Solving QBF Instances With Nested SAT Solvers

Bart Bogaerts and **Tomi Janhunen** and **Shahab Tasharrofi**
Helsinki Institute for Information Technology HIIT
Department of Computer Science
Aalto University, FI-00076 AALTO, Finland

Abstract

We present a new approach towards solving *quantified Boolean formulas* (QBFs) using nested SAT solvers with lazy clause generation. The approach has been implemented on top of the Glucose solver by adding mechanisms for nesting solvers as well as clause learning. Our preliminary experiments show that nested SAT solving performs (out of the box) relatively well on QBF, when taking into account that no particular QBF-oriented solving techniques were incorporated. The most important contribution of this work is that it provides a systematic way of lifting advances in SAT solvers to QBFs with low implementation effort.

1 Introduction

Since the addition of conflict-driven clause learning (Marques-Silva and Sakallah 1999), SAT solvers have made huge leaps forward in two respects: their *popularity* for tackling real-life problems and their *efficiency*. Now that these highly-performant SAT-solvers exist, research often stretches *beyond SAT*, either because of trying to tackle problems of a complexity higher than NP or because the input format of SAT solvers (propositional logic) is too limited to concisely and naturally express certain domain specific constraints, such as graph properties. For this reason, researchers have extended the SAT language with new language constructs, sometimes also called *constraints*, and have extended SAT solvers with dedicated *propagators* for those constraints that communicate with the underlying SAT solver through clauses. This has happened for example in the field of constraint programming (Apt 2003) in the form of solvers with lazy clause generation (Ohrimenko, Stuckey, and Codish 2009), in SAT modulo theories (Barrett et al. 2009) in the form of DPLL(T) solvers (Ganzinger et al. 2004), and in answer set programming (Marek and Truszczyński 1999) where all modern solvers use this architecture (Gebser, Kaufmann, and Schaub 2012; 2013; De Cat et al. 2013; Alviano et al. 2015). Several further extensions to SAT use the same approach (Gebser, Janhunen, and Rintanen 2014; Bayless et al. 2015). Such extensions may (1) increase complexity (for applications in which tasks of a higher complexity than NP are required to be tackled), or (2) remain in NP but either improve an encoding's readability and succinctness, or enhance the strength of propagation via specialized propagators.

Recent work by Janhunen, Tasharrofi, and Ternovska (2016) started from the following observation: "if SAT solvers are this efficient, then why not to use a SAT solver as a smart (in the sense that it learns *good* clauses) oracle for a SAT solver itself?". The idea here is to solve satisfiability problems for theories of the form

$$\mathcal{T} = \varphi \wedge \neg\exists\overline{x} : \psi,$$

where $\overline{x}$ is a sequence of propositional variables and φ and ψ are CNF-theories. Propagation for $\mathcal{T}$ combines unit propagation on φ with an oracle call to a SAT solver for ψ. From the result of this oracle call, a learned clause is generated and added to φ.[1] Advantages of this approach are manifold: **(1)** It is a modular approach that allows plugging in any SAT solver as innermost solver and only requires minor modifications to the outermost solver, hence progress in SAT will automatically translate to solvers of this richer formalism; **(2)** It can be immediately combined with other SAT extensions (such as integer variables, acyclicity, or any other theory propagator); **(3)** No dedicated propagators need to be developed for the new extension because the nested solver is (automatically) used as a propagator for its internal theory; for example, it was shown by Janhunen, Tasharrofi, and Ternovska (2016) how using an internal SAT solver to propagate reachability constraints leads to a simple encoding of Hamiltonian paths that performs much better when compared to a direct encoding (i.e., a SAT encoding without second-order structure). A solver, called SAT-TO-SAT, that implements this idea was presented (Janhunen, Tasharrofi, and Ternovska 2016).

Since in principle *any* solver[2] can be nested, it is also possible to nest SAT-TO-SAT in itself, as an oracle. By allowing such arbitrarily deep nesting, we essentially obtain a QBF solver. This paper presents how SAT-TO-SAT can be used for QBF solving. We evaluate how this technique performs with respect to state-of-the art QBF solvers and conclude that SAT-TO-SAT is still slower than the best QBF solver around. However, it deserves to be mentioned that our implementation is generic and performs no optimizations

[1] See Section 3 for a detailed explanation.

[2] Any solver that respects the interface given in Definition 3.3.

designed for QBF specifically. Furthermore, our current implementation is built on the popular SAT solver GLUCOSE (Audemard and Simon 2009). In principle, any SAT solver can be plugged in, resulting in a strongly improved performance.

The ideas presented here also shed new light on techniques used in QBF. For example, conflict-driven clause-learning and solution-driven cube-learning (Giunchiglia, Narizzano, and Tacchella 2002; Letz 2002; Zhang and Malik 2002; Chu and Stuckey 2014) are completely unified in our framework as clause-learning occurring in even (respectively odd) levels of nesting depth.

The main contributions of this paper are as follows: **(1)** We show how SAT-TO-SAT can be extended to a QBF-solver. This results in a principled, low-cost way to transfer *improvements* from SAT to QBF; **(2)** Furthermore, since the nesting idea is completely orthogonal to other language extensions, we can also lift *extensions* of SAT to QBF, for example resulting in QBF *modulo theories* (QBF(T)) or solvers for QBF *modulo acyclicity* (Acyc-QBF).

