Solving the Resolution-Free SAT Problem in Polynomial Time by Sub-Model Propagation*

Gábor Kusper
gkusper@risc.uni-linz.ac.at

Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, Linz, Austria
http://www.risc.uni-linz.ac.at/

January 18, 2001

Abstract

The SAT problem is the problem of finding a model for a clause set. It is well known that the SAT problem is **NP**-complete on general clause sets. We have developed a new method, called Unicorn-SAT algorithm, which solves the SAT problem in polynomial time on resolution-free clause sets. A clause set is resolution-free if and only if no resolution can be performed on any two clauses of the clause set. For such a restricted clause set, we can find a model in polynomial time by sub-model propagation. We obtain the sub-model, i.e. a part of the model, by Lucky-negation of that clause of the set which has the smallest number of literals. Lucky-negation of a clause is the negation of all literals of the clause except one. Sub-model propagation is unit propagation by each literal of the sub-model. We obtain a model by joining the sub-models while we perform sub-model propagation recursively until the clause set becomes empty.

* Sponsored by Upper Austrian Government and European project Calculemus RTN1-1999-00301.

1 Introduction

Propositional Satisfiability is the problem of determining, for a formula of the propostional calculus, if there is an assignment of truth values to its variables for which that formula evaluates the true. By SAT we mean the problem of propositional satisfiability for formulas in conjunctive normal form (CNF).

SAT is the first, and one of the simplest, of the many problems which have been shown to be **NP**-complete. It is dual of propositional theorem proving, and many practical **NP**-hard problems may be transformed efficiently to SAT. Thus, a good SAT algorithm would likely have considerable utility. It seems improbable that a polynomial time algorithm will be found for the general SAT problem but we know there are restricted SAT problems that are solvable in polynomial time. So a "good" SAT algorithm should check first the input SAT instance whether it is an instance of such a restricted SAT problem. In this paper we introduce the resolution-free SAT problem which is solvable in polynomial time.

The restriction of SAT to instances where all clauses have length k is denoted k-SAT. Of special interest are 2-SAT and 3-SAT: 3 is the smallest value of k for which k-SAT is **NP**-complete [1], while 2-SAT is solvable in linear time [2, 3]. Horn-SAT is the restriction to instances where each clause has at most one unnegated variable. Horn-SAT is solvable in linear time [4], as are a number of generalizations such as Re-nameable Horn-SAT [5] and Generalized Horn-SAT [6].

The resolution-free SAT problem is also a restriction of SAT to instances where no resolution can be performed on any two clauses of the clause set. This could be a real problem if we want to prove a theorem by resolution but we do not reach the empty clause after doing all possible resolutions, than we have a resolution-free clause set. In this case the Unicorn-SAT provides a model for this clause set in polynomial time, and this model contains essential information why the theorem does not hold.

The main part of this paper is Section 5, in which we introduce the Unicorn-SAT algorithm and we show it is correct and solves the resolution-free SAT problem in polynomial time. In Section 2 the notions are defined. In Section 3 we show an important property of Lucky-negation. In Section 4 we prove lemmas, that are needed to prove that the Unicorn-SAT is correct. In Section 6 we show an example how the algorithm works.

