Algebraic Simplification Techniques for Propositional Satisfiability

João Marques-Silva

Department of Informatics, Technical University of Lisbon,
IST/INESC/CEL, Lisbon, Portugal
jpms@inesc.pt

Abstract. The ability to reduce either the number of variables or clauses in instances of the Satisfiability problem (SAT) impacts the expected computational effort of solving a given instance. This ability can actually be essential for specific and hard classes of instances. The objective of this paper is to propose new simplification techniques for Conjunctive Normal Form (CNF) formulas. Experimental results, obtained on representative problem instances, indicate that large simplifications can be observed.

1 Introduction

Recent years have seen the proposal of several effective algorithms for solving Propositional Satisfiability (SAT), that include, among others, local search and variations, backtrack search improved with different search pruning techniques, backtrack search with randomization and restarts, continuous formulations and algebraic manipulation. (These different algorithms are further described and cited in [3].) Moreover, these algorithms have allowed efficiently solving different classes of instances of SAT. It is generally accepted that whereas most algorithms for solving SAT can be competitive in proving satisfiability for different classes of instances, backtrack search is preferred when the objective is to prove unsatisfiability. Nevertheless, algebraic simplification solutions are also known to be competitive for proving unsatisfiability in specific contexts [2].

The main goal of this paper is to propose and categorize new simplification techniques, and illustrate the effectiveness of algebraic simplification as a preprocessing tool for SAT algorithms. Moreover, we illustrate the application of the proposed simplification techniques in real-world instances of SAT.

The paper is organized as follows. We start with a few definitions in Section 2. Next we address algebraic simplification, namely the techniques in this paper. Section 4 provides experimental results on applying the proposed simplification techniques on real-world instances of SAT. Finally, Section 5 concludes the paper.

2 Definitions

This section introduces the notational framework used throughout the paper. Propositional variables are denoted $x_1, \ldots, x_n$, and can be assigned truth values 0

R. Dechter (Ed.): CP 2000, LNCS 1894, pp. 537–542, 2000.

(or F) or 1 (or T). In addition to letter x, and whenever necessary, we will use letters y, w and z to denote variables. To denote specific variables we may also use $x_i, x_j, x_k, \ldots$. A literal l is either a variable x_i (i.e. a positive literal) or its complement $\neg x_i$ (i.e. a negative literal). A clause ω is a disjunction of literals and a CNF formula φ is a conjunction of clauses. When referring to specific clauses we will utilize subscripts $a, b, \ldots$, and when referring to sub-formulas of a CNF formula we will utilize subscripts $r, s, \ldots$. Disjunctions of literals, not necessarily representing clauses will be represented as $\alpha, \beta, \gamma, \delta, \epsilon$.

3 Algebraic Simplification

Different formula simplification techniques have been proposed over the years. A detailed account of these techniques if provided in [3]. In this section we concentrate on two new techniques, namely *support-set variable equivalence* and *inference of binary clauses*.

3.1 Support Set Variable Equivalence

In many practical situations, a sub-formula φ_s of a CNF formula φ actually describes a Boolean function $x_i = f(y_1, \ldots, y_k)$. For example, the sub-formula $\varphi_a = (y_1 \vee \neg x) \wedge (y_2 \vee \neg x) \wedge (\neg y_1 \vee \neg y_2 \vee x)$ describes the Boolean function $x = y_1 \wedge y_2$. More interestingly, if we have two sub-formulas $\varphi_a = (y_1 \vee \neg x) \wedge (y_2 \vee \neg x) \wedge (\neg y_1 \vee \neg y_2 \vee x)$ and $\varphi_b = (y_1 \vee \neg z) \wedge (y_2 \vee \neg z) \wedge (\neg y_1 \vee \neg y_2 \vee z)$, then we can conclude that $x = y_1 \wedge y_2$ and $z = y_1 \wedge y_2$. Hence, x and z are indeed *equivalent* and we can replace z with x and vice-versa. Observe that, by suitable resolution operations, we could easily derive the clauses $(x \vee \neg z) \wedge (\neg x \vee z)$, obtaining the same conclusion.

