3 Extending DPLL for Pseudo-Boolean Constraints

As many verification and design automation problems concerning finite state hardware and software systems can be expressed as satisfiability problems for propositional logics, there is a growing demand for efficient satisfiability checkers for such logics. Application domains include combinational and sequential equivalence checking for circuits, test pattern generation, and bounded model checking. More recently, SAT solvers have become essential components in SMT solvers, where they are coupled with solvers for conjunctions of constraints over some background theory (e.g. the theory of linear arithmetic over the reals) in order to solve arbitrary Boolean combinations of such constraints. SAT solvers thus take a key role in technologies which help to master the complexity of ever larger circuits and ever more refined embedded software, which has sparked much research on enhancing their capabilities.

Concerning performance, the most dramatic improvements have been achieved on SAT solvers for conjunctive normal forms (CNFs) that implement variants of the DPLL procedure. As in the classical DPLL procedure, the main algorithmic ingredients of these solvers are unit propagation and backtrack search. These have, however, been enhanced by heuristics for finding a suitable traversal sequence through the backtrack search tree, as well as by refined algorithms and data structures for pruning the search tree and for accelerating unit propagation. Considerable search tree pruning has been achieved through non-chronological backtracking [68, 112, 123, 91] and conflict-driven learning [112, 123, 91], usually combined with random or periodic restarts [9, 91]. Unit propagation is sped up through dedicated data structures [124, 123] and through lazy clause evaluation [91], which delays re-evaluation of the truth value of any clause that is definitely non-unit.

While these techniques actually yield a dramatic speedup in practice, now tackling instances with hundreds of thousands of propositions, which would have been completely impractical a decade ago, they still reach their limits for state-of-the-art verification problems derived from high-level design models (e.g., STATEMATE models [70]) of embedded software. Such models easily yield CNFs with millions of proposi-
tional variables under bounded model checking. While some of this complexity is inherent to the verification task, another part can be attributed to the low expressiveness of conjunctive normal forms. Apart from improving CNF solvers performance-wise, another line of research is therefore aiming at extending such procedures to handle other, more expressive types of constraints.

In this chapter we will study how to extend a DPLL-based SAT solver to efficiently solve conjunctions of pseudo-Boolean constraints. Compared to clauses of a CNF, pseudo-Boolean constraints, i.e. linear inequalities on Boolean variables, allow a significantly more compact description of many discrete problems. Pseudo-Boolean constraints arise naturally in many application domains and have been used to efficiently encode problems from electronic design automation, formal verification, and operations research. Pseudo-Boolean problems have usually been handled by integer linear programming (ILP) solvers. The drawback is that the latter do not take into account the Boolean nature of the problem and thus cannot apply the specialized methods exploited in SAT checkers. In principle, pseudo-Boolean constraints can be encoded as pure CNF formulae and be solved by a conventional SAT solver. However a naive conversion of a pseudo-Boolean constraint into CNF can require an exponential number of clauses, thus preventing the solver from effectively processing the search space. Actually, this increase in problem size can be reduced from exponential to linear by introducing auxiliary variables, however leading to a worst-case exponential blow-up in the size of the backtrack search tree.

It has been observed by Barth [13] that the DPLL procedure can easily be modified to handle pseudo-Boolean constraints directly, although this clearly introduces some overhead in the satisfiability engine caused by the more complex data structures it uses for reasoning. Whittemore, Kim, and Sakallah tried to follow up on the advances in the algorithmics of CNF-SAT solvers by adapting GRASP’s conflict analysis and learning to zero-one linear constraint systems [120], and Aloul, Ramani, Markov, and Sakallah ported CHAFF’s lazy clause evaluation to this setting [3]. Yet, they simply mimicked CHAFF’s lazy evaluation scheme such that their type of lazy clause evaluation is confined to the pure CNF part of the problem, i.e. applies only to those clauses that are disjunctive. As all other clauses are evaluated eagerly, and as clause re-evaluation is known to account for the major part of the runtime of a DPLL-based SAT solver [91], this is far from optimal. In this chapter, we will show that it is possible and effective to generalize lazy clause evaluation to arbitrary linear constraints under a zero-one interpretation. Closely related techniques have been independently devised and implemented by Chai and Kuehlmann [32].