Two Easy Theories Whose Combination is Hard

V.R. Pratt
M.I.T.

Sept. 1, 1977

Abstract

We restrict attention to the validity problem for unquantified disjunctions of literals (possibly negated atomic formulae) over the domain of integers, or what is just as good, the satisfiability problem for unquantified conjunctions. When $=$ is the only predicate symbol and all function symbols are left uninterpreted, or when $\leq$ is the only predicate symbol (taking its standard interpretation on the integers) and the only terms are variables and integers, then satisfiability is decidable in polynomial time. However when $\leq$ and uninterpreted function symbols are allowed to appear together, satisfiability becomes an NP-complete problem. This combination of the two theories can arise for example when reasoning about arrays (the uninterpreted function symbols) and subscript manipulation (where $\leq$ arises in considering subscript bounds). These results are unaffected by the presence of successor, which also arises commonly in reasoning about subscript manipulation.

1 Introduction

Nelson and Oppen [NO77] have shown that conjunctions of equalities and negated equalities between terms involving only uninterpreted function symbols can be tested for satisfiability in time proportional to the square of the length of the conjunction. For example, $x = y \wedge y = z \wedge f(x) = w \wedge w \neq f(z)$ is not satisfiable although it would be in the absence of any one of the four literals making up the formula.

Another language whose satisfiability problem is about as easy to decide is that of conjunctions of integer inequalities between variables or integers. To decide satisfiability, construct a graph whose vertices correspond to the variables of the formula, with an extra vertex corresponding to 0. Construct edges, one per inequality, with edge (x, y) labelled i corresponding to an

inequality which in essence expresses $x + i \leq y$. For example, $\neg(u \leq v)$ would give rise to an edge (v, u) labelled 1, while $u \leq 3$ would generate an edge $(u, 0)$ labelled -3. Then the conjunction is satisfiable if and only if the graph contains a cycle of positive weight, i.e., one whose labels add up to a positive integer. This can be decided, e.g., by forming the max/+ transitive closure of the graph and searching for a self-edge with a positive label. This procedure is evidently adaptable to the case when successor may be used.

2 Result

We now show that conjunctions of literals involving both $\leq$ (with its standard interpretation on the integers) and uninterpreted function symbols have an NP-complete satisfiability problem. To do this we give an easily computed function H from "monotonic normal form" formulae of propositional calculus (negations appear only on atomic formulae) to conjunctions of this language such that P is satisfiable iff $H(P)$ is. We write $x \leq y \leq z$ for $x \leq y \wedge y \leq z$.

The intuition behind the following definition is that P is to be translated to a directed acyclic graph some of whose edges are fixed and some of whose edges are under the control of assignments of truth values to the letters of P. The graph will have an input and an output vertex. The objective is that a path will exist from input to output just when the given assignment of truth values to the letters of P falsifies P. Before explaining this further, let us give the full definition of H.

$H(P)$:	$h(P) \wedge \neg(\text{in}(P) \leq \text{out}(P)) \wedge 0 \leq x_a \leq 1 \wedge 0 \leq x_b \leq 1 \wedge \ldots$	(a,b,... letters of P)
$\text{in}(P)$:	$g_1(P)$	
$h(P)$:	$g_2(P)$	
$\text{out}(P)$:	$g_3(P)$	
$g(a)$:	$[f_i(0), \text{true}, f_i(x_a)]$	(a is any propositional letter)
$g(\neg a)$:	$[f_i(1), \text{true}, f_i(x_a)]$	
$g(P \mid Q)$:	$[\text{in}(P), h(P) \wedge \text{out}(P) \leq \text{in}(Q) \wedge h(Q), \text{out}(Q)]$	
$g(P \wedge Q)$:	$[\text{in}(P), \text{in}(P) \leq \text{in}(Q) \wedge h(P) \wedge h(Q) \wedge \text{out}(P) \leq \text{out}(Q), \text{out}(Q)]$	

The intent of "f_i" is that a different i be chosen for each invocation of g, i.e., for each occurrence of a propositional letter; thus $H(P)$ will always have exactly two occurrences of each f_i appearing in it.

The function $g(x)$ produces the triple $[g_1(x), g_2(x), g_3(x)]$ which for clarity we have renamed $[\text{in}(x), h(x), \text{out}(x)]$. Although H produces a conjunc-