The previous example suggests a pattern-matching approach for identifying a set E of variables that can be expressed as a Boolean function f of some other set S of variables (i.e., *the support set*) and thus replace the variables in set E by a single variable. This approach is simply too time consuming for sets S of arbitrary size, and so we restrict S to be of size 2, i.e. we only consider Boolean functions of two variables.

If the support set S is restricted to be of size 2, it becomes feasible to enumerate all possible Boolean functions of 2 variables, and determine the *irredundant* CNF formulas associated with each Boolean function. This information is shown in Table 1. (Observe that the missing six boolean functions are either constant 0 or 1, the actual function of a *single* variable or of its complement.) We can now apply a straightforward pattern matching algorithm to a CNF formula, and identify sets E of variables with a common two-variable support set. These variables can then be replaced by a single variable. Clearly, after variable replacement we can apply well-known formula simplification techniques [3].

Other researchers [1] have observed the existence of variable equivalences based on support sets, but on arbitrary instances of SAT (i.e. not in CNF format). Moreover, the underlying approach of [1,4] is significantly more time consuming (indeed exponential in the worst-case) than the one proposed above.

Table 1. Two-variable CNF formulas

Boolean function	CNF formula
$x \equiv f_1(a,b) = a \wedge b$	$(a \vee \neg x) \wedge (b \vee \neg x) \wedge (\neg a \vee \neg b \vee x)$
$x \equiv f_2(a,b) = a \vee b$	$(\neg a \vee x) \wedge (\neg b \vee x) \wedge (a \vee b \vee \neg x)$
$x \equiv f_3(a,b) = a \leftrightarrow b$	$(a \vee b \vee x) \wedge (a \vee \neg b \vee \neg x) \wedge (\neg a \vee \neg b \vee x) \wedge (\neg a \vee b \vee \neg x)$
$x \equiv f_4(a,b) = \neg a \wedge b$	$(\neg a \vee \neg x) \wedge (b \vee \neg x) \wedge (a \vee \neg b \vee x)$
$x \equiv f_5(a,b) = a \wedge \neg b$	$(a \vee \neg x) \wedge (\neg b \vee \neg x) \wedge (\neg a \vee b \vee x)$
$x \equiv f_6(a,b) = \neg(a \wedge b)$	$(a \vee x) \wedge (b \vee x) \wedge (\neg a \vee \neg b \vee \neg x)$
$x \equiv f_7(a,b) = \neg(a \vee b)$	$(\neg a \vee \neg x) \wedge (\neg b \vee \neg x) \wedge (a \vee b \vee x)$
$x \equiv f_8(a,b) = \neg(a \leftrightarrow b)$	$(a \vee b \vee \neg x) \wedge (a \vee \neg b \vee x) \wedge (\neg a \vee b \vee x) \wedge (\neg a \vee \neg b \vee \neg x)$
$x \equiv f_9(a,b) = a \vee \neg b$	$(\neg a \vee x) \wedge (b \vee x) \wedge (a \vee \neg b \vee \neg x)$
$x \equiv f_{10}(a,b) = \neg a \vee b$	$(a \vee x) \wedge (\neg b \vee x) \wedge (\neg a \vee b \vee \neg x)$

Another interesting result, is that the equivalence reasoning conditions proposed by C.-M. Li in [2] are also superseded by support set equivalences and *selective resolution* (see [1,3] for a definition). We should observe, in particular, that the utilization of function f_3 on two variables $x = (a \leftrightarrow b)$ and $y = (a \leftrightarrow b)$, for deriving the equivalence between x and y, corresponds to Li's inference rule (5) [2], the other rules being superseded by selective resolution, unit-clause rule and two-variable equivalence.

Despite the potential interest of identifying support sets of variables in a CNF formula, in the next section a further generalization is proposed, that subsumes variable equivalences based on two-variable support sets.

