A Proposal of a project on Satisfiability

Tracy Larrabee, Heather Trumbower, Yumi Tsuji, Allen Van Gelder

January 15, 1993

For the DIMACS challenge, we wish to participate in the area of satisfiability testing. We can provide a collection of input formulas and an input formula generator, a variety of satisfiability algorithms and their implementations, and an investigation on how statistical methods may be applied in making inferences from the experimental results.

1 Input Formulas and a Generator

One contribution we can make to the DIMACS challenge is a collection of CNF formulas and a program to produce CNF formulas. A satisfying binding for these formulas would reduce to an input vector that would detect the presence of a particular kind of flaw in a digital circuit. That is, our CNF formulas are not randomly or theoretically generated but represent real-world practical problems. We have eleven circuits, each of which can generate thousands of formulas representing searches for many different kinds of circuit flaws. Most of the formulas produced are easy for our satisfier, but we have several hundred formulas that are either very difficult for our satisfier or that our satisfier has yet to handle. The formulas range in size from a few hundred to about 10,000 variables.

2 Satisfiability Algorithms and their Implementations

Following list is the various satisfiability algorithms which either already exist or being proposed. A brief description of each algorithm is given.

1. Nemesis Satisfier

   One satisfiability algorithm is part of Nemesis, a successful Automatic Test Pattern Generation system [Lar92, LT92]. Nemesis generates a Boolean formula that includes all the constraints that must be satisfied in order to generate a test pattern for a particular fault in a circuit, and then it satisfies the resultant formula. The design of the satisfier was originally motivated by the preponderance of width-2 clauses in the formulas generated in order to detect circuit faults: at least 2/3 of the clauses in every formula had only two literals, and more commonly 80–90% of the clauses were width-2 clauses.

   Accordingly, we decided to use the 2SAT satisfier proposed by Aspvall, Tarjan, and Plass [APT79] in order to generate a satisfying assignment for a CNF expression. To summarize the method: if necessary we assign values to a few variables until we have a portion of the formula that is purely 2CNF. We then iterate through the 2SAT satisfying assignments until we either find one that is consistent with the non-2CNF portion of the formula or until we prove that there is no such assignment. When we completed the satisfier we discovered that it worked well on CNF formulas that were not primarily composed of width-2 clauses.

2. Nemesis Satisfier with dynamic SCC's

The current implementation of our method builds an implication graph and determines strongly connected components from available binary clauses at discrete intervals only. After building the graph and determining variable equivalence classes, we begin to make variable assignments as we iterate through 2SAT bindings. In the course of making these variable assignments, we generate new binary clauses, which may be temporary because we may backtrack. These new binary clauses could be included in the original implication graph, with the provisio that the implication graph must degrade to the original state if we backtrack. That is, we would like the implication graph to change dynamically. This "dynamic implication graph" may speed up the search for a solution. We will report on the effect of this modification in terms of number of backtracks and execution time.

3. VG Satisfier - Improved DP Variant

   This algorithm for general clausal formulas is named *VG* as it was developed by A. Van Gelder [VG88]; it has three variants: *VG-normal*, *VG-dp*, and *VG-mis*. They all include the basic Davis-Putnam rules (unit clause and pure literal) as originally described [DP60] and the implementation of the backtracking techniques described by Davis, Logemann, and Loveland [DLL62]. The major difference among the three forms is in the strategy for the choice of a branching variable when the splitting rule is applied resulting in effectively three different algorithms.

   - **VG-normal**: The full algorithm described by Van Gelder [VG88]. In summary, VG-normal as applied to clausal formulas employs unit-clause and pure-literal rules. For general clausal formulas, equivalent literals are identified as well. It also selects branch variables dynamically according to its own scoring scheme, which weights the variables.
   - **VG-dp**: VG-normal degraded to simulate the Davis-Putnam procedure [DLL62].
   - **VG-mis**: VG-normal with a new branch-variable selection criterion; it transforms the input formula guided by a scheme using an algorithm which finds maximal independent set among the variables with certain properties. The VG-mis strategy is essentially a divide-and-conquer approach based on the following fact: If a formula is a conjunction of subformulas that are independent of other subformulas, in the sense that no two subformulas share a common variable, then this formula is satisfiable if and only if all of its subformulas are independently satisfiable.

4. k-closure Satisfier

   This is a satisfiability tester for CNF formulas that is based on the k-closure algorithm[VG93]. The method involves a limited use of resolution, called k-closure. The operation of k-closure consists of deriving by resolution clauses of length at most k, starting from the given formula, until no further such derivations are possible. More precisely, the current set of clauses S is *k-closed*; i.e. no clause of S is subsumed by a different clause of S, and the resolvent of each pair of resolvable clauses either is implied by (subsumed by) some clause in S or has more than k literals. Since the number of such clauses in a formula of n variables is $O(n^k)$, one pass of k-closure is in polynomial time. Our implementation is for $k = 3$, so this algorithm will perform 3-closure on the current formula, then checks whether there is any variable with implicit information. (First, check for $\emptyset$.) If so, then "guess" an assignment to that variable. If this guess leads to an unsatisfiable formula, then backtrack and "guess" the complementary assignment. In both cases the guessed assignment is added to the current formula as a new unit clause, and the algorithm is called recursively. The choice of a guessing variable with the greatest implicit information brings the formula closer to one whose 3-closure is its closure.

5. Partial Horn Renamability in CNF

   We will investigate whether variable flipping in a CNF formula can speed up satisfiability testing by making a substantial fraction of the clauses into Horn clauses, which has efficient algorithms.

3 Statistical Inference Methods

We wish to investigate the statistical techniques that would allow us to make probability statements on the accuracy of the inferences made from the experimental results. Often there are very little known assumptions regarding the underlying population from which our sample is drawn. In such cases we may apply nonparametric techniques in making estimations from the test statistics and in drawing some statistically meaningful conclusions from the observations produced by the experiment.

There are some interesting questions that can be raised regarding the satisfiability testing.

- What can you say about the different algorithms run on the same sample. (and with respect to what measurement)
- What can you say about the difficulty of the different classes of input formulas.

References

[APT79] B. Aspvall, M. Plass, and R. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8:121–123, 1979.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the Association for Computing Machinery*, 5:394–397, 1962.

[DP60] M. Davis and H. Putman. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[Lar92] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, pages 6–22, January 1992.

[LT92] T. Larrabee and Y. Tsuji. Evidence for a satisfiability threshold for random 3CNF formulas. Technical Report UCSC–CRL–92–42, UC Santa Cruz, October 1992.

[VG88] A. Van Gelder. A satisfiability tester for non-clausal propositional calculus. *Information and Computation*, 79:1–21, October 1988.

[VG93] A. Van Gelder. Satisfiability Testing with Less Guessing via k-Closure. *AAAI Spring Symposium on AI and NP-Hard Problems*, March 1993.