2 Background

Vocabularies and Interpretations. A *vocabulary* is a set of symbols, also called *atoms*; we use σ, τ, ν to refer to vocabularies. If σ is a vocabulary, a (two-valued) σ-interpretation is a mapping $\sigma \to \{\mathbf{t}, \mathbf{f}\}$ where $\mathbf{t}$ denotes *true* and $\mathbf{f}$ *false*. A *partial σ-interpretation* is a mapping $\sigma \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$, where $\mathbf{u}$ denotes unknown. We often identify a partial σ-interpretation J with a set of tuples p^v with $p \in \sigma, v \in \{\mathbf{t}, \mathbf{f}\}$ (with each atom occurring at most once), meaning that J sends all atoms occurring in this set to their corresponding values, and all others to unknown. This allows us to define the "union" of two interpretation. E.g., if σ and τ are disjoint, I is a (partial) σ-interpretation and J a (partial) τ-interpretation, we use $I \cup J$ to interpret symbols in σ in the same way as I and symbols in τ in the same way as J. If I and J are two σ-interpretations, we will use the expression $I \cup J$ only if $I \cup J$ indeed defines a partial interpretation (i.e., contains not both $p^{\mathbf{t}}$ and $p^{\mathbf{f}}$). The truth order $<_t$ on truth values is induced by $\mathbf{f} <_t \mathbf{u} <_t \mathbf{t}$. The precision order $<_p$ on truth values is induced by $\mathbf{u} <_p \mathbf{t}, \mathbf{u} <_p \mathbf{f}$. This order is extended pointwise to partial interpretations: $I <_p J$ if $I(q) <_p J(q)$ for all q in σ. If I is a σ-interpretation and $\sigma \subseteq \tau$, any (partial) σ-interpretation is identified with the partial τ-interpretation equal to I on σ and mapping all symbols in $\tau \setminus \sigma$ to $\mathbf{u}$.

Formulas. A *propositional formula* is recursively built from propositional atoms $p, q, r, \ldots$ using connectives $\wedge$, $\vee$ and $\neg$. A propositional formula is a *σ-formula* if its atoms are in σ. A *literal* is an atom or its negation. A *clause* is a disjunction of literals. A CNF is a conjunction of clauses. A sub-formula occurs *positively* (resp. *negatively*) if it is within the scope of an even (resp. odd) number of negations.

A *quantified Boolean formula* (QBF) is built using the same recursive rules, but with added quantifiers $\forall$ and $\exists$ to quantify over propositional atoms. A σ-QBF is a QBF with free symbols belonging to σ. A σ-QBF with $\sigma = \{\}$ is called a QBF sentence. We use $\exists \tau : \varphi$ to abbreviate $\exists p_1 \ldots \exists p_n : \varphi$ if $\tau = \{p_1, \ldots, p_n\}$. If φ is a propositional formula, we use $\varphi(\sigma)$ to denote that the free symbols of φ are all in σ, i.e., that φ is a σ-QBF. A *prenex* QBF is a QBF in which all quantifiers are in the front, i.e., a set of quantifiers followed by a propositional formula. The QDIMACS format is a numerical format to describe a prenex QBF in which the propositional formula is a CNF formula. The format is the de-facto standard for representing QBF instances.

Satisfiability Relation for QBFs. The satisfiability relation between σ-interpretations I and a σ-QBFs φ, denoted by $I \models \varphi$, is defined recursively in the standard way:

- $I \models p$ if $I(p) = \mathbf{t}$.
- $I \models \neg\varphi$ if $I \not\models \varphi$;
- $I \models \varphi \wedge \varphi'$ (resp. $I \models \varphi \vee \varphi'$) if $I \models \varphi$ and (resp. or) $I \models \varphi'$;
- $I \models \forall x : \varphi$ (resp. $I \models \exists x : \varphi$) if $(I \cup \{x^{\mathbf{t}}\}) \models \varphi$ and (resp. or) $(I \cup \{x^{\mathbf{f}}\}) \models \varphi$.

Let I be a (partial) σ-interpretation. We call a σ-QBF φ *I-satisfiable* if there exists a model of φ more precise than I and *I-unsatisfiable* otherwise.

3 SAT-TO-SAT

In order to discuss the working of SAT-TO-SAT, we first present a formalisation of SAT-solvers for our purposes.

SAT solving. The principal goal of a SAT solver is to find a model for a CNF, i.e., to check the validity of a formula $\exists \nu : \varphi$, where φ is a CNF. Many modern SAT solvers do more than that: they explain their answer in terms of a set of so-called *assumptions* (Nadel and Ryvchin 2012). In this text, we assume[3] that ν is the disjoint union of two vocabularies σ and τ, an *assumption vocabulary* σ and an internal vocabulary τ. A solver not only returns a model or `UNSAT` but also explains this in terms of the assumptions.

Definition 3.1 (Explaining Satisfiability)**.** Let φ be any ν-formula and $\nu = \sigma \cup \tau$ where σ and τ are disjoint. Let J be a partial σ-interpretation and M be a partial τ-interpretation. We say that (J, M) *explains* the *satisfiability* of φ if each ν-interpretation more precise than $J \cup M$ is a model of φ

Example 3.2. Let $\sigma = \{o, p, q\}$, $\tau = \{r\}$ and

$$\psi_1 = (p \vee q \vee r) \wedge (\neg o \vee \neg r).$$

Furthermore, let J be the partial σ-interpretation $\{p^{\mathbf{t}}\}$ and M the τ-interpretation $\{r^{\mathbf{f}}\}$. In this case (J, M) explains the satisfiability of ψ_1. Indeed, J guarantees that the first clause of ψ_1 is satisfied, while M guarantees that the second is. □

Definition 3.3 (SAT-solver)**.** Suppose that $\nu = \sigma \cup \tau$. A *SAT-solver* is a procedure that takes as input a ν-CNF $\mathcal{T}$ and a two-valued σ-interpretation I.

