CAD of VLSI Design - 046880

Homework Assignment No 5
Out : 22/12/2009 Due by: 15/01/2009

Topic: Solving Satisfiability problems using BDDs

Introduction

Boolean satisfiability (SAT) is a famous NP-complete decision problem. Given a boolean function f(x1,x2,x3,….), we need to decide if there exists a *satisfying assignment* of boolean values to the input variables x1,x2,x3,…, that makes the function TRUE. If the decision is NO (i.e. f=0 for all possible assignments to x), f is called *infeasible* or *unsatisfiable*. If the decision is YES, we usually want to find the input values of a satisfying assignment.

In the classical formulation, the function f is represented as a product of sums (POS), which is also called *conjunctive normal form* (CNF). For example:

$F_1 = (a + c)\ (b+ c)\ (a' + b' + c)(d + e)$

In this form, F_1 is a product (conjunction) of *clauses*, and each clause contains a sum (disjunction) of literals. Literals can be positive (i.e. input variables a, b, c,…) or negative (i.e. complements: a', b', c',….).
In a more general formulation, a function f can be represented as a product of other functions, called *constraints*. For example:

$F_2= (\ u(x_1,x_2,x_3,\ldots.))\ (\ v(x_1,\ldots..))\ \ (\ u(x_1,\ldots.))$

Thus the satisfiability problem is the quest for a variable assignment that makes all the clauses or constraints TRUE.

You are required to answer three of the following four questions.

1. Simple questions

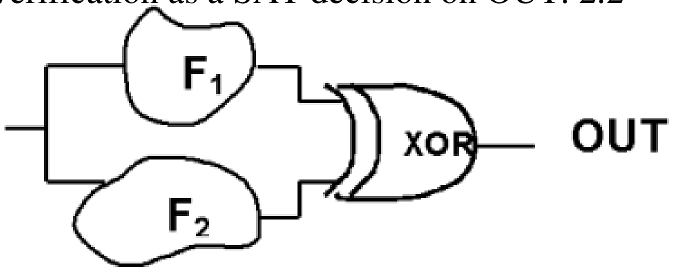
1.1 Look at F_1 above. Is it satisfiable? Can you suggest a satisfying assignment? Is it unique, or are there other satisfying assignments?

1.2 Petrick's method to solve the problem is to perform all the products using rules of boolean algebra, until a sum of products is obtained (this is called DNF – explain the meaning of the name). Every term in the expression obtained is a satisfying solution. An example of the application of Petrick's method to the covering problem is described in DeMicheli's book, p.279. Use this method to find all the satisfying assignments of the function: $F_{12}=(a)(a+b)(b+c')(c+d)(d)$

1.3 Petrick's method seems straightforward, but the SAT problem is known to be NP-complete. Prove/explain that Petrick's method indeed involves an exponential number of operations, such that it is not practical for functions with many variables.

2. Using SAT for logic verification

2.1 We have two implementations F1 and F2 of some logic function, and we want to verify that they are equivalent. We feed the same inputs to f1 and f2, and combine their outputs with a xor gate as shown in Fig. 2. Describe the logic equivalence verification as a SAT decision on OUT. 2.2



2.2 Recall that if a NXOR gate replaces the XOR gate in Figure 2, the problem becomes a tautology problem.

2.2.1 State in your own words the definition of the satisfiability problem and the tautology problem, as you understand them.

2.2.2 What is the relationship between the two problems?

2.2.3 Compare this SAT-based method of logic equivalence verification with the tautology-based method. [hint: DeMicheli's book, p.86]

2.3 Formulate the tautology problem such that a process similar to Petrick's method can be applied to solve it. Prove that it is also exponential.

