results about them as well as about 2–PIF itself. We define S_i to contain all those formulas F in 2–PIF, such that $V_r(F)$ occurs at most i times in F. The definition of these sets immediately implies the following lemma:

Lemma 1 *For any Boolean formula F in 2–PIF, the membership problem whether F belongs to S_i can be determined in linear time.*

Proof Obvious.□

Another interesting point that a hierarchy must fulfill to be interesting is that it must also be a real hierarchy, i.e. that it must not collapse beyond a certain class, as in the case of CNF–SAT, where an increase of the number k of literals allowed in one clause does not change the complexity of the problem if $k \geq 3$. The following theorem based on a theorem by Kleine B"uning et al. given in [3], however, gives a strong hint that this is indeed the case with the hierarchy induced by the S_i:

Theorem 1 *The falsifiability problem for formulas in 2–PIF is NP–complete.*

Proof We reduce the wellknown NP–complete SAT–problem for Boolean formulas in CNF where every variable occurs at most 3 times to the falsifiability problem for Boolean formulas in PIF. Let F be such a formula in CNF. W.l.o.g. we may assume that every variable with 3 occurrences occurs exact once positive and twice negative in F. Let a be such a variable and let C_1, C_2 be the clauses such that $C_1 = \neg a \vee C_1'$ and $C_2 = \neg a \vee C_2'$. We then introduce new variables a', a'' and replace C_1, C_2 by $\neg a \vee (a' \wedge a'')$, $\neg a' \vee C_1'$ and $\neg a'' \vee C_2'$. By repeating this process for every variable occurring three times in F we get a new formula F' s.t. every variable is contained at most twice in F'.

The next step is to eliminate the logical operations $\wedge$, $\vee$ and $\neg$. Without changing the number of variables this can be achieved by substitution of $a \rightarrow false$ for $\neg a$, $(a \rightarrow false) \rightarrow b$ for $a \vee b$ and $(a \rightarrow (b \rightarrow false)) \rightarrow false$ for $a \wedge b$. At this point we may apply some simplification rules, e.g. substituting a for $a \rightarrow false \rightarrow false$. To eliminate the logical constant $false$, we replace every occurrence of $false$ by a new variable z, which will forced to be set to $false$ later on.

Let F'' be the result of these transformations. Clearly, F'' contains every variable at most twice and is satisfiable by every assignment that satisfies the original formula F and sets z to $false$, thereby setting those "variables" to $false$, where z was replaced for the constant value $false$. This immediately results in the formula $F'' \rightarrow z$ being falsifiable iff F was satisfiable, hence the falsifiability problem for Boolean formulas in 2–PIF is NP–complete.□

3 Main Theorem

Theorem 1 showed that every NP–complete problem must be contained in one of the set S_i. In the next step we show that our hierarchy is indeed a polynomial hierarchy, i.e. that the falsifiability problem is polynomially solvable for every fixed S_i.

We will formulate this by proving the runtime bound and the correctness for the following algorithm PIF_solve, initially called with the parameters F and $\emptyset$.

```
procedure PIF_solve(F:PIF,Z:set);

begin
  let F = F_1 → ...F_j → ... → F_k → z
  Z = Z ∪ {z}
  if Z ∩ ⋃_i V_r(F_i) = ∅ then begin
    print solution Z and exit
  end
  find the smallest j such that V_r(F_j) ∈ Z
  let F_j = G_1 → ...G_h → z'
  for l=1 to h do begin
    PIF_solve(F_1 → ...F_{j-1} → F_{j+1} → ... → G_l,Z)
  end
end
```

This can also be seen as a graph manipulation process: if the formula is interpreted as a tree where the inner nodes correspond to operators and the outer nodes correspond to variables, then we can falsify the formula from figure 1 iff for at least one i the formula given in figure 2 is falsifiable.

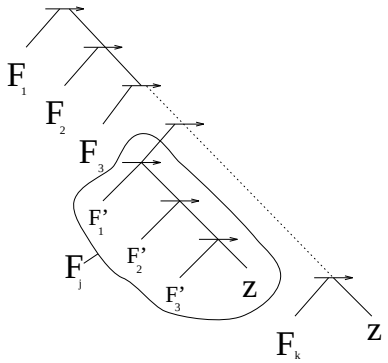

Figure 1

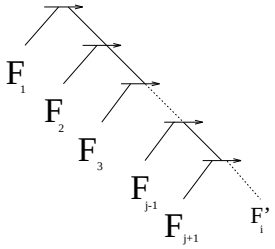

Figure 2

As we mentioned, the number of critical subformulas plays an important role for the runtime of the algorithm. The following lemma gives an upper bound for this number:

Lemma 2 *Let F be a formula in 2–PIF containing i occurrences of $V_r(F)$. Then the number of critical formulas in any recursive call of PIF_solve is bounded above by $i-1$.*