- If $\mathcal{T}$ is I-satisfiable, it returns $(\texttt{SAT}, J, M)$ such that $J \leq_p I$ and (J, M) explains the satisfiability of φ.

[3] This assumption is not vital but simplifies the presentation

- Otherwise, it returns $(\texttt{UNSAT}, J)$ where $J \leq_p I$ is such that $\mathcal{T}$ is J-unsatisfiable.

Hence a SAT solver solves the problem $\exists\tau : \mathcal{T}$ under assumptions I. Note that in formalisations of SAT solvers, often a J that explains the answer of the solver is not present. In this case, J can always be equal to I. Several SAT solvers, such as MiniSAT (Eén and Sörensson 2003), support smart reasoning methods to generate better (less precise) J.

In order to solve problems of form $\exists\nu : \mathcal{T}$, state-of-the-art SAT solvers use the conflict-driven clause learning (CDCL) algorithm (Silva, Lynce, and Malik 2009). The CDCL algorithm works by maintaining a state that represents a partial ν-interpretation. We use $\mathfrak{S}(S)$ to denote the state of a solver S. The algorithm uses operations of propagation, decision, backjumping and restart to manipulate its state. Propagation takes a state $\mathfrak{S}(S)$ and either returns a (possibly) more precise state that is the consequence of its previous state or returns a conflict clause showing no model can extend the current state. The decision operation takes a non-conflicting state $\mathfrak{S}(S)$ and branches the search on a variable v (decision variable) that is currently unassigned in $\mathfrak{S}(S)$. Backjumping takes a conflicting state $\mathfrak{S}(S)$, learns a clause from it and returns to a less precise non-conflicting state. The restart operation restarts the search while remembering learnt clauses.

SAT-TO-SAT. Recently, Janhunen, Tasharrofi, and Ternovska (2016) introduced SAT-TO-SAT, a framework for combining SAT solvers so that, together, they solve ∃∀QBF problems. Essentially, this framework performs lazy clause generation (Ohrimenko, Stuckey, and Codish 2009) where clauses are obtained from calls to another SAT solver. In this section, we recall how SAT-TO-SAT works in a slightly generalised setting.

The input for SAT-TO-SAT is an ∃∀QBF of the form

$$\mathcal{T} = \exists\sigma : \varphi \wedge (\neg\exists\tau_1 : \psi_1) \wedge \cdots \wedge (\neg\exists\tau_n : \psi_n),$$

where φ is a σ-CNF and the ψ_i's are ν_i-CNFs with $\nu_i = \sigma \cup \tau_i$. Without loss of generality, from now on, we assume that $n = 1$ and use τ for τ_1, ψ for ψ_1 and ν for $\sigma \cup \tau$. SAT-TO-SAT checks validity of $\mathcal{T}$, i.e., it returns SAT iff there exists a σ-interpretation I that satisfies φ such that ψ is I-unsatisfiable and returns UNSAT otherwise. To explain how SAT-TO-SAT works, we need some terminology:

Definition 3.4 (Lowerbound/Upperbound Mapping)**.** The *LU-mapping of vocabulary* σ is $\sigma_{lu} = \{p_u \mid p \in \sigma\} \cup \{p_l \mid p \in \sigma\}$ with p_u (resp. p_l) representing upper- (resp. lower-) bound of p. The *LU-mapping* of a partial interpretation I, denoted as I_{lu}, is a 2-valued σ_{lu}-interpretation so that $I_{lu}(p_u) = \mathbf{t}$ if and only if $I(p) \neq \mathbf{f}$ and $I_{lu}(p_l) = \mathbf{t}$ if and only if $I(p) = \mathbf{t}$.

Note that for each atom p in the vocabulary σ, I_{lu} satisfies $I_{lu}(p_l) \leq_t I(p) \leq_t I_{lu}(p_u)$, i.e., p_l (respectively p_u) is a *lower* (respectively *upper*) bound on the truth of p.

Definition 3.5 (σ-under-approximation)**.** Let ψ be a $\sigma \cup \tau$ formula. A *σ-under-approximation* of ψ is any $\sigma_{lu} \cup \tau$-formula η such that for all interpretations I:

(1) if I is a two-valued σ-interpretation, then $\exists\tau : \eta$ is satisfied in I_{lu} iff $\exists\tau : \psi$ is satisfied in I, and

(2) if I is a partial σ-interpretation, then $I_{lu} \models \exists\tau : \eta$ implies that every two-valued σ-interpretation more precise than I satisfies $\exists\tau : \psi$.

The first condition guarantees that in two-valued interpretations the approximation coincides with the original formula. The second states that if I_{lu} can be expanded to a model of η, then I can be expanded to a model of ψ.

Example 3.6 (Example 3.2 continued)**.** Let

$$\eta_1 = (p_l \vee q_l \vee r) \wedge (\neg o_u \vee \neg r).$$

In this case η_1 is a σ-under-approximation of ψ_1. We show this for some partial σ-interpretations.

- When I is 2-valued, $I_{lu}(x_l) = I(x) = I_{lu}(x_u)$ for all $x \in \sigma$. Hence Condition (1) in Definition 3.5 is satisfied.
- Let I_0 be the partial σ-interpretation that maps all atoms to $\mathbf{u}$. In this case, η_1 is $(I_0)_{lu}$-unsatisfiable, hence Condition (2) in Definition 3.5 is clearly satisfied as well.
- Now, let I_1 be the partial σ-interpretation $\{p^{\mathbf{t}}\}$. Since $(I_1)_{lu}(p_l) = \mathbf{t}$, $M = \{r^{\mathbf{t}}\}$ is a model of $\exists\tau : \eta_1$. For each two-valued interpretation $I \geq_p I_1$, it holds that $I \cup M$ is a model of the original formula ψ_1. □

Assuming a σ-under-approximation η for ψ, the SAT-TO-SAT algorithm instantiates two CDCL-solvers S_φ (tasked with solving φ) and S_η (tasked with solving η). After each unit propagation phase of S_φ, solver S_η is called with assumptions $\mathfrak{S}(S_\varphi)_{lu}$.