tion of inequalities, g is considered to produce the corresponding graph, [input vertex, set of edges, output vertex]. Let us look at a particular example to see what is happening. The formula consisting of just the propositional letter a is mapped by H to $\neg(f_0(0) \leq f_0(x_a)) \wedge 0 \leq x_a \leq 1$. Clearly the latter is satisfied by taking x_a to be 1 and f_0 to be negation. Looking at the role of $g(a)$ in this, we see that $g(a)$ built a graph with input vertex $f_0(0)$, output vertex $f_0(x_a)$, and no edges (true corresponds to the empty conjunction). There is a path from input to output just when a is false, in the sense that $f_0(0) \leq f_0(x_a)$ just when $x_a = 0$. Now let us consider the more complicated example $a \wedge \neg a$, which is mapped to $f_0(0) \leq f_1(1) \wedge f_0(x_a) \leq f_1(x_a) \wedge \neg(f_0(0) \leq f_1(x_a)) \wedge 0 \leq x_a \leq 1$. To see that this is unsatisfiable, first observe that only $x_a = 0$ or $x_a = 1$ could satisfy it. In either case we may infer $f_0(0) \leq f_1(x_a)$, which contradicts the explicitly given $\neg(f_0(0) \leq f_1(x_a))$. Viewing $g(a \wedge \neg a)$ as a graph, if we consider "in$(P) \leq$ in(Q)" and "out$(P) \leq$ out(Q)" to have the effect of connecting respectively the inputs and outputs of the graphs of P and Q together (parallel connection), then all we have done is provide for the existence of a path no matter whether a is true or false, i.e., no matter whether the circuit is "closed" via $f_0(0) \leq f_0(x_a)$ or $f_1(1) \leq f_1(x_a)$. In the case of the disjunction $P \mid Q$, parallel connection is replaced by series connection. We hope that this informal discussion will make a more formal argument unnecessary. The key observation in the formal argument is that P is false for a given assignment of truth values to the propositional letters of P if and only if $H(P)$ is unsatisfiable when each x_a is assigned 1 if propositional letter a is assigned true and 0 otherwise.

3 Interpretation

The interest in this result lies not so much in the proof, which is routine, as in its relevance to the mechanization of logics incorporating many theories. This is the situation that obtains for example with systems for verifying computer programs, which may need to deal with programs that operate on a variety of distinct data types each having its own theory. It is natural to ask, what does the complexity of the constituent theories tell us about the complexity of their combination? This result shows that the overhead associated with simply combining quantifier-free disjunctive theories may be as great as the gap between P and NP.

We were led to the question answered above in the following way. In the course of implementing a proof checker [LP77] for dynamic logic (a new

approach to program verification), we found that the quantifier-free disjunctive theory of $\leq$ and successor over the integers fulfilled all our arithmetic needs for the kinds of programs we were encountering, which in view of the existence of the polynomial decision procedure described above turned out to be most convenient. When we attempted to incorporate "read-only" arrays into the theory (essentially the theory of equality with uninterpreted function symbols, but without array updating operations, which themselves lead to NP-completeness [DS76]) we encountered the difficulty summarized by the above result.

The interest in quantifier-free conjunctive theories stems from the approach to handling the combination of many theories in which the theorem to be proved is first negated, converted to disjunctive normal form, Skolemized, universal variables instantiated somehow when appropriate (e.g., by the user), and tested for satisfiability disjunct by disjunct. Provided each disjunct is easy to so test, the bulk of the running time can be accounted for by the large number of disjuncts that may arise. While this in theory remains an apparently insuperable obstacle, in practice one can make very effective use of bit vector representations of disjuncts to make the generation of the disjuncts relatively painless for quite substantial problems. This leaves the cost of testing each of the disjuncts. This too can be made similarly painless by keeping track of which literals of each disjunct were responsible for its unsatisfiability. (It is not necessary to search for the smallest set of such literals in each disjunct, the method is extremely effective even with token attention to finding a small subset.) Thus the first step in testing a disjunct is to see whether it contains as a subset one of the unsatisfiable sets already recorded, a process requiring just one bit vector operation per recorded set. In practice we have found this technique to be of value, often reducing running times from the order of minutes to fractions of a second.

Our experience with the number of sets of unsatisfiable literals that are recorded, for the particular problems with which we have to deal, is that the sets are in almost one-to-one correspondence with the "elementary" facts apparently needed to prove the theorem in question. Thus even though the propositional form of the theorem may have given rise to several thousand disjuncts, only ten or twenty sets may be recorded, so that almost all of the disjuncts can each be disposed of with just a small number of bit vector operations (in our case involving a few dozen bits per vector). The effect is to reduce the constant factor in front of the inevitable exponential to such an extent that the "visibly useful" work of discovering and checking the elementary facts supporting the theorem is comparable with the work done manipulating the many disjuncts. Of course this approach is enormously

sensitive to problem size, putting a very sharp limit on the size of problem treatable in this way. In applying this technique to proof checkers, as in our case, this simply means that the user must structure his proof so that the steps of the proof are not "unreasonably" large from the system's point of view.

In view of these remarks it should be clear how polynomial-time algorithms for satisfiability of quantifier-free conjunctive theories are of considerable value despite the NP-completeness of the unrestricted-format (but still quantifier-free) formulae in which they arise. How were we [LP77] affected by the NP-completeness of the combined theories described above? Our approach has been to permit the user to claim that a proof step will in fact hold for the rationals, for which domain the NP-completeness result above is replaced by a polynomial-time algorithm. In fact our experience has been that most arguments that arise in practice about the integers happen to hold for the rationals. In this way, the user can hold complexity at bay either by using small proof steps on those rare occasions when the complete integer theory is needed, or by making more general claims that, despite their greater generality, are easier to check. This illustrates the principle that there are other ways of making a proof easier to follow than just taking shorter steps.

References

[DS76] P.J. Downey and R. Sethi. Assignment commands and array structures. In *17th IEEE Symposium on Foundations of Computer Science*, pages 57–66, October 1976.

[LP77] S.D. Litvintchouk and V.R. Pratt. A proof checker for dynamic logic. In *5th International Joint Conference on A.I.*, pages 552–558, August 1977.

[NO77] G. Nelson and D.C. Oppen. Fast decision algorithms based on union and find. In *18th IEEE Symposium on Foundations of Computer Science*, October 1977.