Partial Witnesses from Preprocessed Quantified Boolean Formulas

Martina Seidl
Inst. for Formal Models and Verification, JKU Linz
Business Informatics Group, TU Wien
Email: martina.seidl@jku.at

Robert Könighofer
Inst. for Applied Information Processing and Communications
Graz University of Technology
Email: robert.koenighofer@iaik.tugraz.at

Abstract—**For effectively solving quantified Boolean formulas (QBFs), preprocessors have shown to be of great value. A preprocessor rewrites a formula such that helpful information is made explicit and irrelevant information is removed. For this purpose, techniques, which would be too costly when repeatedly applied during the solving process, are used. Unfortunately, most preprocessing techniques are not model preserving and therefore incompatible with certification frameworks. In consequence, the application of a preprocessor prohibits the extraction of witnesses encoding a solution or a counterexample of a formula.**

In this paper, we show how to obtain partial witnesses from preprocessed QBFs. Partial witnesses are assignments for the variables of the outermost quantifier block and are extensible to full witnesses, which are usually represented as functions reflecting the dependencies between variables. For many applications, however, partial witnesses are sufficient. We modified the publicly available preprocessor bloqqer for extracting partial witnesses. We empirically compare the effectiveness of the modified and the original version of bloqqer. Further, we apply the new version of bloqqer for solving hardware synthesis problems for which it turns out to be extremely beneficial.

I. Introduction

Quantified Boolean formulas (QBFs) [7], [12] provide a powerful framework for encoding and solving application problems located in PSPACE. Amongst others, QBFs are suitable for various kinds of verification and reasoning problems [3]. To this end, the original problem is translated to a QBF which is handed over to a QBF solver. The QBF solver returns a witness certifying the (un)satisfiability of the QBF which is then mapped back to the domain of the original problem.

Much progress has been made in realizing efficient solvers supported by powerful preprocessors [4], [13], [22], [11], [19] which rewrite formulas in prenex conjunctive normal form (PCNF) such that important information is made explicit and irrelevant, redundant information is removed. In the QBF Gallery 2013 [17], a non-competitive, community organized evaluation of tools for quantified Boolean formulas, it has been recently pointed out that preprocessing strongly influences the behavior of QBF solvers. Modern preprocessors like bloqqer [4], which incorporates many state-of-the-art preprocessing techniques, rewrite the formula such that it becomes easier to solve for modern QBF solvers in many cases. Today, the majority of QBF solvers is based on a generalization of the well-investigated DPLL algorithm [8] for which preprocessing seems to be of particular importance. For solvers of this kind it is easy to return an assignment of the variables quantified in the outermost quantifier block. If the outermost quantifier block is existential (resp. universal) and the formula is satisfiable (resp. unsatisfiable) then such a variable assignment represents a partial witness of the formula, because under this assignment Skolem (resp. Herbrand) functions can be calculated which are full witnesses reflecting the formula's variable dependencies. For obtaining complete witnesses from DPLL solvers, a resolution proof, which is potentially exponential in the formula size, has to be analyzed [1]. The resulting witnesses are often large and hard to handle [20]. For many applications (e.g., [2], [6]), knowledge of the assignment of the variables quantified in the outermost quantifier block is sufficient for the solution of the original application problem. However, when a preprocessor is applied, this information is lost, because preprocessors implement simplification techniques which are not model preserving in general. Therefore, it is not directly possible to reconstruct assignments which satisfy the original as well as the preprocessed QBF in the case of satisfiability, or which represent a counterexample for the original as well as for the preprocessed formula in the case of unsatisfiability.

In this paper, we show how to extract the (un)satisfying assignments for the variables occurring in the outermost quantifier block requiring only minimal modifications of the preprocessor. For this purpose, we shortly introduce QBF basics in Section II and modern preprocessing techniques in Section III. Then we argue in Section IV how to modify the application of the preprocessing techniques for getting an (un)satisfying assignment for variables in the outermost quantifier block. We realize the extraction of assignments in the preprocessor bloqqer and evaluate the impact of the extraction in Section V. Further, we evaluate how the application of our extended preprocessor influences a learning-based hardware synthesis algorithm which requires partial witnesses of QBFs. We conclude this work with a discussion of future work in Section VI.