2 Definitions

Let V be a set of Boolean variables. The negation of a variable v is denoted by $\bar{v}$. Given a set U, we denote $\overline{U} = \{ \bar{u} \mid u \in U \}$. *Literals* are the members of the set $W = V \cup \overline{V}$. *Positive literals* are the members of the set V. *Negative literals* are their negations. If w denotes a negative literal $\bar{v}$, then $\overline{w}$ denotes the positive literal v. A *clause* is a finite set of literals that does not contain simultaneously any literal together with its negation. The empty clause, denoted by $\varnothing$, is interpreted as *False*. A *clause set* (*formula in CNF*) is a finite set of clauses. The empty clause set, denoted by $\varnothing$, is interpreted as *True*. The *length of a clause* C is its cardinality, denoted by $|C|$. The *length of a clause set* S is its cardinality, denoted by $|S|$. Clauses of a clause set with minimal length are called *minimal clauses* of this clause set. If we say that a *literal* v *occurs* in a clause or in a clause set, we mean that this clause or this clause set contains the literal v. However, if we say that a *variable* v *occurs* in a clause or in a clause set, we mean that this clause or this clause set contains the literal v, or it contains the literal $\bar{v}$. We denote by $V\{ S \}$ the *set of variables occurring in* the clause set S, and by $|V\{ S \}|$ its cardinality. We say that two clauses *differ in some variables* if these variables occur in both clauses but as different literals. We say that two clauses are *semi-equivalent* if and only if they differ in no variable. We say that *resolution can be performed on* two clauses if they differ only in one variable. A clause set is *resolution-free* if and only if no resolution can be performed on any two clauses of the clause set. *Lucky-negation* of a clause C, denoted by $L(C)$, is the negation of all literals of the clause except the first one (according to some variable ordering). Note that Lucky-negation could be defined in different ways, not necessary by negation of the first literal. An *assignment* or a *sub-model* is a finite subset of W that does not contain any literal together with its negation. Informally speaking, if an assignment I contains a literal v, it means that v has the value *True* in I.
Now we define the crucial notion of *sub-model propagation*.

Definition 2.1 (Sub-Model Propagation)

$S[I]$ (read: "sub-model propagation on clause set S by assignment I") :=

$$\{ C \setminus \bar{I} \mid C \in S \wedge \forall v \; v \in I \Rightarrow v \notin C \}.$$

$C[I]$ (read: "sub-model propagation on clause C by assignment I") :=

$C \setminus \bar{I}$, if C and $\bar{I}$ are semi-equivalent, undefined otherwise.

Note that if $\forall v \; v \in I \Rightarrow v \notin C$ then C and $\overline{I}$ are semi-equivalent. Informally $S[I]$ means for the clause set S and the sub-model $I = \{v_1,\ldots,v_n\}$, we

- remove from S all clauses containing the literals v_i, and
- we delete all occurrences of the literals $\overline{v}_i$ from the other clauses.

With other words sub-model propagation is equivalent to unit propagation [7] by each literal of the sub-model I. An assignment I is a *model* for a clause set S if $S[I]$ is the empty clause set. A clause set is *satisfiable* if there exists a model assignment for it.

3 Properties of Lucky-negation

In this section let C be a non-empty clause.

Lemma 3.1 (Trivial Properties of Lucky-negation)

L(C) is a clause, is an assignment, contains the first literal of C, contains the negation of all literals occurring in C except the first one, contains all variables of C.

At this point the reader should realize that in this paper clause and assignment are defined by the same definition. Thus, we can use L(C) either as a clause or an assignment.

Lemma 3.2

Resolution can be performed on C and L($\overline{C}$).

Proof:

From Lemma 3.1 we know that C and L(C) differ in every variable of C except the first. So C and L($\overline{C}$) differ only in the first variable of C.

Hence, resolution can be performed on C and L($\overline{C}$). ❑

Lemma 3.3 (The Lucky Property)

Let S be a resolution-free clause set that does not contain the empty clause. Let C be an element of S. Then L($\overline{C}$) is not element of S.

Proof:

If L($\overline{C}$) would be an element of S then S would be non resolution-free since we know from Lemma 3.2 that resolution can be performed on C and L($\overline{C}$).

Hence, L($\overline{C}$) is not element of S. ❑

This is the property why we name Lucky-negation to "lucky", namely it provides a clause outside from the clause set and this is not so trivial.

4 Properties of Sub-Model Propagation

In this section we will see that clause set remains the same for certain properties after sub-model propagation by the Lucky-negation of a minimal clause of the set.

Lemma 4.1

Let S be a resolution-free clause set that does not contain the empty clause. Let C be a minimal clause of S. Then S [L(C)] does not contain the empty clause.

Proof:

It is suffices to show that there is no such a clause in S such that the sub-model propagation by L(C) deletes every literal from it, i.e., L($\overline{C}$) is not element of S. (We do not need consider clauses which are subset of L($\overline{C}$) since C is a minimal clause from S). We know from Lemma 3.3 that L($\overline{C}$) is not element of S.
Hence, S [L(C)] does not contain the empty clause. ❑

We see that Lucky-negation allows to simplify (perform sub-model propagation) the clause set without the danger that the clause set becomes unsatisfiable (it will not contain the empty clause).