Proof The proposition is surely true when PIF_solve starts, since there may be at most $i-1$ backbone implicants whose rightmost variable is $z_0 := V_r(F)$. If, however, we see the process of finding critical and compensating subformulas as a variable renaming process—instead of storing those variables in Z that we have decided to be assigned the value $false$, we could simply rename them to z_0—then we see that the number of occurrences of z_0 can never go beyond i, since whenever two occurrences of z_0 are put into F by selecting a compensating subformula, we delete two other occurrences of z_0 in the formula.□

It remains to prove the correctness of PIF_solve. This is expressed in the following lemma:

Lemma 3 *Let F be a formula in 2–PIF. Then F is falsifiable iff one of the following conditions holds:*

1. *F doesn't contain any critical subformulas.*
2. *An assignment exists such that for every critical subformula F' of F there is a compensating subformula F''.*

Proof If F does not contain any critical subformulas, then it can easily be falsified by assigning $false$ to $V_r(F)$ and assigning $true$ to all other variables, this satisfies all backbone implicants, hence F is falsified. Assume now that F contains critical subformulas. If we can extend the partial assignment implied by Z in such a way that for every critical subformula formula a compensating subformula exists, we get an assignment where for every critical subformula at least one of its backbone implicants is assigned the value $false$, i.e. the critical subformula is satisfied even if its rightmost variable was assigned the value $false$. If we cannot extend the partial assignment given by Z in such a way, then for every assignment at least one critical subformula F' exists such

that no backbone implicant of F' is compensating, hence every assignment to the variables of F will falsify F' and therefore satisfy F.□

We note that for any formula $F = F_1 \rightarrow F_{j-1} \rightarrow F_j \rightarrow F_{j+1} \cdots \rightarrow F_r \rightarrow z$ in 2–PIF the test whether a certain subformula F' is compensating for a critical subformula F_j is the same as the test whether after removal of F_j from F there exists an assignment that falsifies F and F' at the same time, this can also be achieved by testing $F = F_1 \rightarrow F_{j-1} \rightarrow F_{j+1} \cdots \rightarrow F_r \rightarrow F'$ for falsifiability under the condition that $z = false$.

Now we state our main result, whose correctness, after the preceeding lemmas is almost obvious:

Theorem 2 *Let F be a formula from 2–PIF with n variables and $i \geq 2$ occurrences of $V_r(F)$, then PIF_solve finds a falsifying assignment for F iff one exists, furthermore the runtime of PIF_solve is bounded by $O(in^{i-1})$.*

Proof Since there are no more than $i-1$ critical formulas in F at the same time, there are at most $\sum_{k=0}^{i-1} \binom{n}{k} = O(n^{i-1})$ ways to distribute them amongst all subformulas. To get a runtime bound, we note that at most $2i$ occurrences of variables in Z are contained in F, hence the test whether $Z \cap \bigcup_i V_r(F_i) = \emptyset$ and the selection of j such that $V_r(F_j) \in Z$ can be carried out in time $O(i)$, all other steps take constant time, hence we get a runtime of $O(in^{i-1})$. The correctness follows from the fact that if no critical subformula is found, then PIF_solve prints a solution, else it enumerates all possible ways to find a compensating subformula for F_j.□

4 Relationships to other input classes

Since CNF–SAT is NP–complete, a number of input restrictions have been developed that permit to test a formula for satisfiability in polynomial time. The most common of these restrictions are restrictions for formulas in CNF, they are defined as follows:

- 2–SAT: Clauses may contain at most 2 literals.
- Horn formulas: Clauses may contain at most 1 positive literal.
- Nested SAT: An ordering of the clauses must exist with the property that if a clause C preceeds another clause C' then no variable from C

except the first and the last (w.r.t to an ordering of the variables) may be contained in C'.

- READ–2: No variable may occur more than twice in a formula.

It is well known that for inputs from these classes the satisfiability problem is solvable in time proportional to the length of the formula, see [4, 3]. The following remark formalizes the relationship between S_2 and these classes:

Remark 1 *For any one of the classes 2–SAT, HORN, nested SAT and READ–2 there is a boolean function function whose complement can be expressed in S_2 but that cannot be expressed in 2–SAT, HORN, nested SAT and READ–2, respectively.*

References

[1] S. Cook. The Complexity of Theorem Proving Procedures. *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pages 151–158, 1971.

[2] G. Gallo and M.G. Scutella. Polynomially Solvable Satisfiability Problems. *Information Processing Letters*, 29(5):221–227, 1988.

[3] H. Kleine Büning and T. Lettman. *Aussagenlogik: Deduktion und Algorithmen*. B. G. Teubner, Stuttgart, 1994.

[4] D. E. Knuth. Nested satisfiability. *Acta Informatica*, 28:1–6, 1990.