II. Preliminaries

We consider QBFs $\Pi.\psi$ in prenex conjunctive normal form (PCNF) consisting of a propositional formula ψ called *matrix* and the *quantifier prefix* $\Pi = Q_1X_1 \dots Q_nX_n$ with $Q_i \in \{\forall, \exists\}$ and disjoint sets of variables X_i. The matrix is in conjunctive normal form, i.e., it is a conjunction of clauses. A clause is a disjunction of literals and a literal l is either a variable or a negated variable where $\mathsf{var}(l) = x$ if $l = x$ or

This work was supported in part by the Austrian Science Fund (FWF) through the national research network RiSE (S11406-N23 and S11408-N23) and the Vienna Science and Technology Fund (WWTF) under grant ICT10-018.
978-3-9815370-2-4/DATE14/©2014 EDAA

$l = \neg x$. The negation of a literal l is $\bar{l}$. We write $l \leq k$ if $\mathsf{var}(l) \in X_i$ and $\mathsf{var}(k) \in X_j$ and $i \leq j$. If convenient, we represent a clause as a set of literals and the matrix of a QBF as a set of sets. A QBF $\forall x.\phi$ (resp. $\exists x.\phi$) is true iff $\phi[x|\top]$ and (resp. or) $\phi[x|\bot]$ is true where $\phi[x|f]$ denotes the replacement of x by f and $\top$ and $\bot$ are the truth constants *true* and *false*. An assignment σ is a set of literals. A full assignment of a QBF $\Pi.\psi$ is an assignment containing all variables of Π. An assignment σ satisfies a QBF $\Pi.\psi$ iff for each clause $C \in \psi$ there exists a literal l with $l \in C$ and $l \in \sigma$. We sometimes use a satisfying assignment as a cube, i.e., as a conjunction of literals. An assignment σ falsifies a QBF $\Pi.\psi$ if there is a clause $C \in \psi$ such that there is no $l \in C$ with $l \in \sigma$. Then C is called *conflicting clause* or *conflict*. The application of an assignment σ on a QBF ϕ (denoted as $\phi\sigma$) is the QBF obtained by removing all clauses C with $l \in C$ and all literal occurrences $\bar{l}$ forall $l \in \sigma$. Given an assignment σ, then σ_l denotes $\sigma\backslash\{\bar{l}\} \cup \{l\}$. We call an assignment *partial witness* for a QBF $\phi = QX\Pi.\psi$ with $Q = \exists$ and ϕ is satisfiable or $Q = \forall$ and ϕ is unsatisfiable, if (1) $X = \{\mathsf{var}(l) | l \in \sigma\}$ and (2) ϕ and $\phi\sigma$ have the same truth value. *Full witnesses* for QBFs are given by assignment trees expressing QBF (counter)models, which basically show sets of assignments relevant for justifying the (un)satisfiability of a QBF based on the variable quantification. Details on the construction of assignment trees and the extraction of (counter)models as full witnesses can be found in [11], [22]. For this work, it is sufficient to consider a full witness as a set of assignments. For each partial witness σ, there is a full witness Ξ such that $\sigma \subseteq \tau$ forall $\tau \in \Xi$. If the outermost quantifier block is existential (resp. universal) and Ξ is a full witness for the satisfiability (resp. unsatisfiability) of the formula, then there is a partial witness σ with $\sigma \subseteq \tau$ forall $\tau \in \Xi$.