- If S_η returns $(\texttt{SAT}, J, M)$, it then follows from the fact that η is a σ-under-approximation of ψ that $\neg\exists\tau : \psi$ (and hence also $\mathcal{T}$) is I-unsatisfiable. In this case, J is used to create a clause that falsifies S_φ's current assignment; this clause is added to φ.
- If S_η returns $(\texttt{UNSAT}, J)$, nothing can be concluded. Literals in J are used as watched literals to avoid calling S_η again as long as $\mathfrak{S}(S_\varphi)_{lu}$ is more precise than J.

The use of the under-approximation has the effect that if $\mathfrak{S}(S_\varphi)$ is not exact, the call to the nested solver S_η remains sound in the sense that whenever S_η finds a model, a conflict clause can be added to φ. In case S_η is unsatisfiable (with the given assumption), nothing final can be concluded yet. In this case, the explanation of unsatisfiability can be used to avoid calling the nested solver too often. The use of the under-approximation roughly has the same effect as calling a SAT-solver for ψ, with assumptions $\mathfrak{S}(S_\varphi)$, but obliging the nested solver to keep all variables in σ that are unassigned in $\mathfrak{S}(S_\varphi)$ unassigned. This would require us to modify the internals of the solver S_η to find models in a partial context, which is something we try to avoid. As such, the under-approximation serves two purposes **(1)** it allows us to call the nested solver after each unit propagation, **(2)** it ensures we can use the nested SAT solver as a blackbox.

Example 3.7 (Example 3.6 continued)**.** Let $\mathcal{T}$ be the QBF

$$\varphi_1 \wedge \exists\tau : \psi_1,$$

where $\varphi_1 = \neg p \vee \neg q$. SAT-TO-SAT solves the satisfiability task for $\mathcal{T}$ as follows:

- SAT-TO-SAT starts from the partial σ-interpretation I_0 in which all atoms in σ are unknown. In this case, as

shown in Example 3.6, η_1 is unsatisfiable. A SAT-solver for η_1 can return $(\texttt{UNSAT}, \{p_l^{\mathbf{f}}, q_l^{\mathbf{f}}, o_u^{\mathbf{t}}\})$. SAT-TO-SAT interprets this result by watching literals $\neg p, \neg q$, and o. As soon as one of these literals becomes false, the sub-solver will be called again.

- SAT-TO-SAT chooses $I_1 = \{p^{\mathbf{f}}\}$. None of the watches fire hence the internal solver is not called.
- SAT-TO-SAT chooses $I_2 = \{p^{\mathbf{f}}, q^{\mathbf{t}}\}$. In this case η_1 has a model ($M = \{r^{\mathbf{f}}\}$) more precise than $(I_2)_{lu}$. The internal solver returns $(\texttt{SAT}, \{q_l^{\mathbf{t}}\}, M)$. SAT-TO-SAT interprets this result by adding the clause $\neg q$ to φ_1.
- The solver for φ_1 now finds a conflict, backjumps and continues search. □

The only thing that remains to be explained is how SAT-TO-SAT obtains a σ-approximation of ψ. This is done by the following syntactical transformation.

Lemma 3.8. *Let ψ be a $\sigma \cup \tau$-CNF. Let ψ_{lu} be the $\sigma_{lu} \cup \tau$-CNF obtained from ψ by*
1. replacing each literal $\neg p$ in ψ with $p \in \sigma$ by $\neg p_u$, and
2. replacing each literal p in ψ with $p \in \sigma$ by p_l.
Then, ψ_{lu} is a σ-under-approximation of ψ.

This lemma follows the fact that, for each partial σ-interpretation I, we have $I_{lu}(p_l) \leq_t I(p) \leq_t I_{lu}(p_u)$.

Example 3.9 (Example 3.6 continued)**.** The formula η_1 equals $(\psi_1)_{lu}$ as defined in Lemma 3.8.

4 Solving QBF With SAT-TO-SAT

The previous section discussed how SAT-TO-SAT solves $\exists\forall$QBF validity problems. These ideas easily generalise to QBF validity problems: instead of nesting a SAT-solver in another SAT-solver, we can nest SAT-TO-SAT inside a SAT-solver (recursively). In order to do this, two obstacles are to be overcome. First, we must extend SAT-TO-SAT so that it not only outputs SAT or UNSAT, but also explains this result in terms of assumptions given to it, i.e., it should respect the interface we specified in Definition 3.3. Second, it is necessary to define how an approximation of a QBF theory can be obtained, i.e., extend Lemma 3.8 to general QBFs. For the first, we use the following theorem.

Theorem 4.1. *Let $\mathcal{T}$ be a* QBF *of the form*

$$\mathcal{T}(\nu) = \exists\sigma : \varphi \wedge (\neg\exists\tau : \psi),$$

where φ is a CNF and ψ is an arbitrary QBF*. Let I be a ν-interpretation. Suppose the following hold:*

- *$J_1 \leq_p I$ is a partial ν-interpretation, and M a 2-valued σ-interpretation s.t. (J_1, M) explains φ's satisfiability, and*
- *J_2 is a partial $(\nu \cup \sigma)$-interpretation, $J_2 \leq_p I \cup M_{lu}$ and ψ is J_2-unsatisfiable.*

Then, with $J = J_1 \cup J_2|_\nu$, it holds that $J \leq_p I$ and (J, M) explains the satisfiability of $\varphi \wedge (\neg\exists\tau : \psi)$.