Lemma 4.2

Let S be a resolution-free clause set and let I be an assignment. Then S [I] is resolution-free.

Proof:

Let B' and C' be elements of S [I]. Then we know by definition of sub-model propagation that $B' = B$ [I] and $C' = C$ [I] for some elements B and C of S. It can be shown that B and $\overline{I}$ are semi-equivalent and C and $\overline{I}$ are semi-equivalent, i.e., B and C differ in no variable among variables of I. From this and from the definition of sub-model propagation we can obtain that B [I] and C [I] differ in the same variables like B and C, i.e., no resolution can be performed on B [I] and C [I]. Hence, S [I] is resolution-free. ❑

Now we see that a resolution-free clause set that does not contain the empty clause has the same properties after sub-model propagation by the Lucky-negation of some minimal clause of the set.

Lemma 4.3

Let X be a clause set, I be an assignment. Let S be a clause set such that X [I] = S. Let B be an element of S. Then X [$I \cup$ L(B)] = S [L(B)].

Proof:

I and B have no common variable since B is element of $S = X[I]$. Therefore $X[I \cup L(B)] = X[I][L(B)]$. Hence, $X[I \cup L(B)] = S[L(B)]$. ❑

This lemma allows us to join the sub-models to get the final model. Note that this lemma could be written in a more general way.

Lemma 4.4

Let S be a non-empty clause that does not contain the empty clause. Let B be an element of S. Then $|S[L(B)]| < |S|$ and $|V\{S[L(B)]\}| < |V\{S\}|$.

Proof:

S and $L(B)$ have at least one common variable.

Hence, $|S[L(B)]| < |S|$ and $|V\{S[L(B)]\}| < |V\{S\}|$. ❑

5 The Unicorn-SAT Algorithm

Now we introduce the Unicorn-SAT algorithm and we show that it solves the resolution-free SAT problem in polynomial time.

Let $RfS(S)$ be the predicate which is true if S is resolution-free clause set, false otherwise. Let $MinC(S)$ be the set of minimal clauses of S. Let $M(S, I)$ be the predicate which is true if I is a model for S, false otherwise.

We use "{}" to mark formulas that are true at the respective points of algorithm, in order to prove the correctness of the algorithm in the Hoare calculus.

Algorithm 5.1 (Unicorn-SAT):

Unicorn-SAT(X, Z)

input: clause set X that is resolution-free and does not contain the empty clause,

output: clause Z, a model of X.

START

$\{\emptyset \notin X \wedge RfS(X)\}$	the precondition
$\{\emptyset \notin X \wedge RfS(X) \wedge X[\emptyset] = X\}$	logical consequence rule and by a trivial property of sub-model propagation
$(S, I) := (X, \emptyset)$;	
$\{\emptyset \notin S \wedge RfS(S) \wedge X[I] = S\}$	by assignment axiom
$\{\emptyset \notin S \wedge RfS(S) \wedge X[I] = S\}$	the loop invariant
while $(S \neq \emptyset)$ do	
$\{\emptyset \notin S \wedge RfS(S) \wedge X[I] = S \wedge S \neq \emptyset\}$	by while rule
$\{\emptyset \notin S \wedge RfS(S) \wedge X[I] = S\}$	logical consequence rule
let $B \in MinC(S)$;	

$\{\emptyset \notin S \wedge \mathrm{RfS}(S) \wedge X[I] = S \wedge B \in \mathrm{MinC}(S)\}$ by assignment axiom

$\{\emptyset \notin S[\mathrm{L}(B)] \wedge \mathrm{RfS}(S[\mathrm{L}(B)]) \wedge X[I \cup \mathrm{L}(B)] = S[\mathrm{L}(B)]\}$ logical consequence rule and by Lemmas 4.1 - 4.3

$S := S[\mathrm{L}(B)];$

$\{\emptyset \notin S \wedge \mathrm{RfS}(S) \wedge X[I \cup \mathrm{L}(B)] = S\}$ by assignment axiom

