BerkMin: A Fast and Robust Sat-Solver

Evgueni Goldberg\textsuperscript{1} and Yakov Novikov\textsuperscript{2}

Abstract We describe a SAT-solver, BerkMin, that inherits such features of GRASP, SATO, and Chaff as clause recording, fast BCP, restarts, and conflict clause “aging”. At the same time BerkMin introduces a new decision making procedure and a new method of clause database management. We experimentally compare BerkMin with Chaff, the leader among SAT-solvers used in the EDA domain. Experiments show that our solver is more robust than Chaff. BerkMin solved all the instances we used in experiments including very large CNFs from a microprocessor verification benchmark suite. On the other hand, Chaff was not able to complete some instances even with the timeout limit of 16 h.

1 Introduction

Given a conjunctive normal form (CNF) $F$ specified on a set of variables $\{x_1, \ldots, x_n\}$, the satisfiability problem is to satisfy (set to 1) all the disjunctions of $F$ by some assignment of values to variables from $\{x_1, \ldots, x_n\}$. A disjunction of $F$ is also called a clause of $F$. Many problems such as ATPG [16], logic synthesis [5], equivalence checking [6, 9], and model checking [4] reduce to the satisfiability problem.

In the last decade substantial progress has been made in the development of practical SAT algorithms [1, 2, 8, 11–13, 15, 18]. All of them are search algorithms that aim at finding a satisfying assignment by variable splitting. Search algorithms of that kind are descendants of the DPLL-algorithm [7].

DPLL-algorithm can be considered as a special case of general resolution which is called tree-like resolution. It was shown in [3] that there is exponential gap between the performance of tree-like resolution and that of general resolution.

Modern SAT-solvers have made at least two steps towards general resolution trying to eliminate the drawbacks and limitations of pure tree-like resolution. First,

\textsuperscript{1}Cadence Berkeley Labs(USA), Email: egold@cadence.com

\textsuperscript{2}Academy of Sciences (Belarus), Email: nov@newman.bas-net.by
they record so called conflict clauses [15], which are implicates of the original CNF. Adding conflict clauses allows one to prune many of the branches of the search tree that are yet to be examined [1, 13, 15, 18]. The deduced implicates are just added to the current CNF which we will also refer to as (clause) database.

Second, some of the state-of-the-art SAT-solvers use the strategy of restarts when the SAT-solver abandons the current search tree (without completing it) and starts a new one. So instead of one complete search tree the SAT-solver constructs a set of incomplete (except the last one) trees. In [1, 10] the usefulness of restarts was proven experimentally. Restarts are effectively used in Chaff [13].

We introduce a new SAT solver called BerkMin (BerkMin stands for Berkeley–Minsk, the cities where the authors live). BerkMin can be considered as the next representative of the family of SAT-solvers that includes GRASP [15], SATO [18], Chaff [13]. BerkMin uses the procedures of conflict analysis and nonchronological backtracking introduced in GRASP, fast BCP suggested in SATO, and Chaff’s idea of reducing the contribution of “aged” conflict clauses into decision making. Besides, BerkMin uses restarts.

At the same time BerkMin introduces many new features into decision-making and clause-database management. First, the set of conflict clauses is organized as a chronologically ordered stack (the top clause is the one deduced the last). If in the current node of the search tree there are unsatisfied conflict clauses, the next branching variable is chosen among the free variables, whose literals are in the top unsatisfied conflict clause. Second, we introduce a heuristic to pick which out of two possible assignments to the chosen branching variable should be examined first. In the case of using a single search tree, spending time on the selection of the branch to be examined first makes sense only for satisfiable CNFs. (For unsatisfiable CNFs both branches are “symmetric” i.e. the search tree size is not affected by whichever branch is examined first.) However, when using restarts the symmetry between the two alternative branches is broken even for unsatisfiable CNFs.

Third, our procedure of computing the activity of variables is different from that of Chaff. The activity of variables in conflict making is used by Chaff to single out good candidates for branching variables. For computing the activity of a variable $x$, Chaff counts the number of occurrences of $x$ in conflict clauses. This may lead to overlooking some variables that do not appear in conflict clauses while actively contributing to conflicts (e.g. if these variables are deduced). BerkMin solves this problem by taking into account a wider set of clauses involved in conflict making.

Fourth, we use a new procedure of clause database management performed after the current search tree is abandoned. The novelty of the procedure is that the decision whether a database clause should be removed is based not only on its size (the number of literals). It is also based on the “activity” of this clause in conflict making and its “age”.

We experimentally compare the performance of BerkMin with that of Chaff that is currently considered as the best SAT-solver used in the EDA domain. Experiments clearly show that BerkMin is more robust than Chaff. By greater robustness of BerkMin we mean that it is able to solve more instances than Chaff in a reasonable