Theorem 4.1 shows how an explanation of satisfiability of φ and one of unsatisfiability of ψ can be combined to provide an explanation of satisfiability of the combined expression. This can be directly used to extend SAT-TO-SAT's output in case of satisfiability.

Example 4.2 (Example 3.7 continued)**.** Let $\sigma_1 = \{p, o\}, \sigma_2 = \{q\}$ and

$$\mathcal{T}(\sigma_1) = \exists\sigma_2 : (\varphi_1 \wedge \exists\tau : \psi_1).$$

Furthermore, let $J_\varphi = \{p^{\mathbf{f}}\}$ and $J_\psi = \{p^{\mathbf{f}}, q^{\mathbf{f}}, o^{\mathbf{t}}\}$. In this case, J_φ explains satisfiability of φ_1 and J_ψ explains unsatisfiability of ψ_1. Theorem 4.1 shows that $J = \{p^{\mathbf{f}}, o^{\mathbf{t}}\}$ explains satisfiability of $\mathcal{T}(\sigma_1)$. □

The case of unsatisfiability is easier: every time SAT-TO-SAT finds a model for its nested expression, a clause is added to φ that invalidates the current partial interpretation. Hence, in the end, the only way for $\mathcal{T}$ to become unsatisfiable is that φ becomes unsatisfiable.

Theorem 4.3. *Let $\mathcal{T}$ be a* QBF *of the form*

$$\mathcal{T}(\nu) = \exists\sigma : \varphi \wedge (\neg\exists\tau : \psi),$$

*where φ is a CNF and ψ is an arbitrary (*QBF*) formula. Let I be a ν-interpretation. Suppose φ is J-unsatisfiable and $J \leq_p I$, then $\mathcal{T}$ is J-unsatisfiable as well.*

We now show how Lemma 3.8 extends to general QBFs.

Lemma 4.4. *Let ψ be a $\sigma \cup \tau$-QBF. Let ψ_{lu} be the $\sigma_{lu} \cup \tau$-QBF obtained from ψ by*
1. replacing each negative occurrences of $p \in \sigma$ by p_u, and
2. replacing each positive occurrences of $p \in \sigma$ by p_l.
Then, ψ_{lu} is a σ-under-approximation of ψ.

Again, this lemma follows from the fact that, for all partial σ-interpretations I, we have $I_{lu}(p_l) \leq_t I(p) \leq_t I_{lu}(p_u)$.

Lemma 4.4 shows how to obtain under-approximations for QBFs in general. Together with Theorems 4.1 and 4.3, we obtain all ingredients to extend SAT-TO-SAT for general QBFs. The resulting solver is a SAT solver S_1 extended with an oracle S_2, also an instantiation of SAT-TO-SAT. The results from calls to S_2 are used either as learnt clauses in the theory of S_1 or watches that avoid unnecessary calls to S_2. The solver works exactly as described in Section 3 except that the "black box" nested solver is now another instance of SAT-TO-SAT rather than an instance of a SAT solver.

Translating QDIMACS into SAT-TO-SAT Input

Minor syntactical differences aside, a QDIMACS specification could be fed directly to SAT-TO-SAT. The only difference is that QDIMACS supports universal quantifications, while SAT-TO-SAT supports existential quantifications under negation, a difference that can be trivially eliminated. However, contrary to QDIMACS, SAT-TO-SAT does not require theories to be in prenex normal form. When feeding QDIMACS input to SAT-TO-SAT, all clauses are in the innermost solver. During search, other solver instances will gradually learn clauses as well. However, some clauses can be pulled out to prior to any search. This will speed up search, since SAT-TO-SAT will not waste time rediscovering information that was present in the first place. Starting from a QDIMACS specification, we perform two preprocessing steps before feeding it to SAT-TO-SAT.

1. Remove tautological clauses. We remove all clauses containing both a literal and its negation from the theory.

2. Pull clauses outwards. When certain clauses only use certain variables, they can be pulled out. We iteratively apply the following lemma to obtain an equivalent theory that no longer is in prenex normal form and in which each clause is located at the "right" level.

Lemma 4.5. *Let φ denote the* QBF

$$\varphi(\sigma_0) = \exists\sigma_1 : \varphi_1 \wedge \forall\sigma_2 : \exists\sigma_3 : \varphi_3 \wedge (\psi_1(\sigma_0, \sigma_1) \vee \psi_2(\sigma_2)),$$

where the φ_i's are arbitrary formulas and the ψ_i's are clauses. If ψ_2 is no tautology, then $\varphi(\sigma_0)$ is equivalent to

$$\exists\sigma_1 : \psi_1(\sigma_0, \sigma_1) \wedge \varphi_1 \wedge \forall\sigma_2 : \exists\sigma_3 : \varphi_3.$$

5 Evaluation & Future Work

We implemented the aforementioned techniques on top of the Glucose solver (Audemard and Simon 2009). We evaluated the resulting solver on 276 instances from the QBFLIB problems suite (Giunchiglia, Narizzano, and Tacchella 2001) , namely those that were used in the latest QBF competition (QBFEVAL 2014). We compared running times of SAT-TO-SAT with GhostQ (Klieber et al. 2010), the winner of the competition on the QBFLIB track. We only compared a plain version of our solver with the plain version of GhostQ. All tests were ran with a time limit of 900 seconds on an Intel©Xeon©E5-4652 CPU clocked at 2.70GHz with 260Gb of RAM running Ubuntu 14.04LTS.

