Eliminating Redundant Clauses in SAT Instances

Olivier Fourdrinoy, Éric Grégoire, Bertrand Mazure, and Lakhdar Saïs

CRIL CNRS & IRCICA
Université d'Artois
Rue Jean Souvraz SP18
F-62307 Lens Cedex France
{fourdrinoy,gregoire,mazure,sais}@cril.fr

Abstract. In this paper, we investigate to which extent the elimination of a class of redundant clauses in SAT instances could improve the efficiency of modern satisfiability provers. Since testing whether a SAT instance does not contain any redundant clause is NP-complete, a logically incomplete but polynomial-time procedure to remove redundant clauses is proposed as a pre-treatment of SAT solvers. It relies on the use of the linear-time unit propagation technique and often allows for significant performance improvements of the subsequent satisfiability checking procedure for really difficult real-world instances.

1 Introduction

The SAT problem, namely the issue of checking whether a set of Boolean clauses is satisfiable or not, is a central issue in many computer science and artificial intelligence domains, like e.g. theorem proving, planning, non-monotonic reasoning, VLSI correctness checking and knowledge-bases verification and validation. These last two decades, many approaches have been proposed to solve hard SAT instances, based on -logically complete or not- algorithms. Both local-search techniques (e.g. [1]) and elaborate variants of the Davis-Putnam-Loveland-Logemann's DPLL procedure [2] (e.g. [3,4]) can now solve many families of hard huge SAT instances from real-world applications.

Recently, several authors have focused on detecting possible hidden structural information inside real-world SAT instances (e.g backbones [5], backdoors [6], equivalences [7] and functional dependencies [8]), allowing to explain and improve the efficiency of SAT solvers on large real-world hard instances. Especially, the conjunctive normal form (CNF) encoding can conduct the structural information of the initial problem to be hidden [8]. More generally, it appears that many real-world SAT instances contain redundant information that can be safely removed in the sense that equivalent but easier to solve instances could be generated.

In this paper, we investigate to which extent the elimination of a class of redundant clauses in real-world SAT instances could improve the efficiency of modern satisfiability provers. A redundant clause is a clause that can be removed from the instance while keeping the ability to derive it from the remaining part of the instance. Since testing whether a SAT instance does not contain any redundant clause is NP-complete [7], an incomplete but polynomial-time procedure to remove redundant clauses is proposed as a pre-treatment of SAT solvers. It relies on the use of the linear time unit propagation technique. Interestingly, we show that it often allows for significant performance improvements of the subsequent satisfiability checking procedure for hard real-world instances.

The rest of the paper is organized as follows. After basic logical and SAT-related concepts are provided in Section 2, Section 3 focuses on redundancy in SAT instances, and the concept of redundancy modulo unit propagation is developed. In Section 4, our experimental studies are presented and analyzed. Related works are discussed in Section 5 before conclusive remarks and prospective future research works are given in the last Section.

2 Technical Background

Let $\mathcal{L}$ be a standard Boolean logical language built on a finite set of Boolean variables, denoted x, y, etc. Formulas will be denoted using letters such as c. Sets of formulas will be represented using Greek letters like Γ or Σ. An interpretation is a truth assignment function that assigns values from $\{true, false\}$ to every Boolean variable. A formula is consistent or satisfiable when there is at least one interpretation that satisfies it, i.e. that makes it become $true$. Such an interpretation is called a model of the instance. An interpretation will be denoted by upper-case letters like I and will be represented by the set of literals that it satisfies. Actually, any formula in $\mathcal{L}$ can be represented (while preserving satisfiability) using a set (interpreted as a conjunction) of clauses, where a clause is a finite disjunction of literals, where a literal is a Boolean variable that can be negated. A clause will also be represented by the set formed with its literals. Accordingly, the size of a clause c is given by the number of literals that it contains, and is noted $|c|$.

SAT is the NP-complete problem [9] that consists in checking whether a finite set of Boolean clauses of $\mathcal{L}$ is satisfiable or not, i.e. whether there exists an interpretation that satisfies all clauses in the set or not.

Logical entailment will be noted using the $\models$ symbol: let c be a clause of $\mathcal{L}$, $\Sigma \models c$ iff c is $true$ in all models of Σ. The empty clause will represent inconsistency and is noted $\bot$.

In the following, we often refer to the binary and Horn fragments of $\mathcal{L}$ for which the SAT issue can be solved in polynomial time [10,11,12]. A binary clause is a clause formed with at most two literals whereas a Horn clause is a clause containing at most one positive literal. A unit clause is a single literal clause.

In this paper, the *Unit Propagation* algorithm (in short UP) plays a central role. UP recursively simplifies a SAT instance by propagating -throughout the instance- the truth-value of unit clauses whose variables have been already assigned, as shown in algorithm 1.

We define entailment modulo Unit Propagation, noted $\models_{UP}$, the entailment relationship $\models$ restricted to the Unit Propagation technique.

Definition 1. *Let Σ be a SAT instance and c be a clause of $\mathcal{L}$, $\Sigma \models_{UP} c$ if and only if $\Sigma \wedge \neg c \models_{UP} \bot$ if and only if $UP(\Sigma \wedge \neg c)$ contains an empty clause.*

Clearly, $\models_{UP}$ is logically incomplete. It can be checked in polynomial time since UP is a linear-time process. Let c_1 and c_2 be two clauses of $\mathcal{L}$. When $c_1 \subseteq c_2$, we have that