2.4 SAT is a fundamental problem in many areas of CAD. Some examples are: automatic test pattern generation, model property checking, delay analysis, and noise analysis. Here is an example related to test generation: All three outputs of the circuit in Fig. 3 are required to be 1. Formulate a SAT problem for finding an appropriate input assignment. What is the function f in the SAT problem? [hint: treat the 3 outputs as constraints]

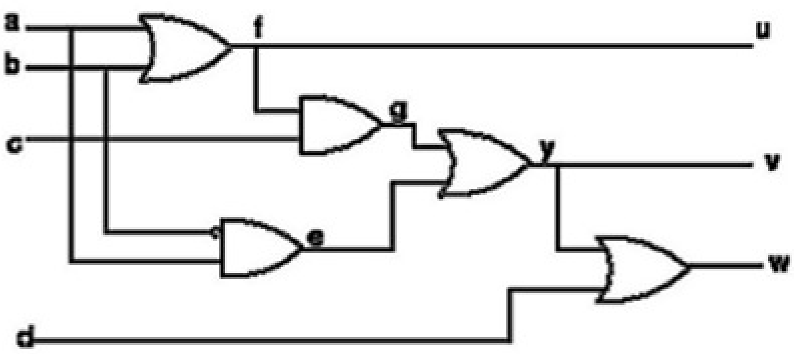


Figure 3

3. Classical SAT algorithms

A naive SAT algorithm will solve the SAT problem in $O(2^n)$ time. However, various methods have been developed to minimize the effective number of literals that need to be checked. The following rules can be used to minimize the CNF:
The unit clause rule states that if a single clause within a CNF is unresolved (i.e. it can be either 0 or 1) when values have been assigned to all of its variables except a single variable, then there is exactly one option to satisfy this clause. For example, (a'+b'+c') when a=b=1, can be satisfied only by c=0.
A variable is called *pure* if its literals are either all positive or all negative. The pure literal rule states that pure variables can be assigned appropriate values that satisfy all clauses containing them, and this does not affect satisfiability of the whole function.

3.1 Prove the unit clause rule in general. How can this rule minimize the number of literals to be checked in the SAT problem?

3.2 Prove the pure literal rule in general. Give examples of the rule.

3.3 What can be said about a function that contains a pure literal?

3.4 Classical SAT algorithms use a backtracking search that, at each node in the search tree, select an assignment and prunes subsequent search by iterative application of the unit clause and pure literal rules. Draw the search tree and explain your pruning decisions for the following function: f=(a+b+c)(a+b+c')(a'+b+c')(a+c+d)(a'+c+d').

4. SAT using BDDs

4.1 Assume that you have a BDD representing f. Explain how to check satisfiability, and why it is trivial. Compare with tautology on BDDs.

4.2 Sometimes the BDD of f is too large and cannot be built in memory. It may be possible to save memory if we are dealing with a unate function, such that we only need to check satisfiability of a cofactor (which is smaller than the whole function). Prove that the pure variable rule (from question 4) can be stated as follows: If a function is positive unate in x, then if a SAT solution exists, it can be found with the assignment x=1.

4.3 Also, prove that if a function is positive unate in x, it is sufficient to look for a SAT solution in f_x [Hint: remember that the cofactors are contained in each other, since f=x*A+B].

4.4 Write a similar conclusion for negative unate functions.

4.5 Assume you have two functions f and g, such that both are positive unate in variable x. You need to solve the SAT problem (f)(g)=1. Prove that it is sufficient to search for the solution in (f_x) (g_x). Use Figure 4 to illustrate your proof.

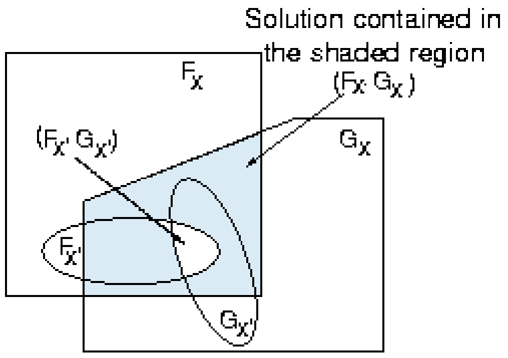


Figure 4