Table 1: The numbers of satisfiable and unsatisfiable instances solved by SAT-TO-SAT and GhostQ.

	SAT	UNSAT	Total
GhostQ	66	57	123
SAT-TO-SAT	28	43	71

Table 1 depicts the number of instances solved by SAT-TO-SAT and GhostQ. As can be seen, SAT-TO-SAT's performance still lags behind the best available QBF solver. The difference seems big. However, some remarks, that also define our future work directions, need to be made. Firstly, combining these results with the results from the latest competition (QBFEVAL 2014) puts SAT-TO-SAT approximately on-par with the plain version of RAReQS (Janota et al. 2012), while a version of RAReQS with QBF preprocessors ended up in the third place of this competition. Finding out which preprocessors boost SAT-TO-SAT's performance is a topic for future work. Secondly, our implementation was built on top of Glucose. It is not hard to replace Glucose with any SAT solver that implements the interface given in Definition 3.3. Related, SAT-TO-SAT allows us to lift all future improvements in SAT solving to QBF. Thirdly, our solver was not optimised for QBF solving. Several optimisations are possible. We discuss three of them below.

(i) One particularly useful optimisation would be to detect *Tseitin variables*. QBF specifications in the QBFLIB often contain patterns of the form $\exists\sigma : \forall\tau : \exists\nu : (p \Leftrightarrow \varphi(\sigma, \tau)) \wedge \psi$, with $p \in \nu$, which is equivalent to

$$\exists\sigma : \forall\tau, p : (p \Leftrightarrow \varphi(\sigma, \tau)) \Rightarrow \exists(\nu \setminus \{p\}) : \psi.$$

These kind of observations allow us to reduce the nesting depth or to pull variables and clauses higher in the nesting hierarchy. Since SAT-TO-SAT is designed to gradually pass more and more information upwards in the hierarchy, doing this in advance could save precious time; in this case, without any memory overhead. Reverse Tseitin engineering is used for example by Goultiaeva and Bacchus (2013). Implementing such techniques is a topic for future work.

(ii) Sometimes when the internal solver in SAT-TO-SAT finds a model, there are several choices on how to construct a conflict clause. More formally, for a given model M of the internal theory, there might be multiple J's such that (J, M) explains satisfiability of the internal theory, as defined in Definition 3.1. Each of these J's gives rise to a different learned clause in the outermost solver.

Example 5.1. Let $\mathcal{T}$ be the following theory:

$$\mathcal{T} = \left\{ \begin{array}{l} \exists o, p, q : \\ \quad (\neg p \vee q) \wedge \\ \quad \neg\exists r : \\ \qquad (p \vee q \vee r) \wedge (\neg r \vee q) \wedge (o \vee \neg q \vee r) \end{array} \right\}$$

If a SAT-solver for the internal theory is called with assumptions $I = \{o^{\mathbf{t}}, p^{\mathbf{t}}, q^{\mathbf{t}}\}$, then $M = \{r^{\mathbf{f}}\}$ is a model. If $J_1 = \{o^{\mathbf{t}}, p^{\mathbf{t}}\}$ and $J_2 = \{o^{\mathbf{t}}, q^{\mathbf{t}}\}$, then both (J_1, M) and (J_2, M) explain satisfiability of $(p \vee q \vee r) \wedge (\neg r \vee q)$.

Returning the first of these would result in the addition of a clause $\neg o \vee \neg p$ to the outermost theory, while the second would result in the addition of a clause $\neg o \vee \neg q$. □

In principle, both clauses found in Example 5.1 are valid consequences and could be added to the top theory. However, there could be exponentially many such clauses. Using ideas from *extended resolution* (Tseitin 1968), we can summarise all of these clauses in linear size in terms of the original theory by introducing new variables. In the example, this would boil down to introducing a variable t and adding an encoding of the following definition $t \Leftrightarrow p \vee q$ and the clause $\neg t \vee \neg o$ to invalidate the current assignment. This generalises both of the above clauses with the cost of introducing an extra variable. We need to research the impact of such learned clauses both on time and memory consumption.

(iii) Several of the preprocessing techniques discussed here involved transforming a prenex normal form QBF into a non-prenex normal form sentence. Sometimes this even involves "rediscovering" problem structure that was probably present at the time of the encoding, but that was lost because of the low-level format used. An example of this is the case **(i)** above, where we need to do clever reasoning to rediscover that one variable is defined functionally in terms of other variables, since CNF's have no native language construct to express this definition. A richer language could directly present this information in the encoding. Hence, we intend to generalise SAT-TO-SAT to accept non-prenex and non-CNF input format such as QCIR (QBF Gallery 2014).

6 Related Work

There exist many CDCL-based algorithms to solve QBF instances. Our approach differs from most of them in three aspects. **(1)** Using under-approximations, we *circumvent a*

common limitation in QBF solving that variables must be chosen in accordance to the quantifier prefix. Thus, nested solvers can be called earlier, during the search process of a solver, before a solver has found a complete assignment. This leads to *faster propagation*. **(2)** *We treat existential and universal quantifiers symmetrically*. That is, all our algorithms are defined for theories of the form $\exists\sigma : \varphi \wedge \neg\exists\tau : \psi$, where neither the structure of ψ, nor the context in which this formula occurs matters. **(3)** Given that our approach is applicable to any SAT solver, *we gain an engineering advantage* over many existing QBF solving techniques.

