SMT SOLVING BIT-VECTOR FORMULAS

MARTIN JONÁŠ

PhD Thesis Proposal

September 2016

The struggle itself toward the heights
is enough to fill a man's heart.
One must imagine Sisyphus happy.

— Albert Camus, The Myth of Sisyphus

ACKNOWLEDGMENTS

I would like to thank to ...

CONTENTS

1 INTRODUCTION

2 STATE OF THE ART

2.1 PRELIMINARIES

This section introduces the notation which will be used in the rest of this chapter. The exposition of propositional logic is mainly based on the work of Nieuwenhuis et al. [NOT06]. The exposition of first-order logic is partly based on the same source, but the definition of first-order theory is different. In particular, instead of defining theory as a set of first-order sentences, the theory will be defined as a set of first-order *structures*, as this definition is more suitable for the bit-vector theory, which will be introduced in the later parts of this chapter.

2.1.1 *Propositional formulas, assignments, and satisfaction*

Let $\mathcal{P}$ be a fixed finite set of propositional variables. For every variable $x \in \mathcal{P}$ there are two literals – a *positive literal* x and a *negative literal* $\overline{x}$. For a given literal l, we define $\neg l$ as $\overline{l}$ if l is positive and as l if l is negative. Literals l and $\neg l$ are called *complementary*. A *clause* is a finite disjunction of of literals. The empty clause is denoted by $\bot$. A formula in the *conjunctive normal form* (CNF) is a finite conjunction of clauses. If convinient, we will use idempotence and commutativity of disjunction and view clauses as sets of literals and therefore ignore the order and multiple occurences of literals. Similarly, if convinient, we will view CNF formulas as sets of clauses.

A *partial assignment* M is a set of literals which does not contain complementary literals, i.e. $\{x, \overline{x}\} \subseteq M$ for no $x \in \mathcal{P}$. A literal l is *true* in the assignment M if $l \in M$, *false* in M if $\neg l \in M$, and *undefined* otherwise. A literal is *defined* in M if it is true or false in M. We call an asignment M *total* over $\mathcal{P}$ if all literals of $\mathcal{P}$ are defined in M. A clause is *true* in M if at least one of its literals is true in M and a CNF formula is *true* in M if all of its clauses are true in M. Clause that is false for a given assignment M is called a *conflict clause* for M. For a clause $C = x_1 \vee \ldots x_n$, the notation $\neg C$ stands for the formula $\neg x_1 \wedge \ldots \wedge x_n$.

If a formula F is true in M, we call M a *model* of F and denote it as $M \models F$. A formula is *satisfiable* if it has a model and *unsatisfiable* otherwise. If every model of a formula F is also a model of a formula F', we say that the formula F' is *entailed* by the formula F and denote it as $F \models F'$. Formulas F and F' are called *equisatisfiable* if F is satisfiable precisely if F' is satisfiable.

2.1.2 *First-order formulas and theories*

2.2 PROPOSITIONAL SATISFIABILITY

A *propositional statisfiability problem* (SAT) is for a given formula F in CNF decide wheter it is satisfiable. The restriction to formulas in CNF is without a loss of generality, as Tseitin transformation can be used to transform every formula to a equisatisfiable formula in CNF with only linear increase of its size [Tse68].

2.2.1 *DPLL*

Historically, the first procedure to solve SAT without explicitly computing the truth table of the formula was proposed by Davis and Putnam [DP60]. During the Davis–Putnam procedure (DP) the propositional variables of the input formula are successively eliminated using the resolution inference rule [Rob65]. If the resolution yields the empty clause, the formula is unsatisfiable; on the other hand, if after ellimination of all variables no clauses remain, the formula is satisfiable. The main problem of DP is its space complexity as the number of the clauses may grow exponentially even for simple formulas. To alleviate this problem, the refinement of DP algorithm was introduced in 1962 by Davis, Putnam, Logemann and Loveland [DLL62].

Davis–Putnam–Logemann–Loveland algorithm (DPLL) iteratively tries to build a satisfying assignment by searching and it backtracks if any of the input clauses becomes false in the current assignment. The search of DPLL is guided by the unit propagation (also known as boolean constraint propagation), which is based on the observation that given a clause $C \vee l$ in which all literals of C are false in the current assignment M and the literal l is undefined, the only way to build a satisfying asignment is to add the literal l to M.