3.2 Generalized Inference of Binary Clauses

In this section we propose to study subsets of clauses for inferring binary clauses, which not only identify support set equivalences, but also provide more general conditions for deriving binary clauses. In what follows, all proposed conditions can be explained by resorting to resolution. However, it is in general extremely hard to decide to which clauses the resolution operation should be applied to, being computationally infeasible to apply the resolution operation to all possible pairs of clauses. The objective of studying sets of clauses is to indirectly select to which sets of clauses the resolution operation should be applied to.

Moreover, the reasoning technique to be described below is categorized in terms of how many binary clauses and ternary clauses are involved. Clearly, the size and number of k-ary clauses involved could be made arbitrary, but the computational overhead could become prohibitive. In general each proposed condition is classified as being of the form mB/nT, meaning that m binary clauses and n ternary clauses are involved.

Let us start by considering an illustrative example. Let $\varphi_c = (y_1 \vee \neg x) \wedge (y_2 \vee \neg x) \wedge (\neg y_1 \vee \neg y_2 \vee z)$ be a sub-formula. The application of the resolution operation between the three clauses allows deriving $(\neg x \vee z)$. Similarly, for the sub-formula $\varphi_d = (y_1 \vee \neg z) \wedge (y_2 \vee \neg z) \wedge (\neg y_1 \vee \neg y_2 \vee x)$, the resolution operation

Table 2. Rules for Inferring Binary/Unit Clauses

Clause Pattern	Inferred Clause(s)
$(l_1 \vee \neg l_2) \wedge (l_1 \vee \neg l_3) \wedge (l_2 \vee l_3 \vee l_4)$	$(l_1 \vee l_4)$
$(l_4 \vee \neg l_2) \wedge (l_1 \vee \neg l_3) \wedge (l_2 \vee l_3 \vee l_4)$	$(\neg l_3 \vee l_4), (l_1 \vee l_4)$
$(l_1 \vee l_2 \vee l_3) \wedge (\neg l_1 \vee \neg l_2 \vee l_3) \wedge (l_1 \vee \neg l_2 \vee l_4) \wedge (\neg l_1 \vee l_2 \vee l_4)$	$(l_3 \vee l_4)$
$(l_1 \vee l_2 \vee \neg l_3) \wedge (l_1 \vee l_2 \vee \neg l_4) \wedge (l_1 \vee l_2 \vee \neg l_5) \wedge (l_3 \vee l_4 \vee l_5)$	$(l_1 \vee l_2)$
$(l_1 \vee l_2) \wedge (\neg l_1 \vee l_2 \vee l_3)$	$(l_2 \vee l_3)$
$(l_1 \vee \neg l_2) \wedge (l_1 \vee \neg l_3) \wedge (l_1 \vee \neg l_4) \wedge (l_2 \vee l_3 \vee l_4)$	(l_1)
$(l_1 \vee l_2 \vee \neg l_3) \wedge (l_1 \vee l_2 \vee \neg l_4) \wedge (l_3 \vee l_4)$	$(l_1 \vee l_2)$
$(l_1 \vee l_2) \wedge (\neg l_1 \vee l_3 \vee l_4) \wedge (l_2 \vee l_3 \vee \neg l_4)$	$(l_2 \vee l_3)$

allows deriving the clause $(x \vee \neg z)$. It is interesting to observe that we have just illustrated a different approach for proving x equivalent to z for the first example of the previous section.

In general, let us consider a sub-formula of the form $\varphi_s = (l_1 \vee \neg l_2) \wedge (l_1 \vee \neg l_3) \wedge (l_2 \vee l_3 \vee l_4)$, where $l_1, \ldots, l_4$ are any literals. The application of resolution allows deriving the binary clause $(l_1 \vee l_4)$. Since 2 binary clauses and 1 ternary clause are involved in deriving the resulting binary clause, we say that 2B/1T reasoning was applied. The other form of 2B/1T reasoning is the following. Let the sub-formula be $\varphi_t = (l_4 \vee \neg l_2) \wedge (l_1 \vee \neg l_3) \wedge (l_2 \vee l_3 \vee l_4)$. Then, application of resolution allows deriving the binary clauses $(l_3 \vee l_4)$ and $(l_1 \vee l_4)$.