The idea that using an NP oracle inside an NP-solver results in a solver for Σ_2^P (and, similarly, for the rest of polynomial hierarchy) is not new and directly follows the definition of these complexity classes. This idea has also been applied before to obtain a QBF solver (Ranjan, Tang, and Malik 2004). Our approach is different in two main ways from Ranjan, Tang, and Malik (2004): **(1)** we use of under-approximations, and **(2)** we are not limited to $2QBF$.

These ideas have been integrated with a new learning technique known as *Counterexample Guided Abstraction Refinement* (CEGAR) (Janota et al. 2012). CEGAR enables gradual expansion of a QBF instance and, in that sense, our approach is similar to CEGAR. However, unlike CEGAR, our approach generates only one solver per quantification level and maintains the states of those solvers for faster search. Recently, Rabe and Tentrup (2015) introduced a new extension of the CEGAR approach that also uses one solver per quantifier level as well as a form of clause selection. Our work is different from theirs because of our under-approximation technique and our uniform treatment of existential and universal quantifiers.

Also, recently, Janota and Marques-Silva (2015) presented a QBF solver based on *clause selection* which, again, is different from our algorithm due to our under-approximation technique and our uniform treatment of quantifiers. Another point where our approach differs from theirs relates to the *learnt clauses*. Their approach uses an *extended language of learning* where each clause is associated with variables that express whether that clause is selected (i.e., should be satisfied in lower levels) or deselected (i.e., is already satisfied) at a certain level. Learn clauses are always over these new, so-called *clause selection variables*. Their method allows to succinctly represent many clauses invalidating different J's that explain satisfiability at once (similar to the technique presented in Example 5.1). We are convinced that the extended language of learning is beneficial to QBF solving. Investigating the exact effect of this aspect on solver performance is a topic for future work.

Several QBF solvers combine CDCL with *solution-driven cube learning* (Zhang and Malik 2002; Goultiaeva, Seidl, and Biere 2013). A *cube* is a conjunction of literals; a formula is in *Disjunctive Normal Form (DNF)* if it is a disjunction of cubes. Zhang and Malik (2002) introduced Augmented CNF (ACNF): a formula is an ACNF if it is of the form $\varphi \vee \psi$ where φ is a CNF and ψ a DNF. They also introduced solution-driven cube learning: a technique where a QBF solver reasons on an ACNF and gradually adds cubes to ψ_2. Currently, many QBF solvers implement solution-driven cube learning in addition to conflict-driven clause learning. Goultiaeva, Seidl, and Biere (2013) show that dual propagation for QBF can be represented using the existing clause-learning and cube-learning methods in QBF solvers.

Let us give the outermost solver in SAT-TO-SAT level one and each nested solver level one plus its parent's level. We show that cube learning and clause learning in QBF both boil down to clause learning in SAT-TO-SAT. That is, clauses learnt in a QDPLL algorithm correspond to learnt clauses in odd levels of SAT-TO-SAT and cubes learnt in a QDPLL algorithm correspond to SAT-TO-SAT clauses in even levels. Informally, this follows easily from the fact that a cube is the negation of a clause and vice versa. Since the clauses in even levels of SAT-TO-SAT have an odd number of negations preceding them, they indeed represent a negated clause, i.e., a cube. Similarly, the even number of negations that precede a clause in an odd level of SAT-TO-SAT cancel each other out to represent a normal QBF clause.

Formally, If $\mathcal{T} = \exists\sigma_1 : \forall\sigma_2 : \exists\sigma_3 : \ldots \forall\sigma_{n-1} : \exists\sigma_n : \varphi$ is a QBF with φ in ACNF, then all learned cubes C have the property that $\mathcal{T}$ is equivalent to

$$\exists\sigma_1 : \forall\sigma_2 : \exists\sigma_3 : \ldots \forall\sigma_{n-1} : \exists\sigma_n : C \vee \varphi. \qquad (1)$$

Lemma 6.1. *Suppose $\mathcal{T} = \exists\sigma_1 : \forall\sigma_2 : \exists\sigma_3 : \ldots \forall\sigma_{n-1} : \exists\sigma_n : \varphi$ is a QBF with φ in ACNF and $C = C' \wedge C_n$ is a cube with C' a $\cup_{i<n}\sigma_i$-cube and C_n a σ_n-cube. Suppose also that $\mathcal{T}$ is equivalent to* (1)*. Then $\mathcal{T}$ is also equivalent to*

$$\exists\sigma_1 : \forall\sigma_2 : \exists\sigma_3 : \ldots \forall\sigma_{n-1} : C' \vee \exists\sigma_n : \varphi$$

and to

$$\exists\sigma_1 : \neg\exists\sigma_2 : \neg\exists\sigma_3 : \ldots \neg\exists\sigma_{n-1} : \neg C' \wedge \neg\exists\sigma_n : \varphi.$$

From Lemma 6.1, we indeed see that (since $\neg C'$ is a clause) learned cubes in QBF correspond to learned clauses in the corresponding SAT-TO-SAT specification.

7 Conclusion

This paper introduced an extension of SAT-TO-SAT with an arbitrary nesting depth and showed how it can be used to solve the validity problem of Quantified Boolean Formulas. Our experiments show that, even without any QBF-specific optimizations, SAT-TO-SAT performs relatively well on QBF instances from the latest QBF competition.

Moreover, the generic architecture of SAT-TO-SAT with respect to its underlying SAT solvers allows us to uniformly lift SAT-related optimizations to QBF. In particular, GLUCOSE, the SAT-solver SAT-TO-SAT is currently based on, can be easily replaced with other SAT solvers.