$\{\emptyset \notin S \wedge \mathrm{RfS}(S) \wedge X[I \cup \mathrm{L}(B)] = S\}$

$I := I \cup \mathrm{L}(B);$

$\{\emptyset \notin S \wedge \mathrm{RfS}(S) \wedge X[I] = S\}$ by assignment axiom

$\{\emptyset \notin S \wedge \mathrm{RfS}(S) \wedge X[I] = S\}$

od

$\{\emptyset \notin S \wedge \mathrm{RfS}(S) \wedge X[I] = S \wedge S = \emptyset\}$ by while rule

$\{X[\mathrm{I}] = \emptyset\}$ logical consequence rule

$Z := I;$

$\{X[Z] = \emptyset\}$ by assignment axiom

$\{\mathrm{M}(X, Z)\}$ the postcondition

HALT.

Now we see that from the precondition (the clause set is resolution-free and does not contain the empty clause) the postcondition follows (the computed assignment is a model for the clause set) if we follow the steps of the algorithm. The only question is whether the algorithm stops always or sometime runs for forever. We know from Lemma 4.4 that it stops always.

Theorem 5.1 (Correctness)

Let S be a resolution-free clause set that does not contain the empty clause. Then after execution of Unicorn-SAT(S, I), I is a model for S.

Proof:

From Algorithm 5.1 and from Lemma 4.4 we can obtain that Unicorn-SAT stops for every resolution-free clause set that does not contain the empty clause and gives back a model for it. ❑

Now we can prove that Unicorn-SAT is a polynomial time algorithm.

Theorem 5.2

Let S be a resolution-free clause set that does not contain the empty clause. Let n be the number of variables occurring in S and m the length of S. Then the time complexity of Unicorn-SAT(S, I) is $\min(O(n^2m), O(m^2n))$.

Proof:

From Lemma 4.4 we know that S decreases in every iterations at least by one clause and also at least by one variable. So there are $\min(n, m)$ iterations in the

worst case. The most expensive computation in an iteration is the sub-model propagation which is an O($m * n$) operation.
Hence, the time complexity of Unicorn-SAT(S , I) is min(O(n^2m), O(m^2n)). ❑

6 Example

We show how the Unicorn-SAT algorithm works for an example. We assume that variables are sorted by alphabetic order.

Example 6.1

Let X := $\{\{a, b, c\}, \{b, c, d, \bar{e}\}, \{a, \bar{b}, \bar{c}, d\}, \{c, \bar{d}, e\}\}$. X is resolution-free clause set and does not contain the empty clause, so Unicorn-SAT can be applied to it. Step of Unicorn-SAT and values of variables:

After initialization (S , I) = ($\{\{a, b, c\}, \{b, c, d, \bar{e}\}, \{a, \bar{b}, \bar{c}, d\}, \{c, \bar{d}, e\}\}$, $\emptyset$).

After first iteration (S , I , B) = ($\{\{d, \bar{e}\}, \{\bar{d}, e\}\}$, $\{a, \bar{b}, \bar{c}\}$, $\{a, b, c\}$).

After second iteration (S , I , B) = ($\emptyset$, $\{a, \bar{b}, \bar{c}, d, e\}$, $\{d, \bar{e}\}$).

The model Z for X is $\{a, \bar{b}, \bar{c}, d, e\}$.

References

1. Stephen Cook. The complexity of theorem proving procedures. In *Proc. 3rd Ann. ACM Symp. on Theory of Computing*, pages 151--158, New York, 1971.
2. S. Even, A. Itai and A. Shamir. On the complexity of timetable and multi-commodity flow problems. *SIAM Journal on Computing*, 5(4), 1976.
3. Bengt Aspvall, Michael F. Plass and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121--123, March 1979.
4. W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulas. *Journal of Logic Programming*, 1(3):267--284, 1984.
5. Bengt Aspvall. Recognizing disguised NR(1) instances of the satisfiability problem. *Journal of Algorithms*, 1:97--103, 1980.
6. Vijaya Chandru and John Hooker. Extended Horn sets in propositional logic. Journal of the ACM, 38(1):205--221, 1991.
7. M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7:201--215, 1960.