As observed by Nieuwenhuis et al. [NOT06], the DPLL algorithm can be presented as a transition system. In this system, the states are Fail and pairs $M \parallel F$, where F is a CNF formula and M is a *sequence* of literals, each marked as *decision* or non-decision literal. Decision literals are denoted as $l^{\bullet}$ and intuitively correspond to literals whose value was set arbitrarily during the search, and hence their value can be changed to $\neg l$ during backtracking if necessary. We will denote a concatenation of sequences M and N by a simple juxtaposition MN and we will treat literals as sequence of length 1. A transition systems further contains a *transition relation* $\Longrightarrow$, which is a binary relation over the set of states. Instead of writing $(s,t) \in \Longrightarrow$, we will write simply $s \Longrightarrow t$. The reflexive and transitive closure of the relation $\Rightarrow$ will be denoted as $\Rightarrow^*$. The transition relation for the DPLL transition system is given by the following set of rules:

UnitPropagate

$$M \parallel F, C \vee l \implies Ml \parallel F, C \vee l \quad \text{if} \begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases}$$

PureLiteral

$$M \parallel F \implies Ml \parallel F \quad \text{if} \begin{cases} l \text{ occurs in } F \\ \neg l \text{ does not occur in } F \\ l \text{ is undefined in } M \end{cases}$$

Decide

$$M \parallel F \implies Ml^{\bullet} \parallel F \quad \text{if} \begin{cases} l \text{ or } \neg l \text{ occurs in } F \\ l \text{ is undefined in } M \end{cases}$$

Fail

$$M \parallel F, C \implies \mathsf{Fail} \quad \text{if} \begin{cases} M \models \neg C \\ M \text{ contains no decision literals} \end{cases}$$

Backtrack

$$Ml^{\bullet}N \parallel F, C \implies M\neg l \parallel F, C \quad \text{if} \begin{cases} Ml^{\bullet}N \models \neg C \\ N \text{ contains no decision literals} \end{cases}$$

A state s is called *final* if there is no state t such that $s \implies t$. It can be shown that if F is a formula and S_f an arbitrary final state such that $\emptyset \parallel F \implies^* S_f$, then F is unsatisfiable precisely if $S_f = \mathsf{Fail}$. Moreover, if $S_f = M \parallel F$, then M is a model of the formula F [NOT06].

Note that the used backtracking strategy is *chronological*, i.e. the value of the last decision made is changed.

2.2.2 *CDCL*

Although the DPLL algorithm is more efficient than the original DP algorithm, it may unnecessarily explore parts of the search space that contain no solution. This problem can be partly solved by a further refinement of

DPLL algorithm called Conflict-Driven Clause Learning (CDCL), which was proposed by Marques-Silva and Sakallah [MSS99] and is a base of almost all modern SAT solvers [KSMS11]. In addition to DPLL, CDCL based SAT solvers have mechanisms to analyze a conflict and learn a new clause from a conflicting assignment. Moreover, in contrast to a *chronological backtracking*, in which the value of the most recent decision is changed, CDCL solvers can perform *non-chronological backtracking* to an earlier decision literal.

2.3 SATISFIABILITY MODULO THEORIES

2.4 SATISFIABILITY OF QUANTIFIER-FREE BIT-VECTOR FORMULAS

2.5 SATISFIABILITY OF QUANTIFIED BIT-VECTOR FORMULAS

2.6 COMPUTATIONAL COMPLEXITY

3 AIM OF THE WORK

3.1 OBJECTIVES AND EXPECTED RESULTS

3.2 EXPECTED OUTPUTS

3.3 PROGRESSION SCHEDULE

[DLL62] Davis, M., G. Logemann, and D. W. Loveland. "A machine program for theorem-proving." In: *Commun. ACM* 5.7 (1962), pp. 394–397.

[DP60] Davis, M. and H. Putnam. "A Computing Procedure for Quantification Theory." In: *J. ACM* 7.3 (1960), pp. 201–215.

[KSMS11] Katebi, H., K. A. Sakallah, and J. P. Marques-Silva. "Empirical Study of the Anatomy of Modern Sat Solvers." In: *Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, MI, USA, June 19-22, 2011. Proceedings*. 2011, pp. 343–356.

[MSS99] Marques-Silva, J. P. and K. A. Sakallah. "GRASP: A Search Algorithm for Propositional Satisfiability." In: *IEEE Trans. Computers* 48.5 (1999), pp. 506–521.

[NOT06] Nieuwenhuis, R., A. Oliveras, and C. Tinelli. "Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T)." In: *J. ACM* 53.6 (2006), pp. 937–977.

[Rob65] Robinson, J. A. "A Machine-Oriented Logic Based on the Resolution Principle." In: *J. ACM* 12.1 (1965), pp. 23–41.

[Tse68] Tseitin, G. S. "On the complexity of derivations in the propositional calculus." In: *Studies in Mathematics and Mathematical Logic* Part II (1968), pp. 115–125.

A APPENDIX