In addition, in Section 5, we have identified several topics for future work: (i) detecting when variables are completely defined in terms of variables form higher levels in order to pull larger parts of the theory to higher levels (ii) learning stronger clauses based on extended resolution when conflicts arise because the internal solver finds a model, and (iii) moving towards a richer input language.

Acknowledgments

This work is supported by the Finnish Center of Excellence in Computational Inference Research (COIN) funded by the Academy of Finland (under grant #251170).

References

Alviano, M.; Dodaro, C.; Leone, N.; and Ricca, F. 2015. Advances in WASP. In *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015. Proceedings*, 40–54.

Apt, K. R. 2003. *Principles of Constraint Programming*. Cambridge University Press.

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern SAT solvers. In Boutilier, C., ed., *IJCAI*, 399–404.

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2009. Satisfiability modulo theories. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. 825–885.

Bayless, S.; Bayless, N.; Hoos, H. H.; and Hu, A. J. 2015. SAT modulo monotonic theories. In Bonet, B., and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.*, 3702–3709. AAAI Press.

Chu, G., and Stuckey, P. J. 2014. Nested constraint programs. In O'Sullivan, B., ed., *Principles and Practice of Constraint Programming - CP 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, 240–255. Springer.

De Cat, B.; Bogaerts, B.; Devriendt, J.; and Denecker, M. 2013. Model expansion in the presence of function symbols using constraint programming. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, 2013*, 1068–1075. IEEE Computer Society.

Eén, N., and Sörensson, N. 2003. An extensible SAT-solver. In Giunchiglia, E., and Tacchella, A., eds., *SAT*, volume 2919 of *LNCS*, 502–518. Springer.

Ganzinger, H.; Hagen, G.; Nieuwenhuis, R.; Oliveras, A.; and Tinelli, C. 2004. DPLL(*T*): Fast decision procedures. In Alur, R., and Peled, D., eds., *CAV*, volume 3114 of *LNCS*, 175–188. Springer.

Gebser, M.; Janhunen, T.; and Rintanen, J. 2014. SAT modulo graphs: Acyclicity. In *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014. Proceedings*, 137–151.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.* 187:52–89.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2013. Advanced conflict-driven disjunctive answer set solving. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence.*

Giunchiglia, E.; Narizzano, M.; and Tacchella, A. 2001. Quantified Boolean formulas satisfiability library (QBFLIB). `http://www.qbflib.org`.

Giunchiglia, E.; Narizzano, M.; and Tacchella, A. 2002. Learning for quantified boolean logic satisfiability. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence.*, 649–654.

Goultiaeva, A., and Bacchus, F. 2013. Recovering and utilizing partial duality in QBF. In *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, 83–99.

Goultiaeva, A.; Seidl, M.; and Biere, A. 2013. Bridging the gap between dual propagation and cnf-based qbf solving. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, 811–814.

Janhunen, T.; Tasharrofi, S.; and Ternovska, E. 2016. SAT-TO-SAT: Declarative extension of SAT solvers with new propagators. In *To Appear in the Proceedings of 30th AAAI Conference on Artificial Intelligence (AAAI-16)*. Preprint available on `https://www.cs.sfu.ca/~sta44/personal/files/jtt-aaai-2016.pdf`.

Janota, M., and Marques-Silva, J. 2015. Solving QBF by clause selection. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015.*, 325–331.

Janota, M.; Klieber, W.; Marques-Silva, J.; and Clarke, E. M. 2012. Solving QBF with counterexample guided refinement. In *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, 2012. Proceedings*, 114–128.

Klieber, W.; Sapra, S.; Gao, S.; and Clarke, E. M. 2010. A non-prenex, non-clausal QBF solver with game-state learning. In *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, Proceedings*, 128–142.

Letz, R. 2002. Lemma and model caching in decision procedures for quantified boolean formulas. In *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2002, Proceedings*, 160–175.

Marek, V., and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In Apt, K. R.; Marek, V.; Truszczyński, M.; and Warren, D. S., eds., *The Logic Programming Paradigm: A 25-Year Perspective*. Springer-Verlag. 375–398.

Marques-Silva, J. P., and Sakallah, K. A. 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers* 48(5):506–521.

Nadel, A., and Ryvchin, V. 2012. Efficient SAT solving under assumptions. In *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, 2012. Proceedings*, 242–255.

Ohrimenko, O.; Stuckey, P. J.; and Codish, M. 2009. Propagation via lazy clause generation. *Constraints* 14(3):357–391.

QBF Gallery. 2014. QCIR-G14: A non-prenex non-CNF format for quantified boolean formulas. Technical report.

2014. QBF gallery 2014 (competition). `http://qbf.satisfiability.org/gallery/index.html`.

Rabe, M. N., and Tentrup, L. 2015. Caqe: A certifying qbf solver. In *Proceedings of the 15th Conference on Formal Methods in Computer-aided Design (FMCAD'15)*, 136–143.

Ranjan, D. P.; Tang, D.; and Malik, S. 2004. A comparative study of 2qbf algorithms. In *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, Online Proceedings*.

Silva, J. P. M.; Lynce, I.; and Malik, S. 2009. Conflict-driven clause learning SAT solvers. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press. 131–153.

Tseitin, G. S. 1968. On the complexity of derivation in the propositional calculus, *Zapiski nauchnykh seminarov*. *LOMI* 8:234–259. English translation of this volume: Studies in Constructive Mathematics and Mathematical Logic, Part 2, A. O. Slisenko, eds. Consultants Bureau, N.Y., 1970, pp. 115-125.

Zhang, L., and Malik, S. 2002. Towards a symmetric treatment of satisfaction and conflicts in quantified boolean formula evaluation. In *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Proceedings*, 200–215.