Additional forms of mB/nT reasoning can be established. Due to efficiency concerns, our analysis will be restricted to 2B/1T, 1B/1T, 1B/2T, 3B/1T and 0B/4T. The resulting set of unit/binary clause inference rules is summarized in Table 2. Moreover, and from the previous examples and claims, we can readily conclude that support set variable equivalence (described in the previous section) is superseded by 2B/1T and 0B/4T reasoning.

4 Experimental Results

This section evaluates the practical application of the algebraic simplification techniques described in this paper. These techniques can be evaluated according to two main metrics:

- The ability to effectively simplify the original formula.
- The effective reduction in the amount of search, when using formula simplification techniques within a preprocessing engine for backtrack search SAT algorithms.

In this paper we concentrate on the ability to simplify the original formula. The reduction in the amount of search is analyzed in [3]. Moreover, the problem instances used in this section are also described in [3]

Table 3 shows the results, namely the number of variables and clauses before and after applying the simplification techniques. As can be concluded, for most of

Table 3. Formula Simplification

Instance	Initial Formula		Final Formula	
	Variables	Clauses	Variables	Clauses
bf1355-075	2180	6778	722	3075
bf2670-001	1393	3434	411	1252
ssa2670-130	1359	3321	440	1295
ssa2670-141	986	2315	283	883
barrel6	2306	8931	519	2209
barrel7	3523	13765	805	3467
longmult5	2397	7431	1483	5180
longmult6	2848	8853	1824	6376
queueinvar12	1112	7335	1049	9884
queueinvar16	1168	6496	1088	7114
c1908	1917	5096	658	2248
c1908_bug	1919	5100	659	2250
c2670	2703	6756	1220	3725
c2670_bug	2708	6696	1162	3553
dlx2_cc_bug02	1515	12808	1486	13965
dlx2_cc_bug08	1515	12808	1486	13890

the instances considered, large reductions in the number of variables and clauses can be achieved. In some cases (e.g. *barrel6* and *barrel7*) the final number of variables is one fourth of the original number. Similar reductions can be observed in the number of clauses. Nevertheless, for some instances (e.g. *queueinvar16* and *dlx2_cc_bug08*) the amount of simplification in the number of variables is negligible. In these cases, it is interesting to observe an increase in the number of clauses, due to the application of clause inference techniques (see Section 3.2).

5 Conclusions

This paper proposes new techniques for the algebraic simplification of propositional formulas. These techniques have been incorporated into a preprocessing system to be used with any SAT algorithm. Preliminary experimental results, obtained on real-world instances of SAT, clearly demonstrate the effectiveness of the proposed techniques, allowing significant reductions in the sizes of the resulting CNF formulas.

Despite the promising results obtained, and given the amount of simplification achieved on most problem instances, the next natural step is to incorporate the same algebraic simplification techniques into backtrack search SAT algorithms, to be applied during the search.

References

1. J. F. Groote and J. P. Warners. The propositional formula checker heerhugo. Technical Report SEN-R9905, CWI, January 1999. 538, 539
2. C.-M. Li. Integrating equivalency reasoning into davis-putnam procedure. In *Proceedings of the National Conference on Artificial Intelligence*, August 2000. Accepted for publication. 537, 539
3. J. P. Marques-Silva. Algebraic simplification techniques for propositional satisfiability. Technical Report RT/01/2000, INESC, March 2000. 537, 538, 539, 540
4. G. Stålmarck. A system for determining propositional logic theorems by applying values and rules to triplets that are generated from a formula, 1989. Swedish Patent 467 076 (Approved 1992), US Patent 5 276 897 (approved 1994), European Patent 0 403 454 (approved 1995). 538