Example 1. *The assignment $\{x, y\}$ is a partial witness for the QBF $\phi = \exists x \exists y \forall a \exists z.((x \vee \neg y) \wedge (\neg x \vee y \vee a) \wedge (z \vee \neg a) \wedge (\neg z \vee a))$ and $\{\{x, y, a, z\}, \{x, y, \neg a, \neg z\}\}$ is a full witness for ϕ.*

III. QBF Preprocessing: State-of-the-Art

As the translation of the application problem in QBF often leads to formulas not in PCNF, an additional normal form transformation [21] step is necessary. However, this normal form transformation step blurs information relevant for the solver and introduces redundancies. To help the solver, special preprocessors have been proposed which reconstruct/eliminate parts of the lost/redundant information while still preserving the CNF form. We now review the techniques of the powerful, publicly available preprocessor bloqqer [4] which we will extend to get partial witnesses.

Standard Simplifications: Preprocessors implement simplification techniques also found in QBF solvers. This includes pure literal elimination, unit literal elimination, equivalence substitution, subsumption as well as universal reduction. These techniques allow the removal of variable occurrences not necessarily quantified in the outermost quantifier block. A literal is called *pure* if it occurs only in one polarity. If a pure literal is existentially (resp. universally) quantified, then all clauses containing the literal (resp. all occurrences of the literal) may be removed. An existential literal is called *unit* if it occurs in a clause of size one. If a unit literal is detected, all clauses containing the literal in the same polarity may be removed, and all occurrences of the literal in the opposite polarity may be eliminated. Obviously, if a universally quantified literal occurs in a clause of size one, the formula automatically evaluates to false. Given two clauses $(l \vee \bar{k})$ and $(\bar{l} \vee k)$ (resp. $(l \vee k)$ and $(\bar{l} \vee \bar{k})$) such that $k \leq l$ and l is existentially quantified, then *equivalence substitution* replaces l by k (resp. $\bar{k}$). A clause D is *subsumed by* a clause C if $C \subseteq D$. Then D may be safely omitted. *Universal reduction* removes a universally quantified literal l from a clause C if there is no existentially quantified literal $k \in C$ with $l < k$. We denote a clause C on which universal reduction has been applied until fixpoint by $\mathsf{univ}(C)$.

Variable Elimination (VE): Existentially quantified variables of the innermost quantifier block are removable by VE. Basically, VE is a restricted application of the algorithm of Davis and Putnam [9]. For a variable x all resolvents are calculated and added to the formula. All clauses containing x or $\neg x$ are removed. In QBF, resolution is defined as follows. Let C and D be non-tautological clauses with $x \in C$ and $\neg x \in D$. The resolvent of C and D (written as $C \otimes_x D$) with pivot element x is the clause $\mathsf{univ}(C)\backslash\{x\} \cup \mathsf{univ}(D)\backslash\{\neg x\}$. Hence, resolution for QBFs works as resolution for propositional logic extended by universal reduction.

Blocked Clause Elimination (QBCE): QBCE is also a resolution-based method. It removes clauses which will never contribute to the derivation of a conflicting clause. A blocked clause C contains an existential literal l such that all resolvents $C \otimes_{\mathsf{var}(l)} D_i$ are tautological and forall D_i there is a literal $k_i \in D_i$ with $\overline{k_i} \in C$ and $k_i \leq l$. The literal l is called *blocking literal* and the literals k_i are called *clashing literals*. If a blocked clause is found, then it can be safely removed.

Universal Expansion (UE): With universal expansion, universally quantified variables of the last universal quantifier block may be eliminated. To this end, the formula has to be copied. In one copy, the expanded variable is set to true, in the other copy the expanded variable is set to false. The two formulas are then connected by a conjunction. In order to realize universal expansion in a sound manner, it is necessary to rename the existential variables of one copy and add the new variables to the last quantifier block.

Besides the previously discussed techniques, variants thereof like *covered clause elimination*, *hidden tautology addition*, and *strengthening* are applied for preprocessing (see [4] for details). Techniques used for preprocessing not realized in bloqqer include hyper-binary resolution [22], equivalence rewriting [13], as well as failed literal probing [11]. We abstain from a detailed discussion, because conceptually, the generation of partial witnesses works similar as described below.

IV. Partial Witnesses for Preprocessed QBFs

A set of satisfying/falsifying assignments, i.e., a full witness, has to be found to determine the truth value of a QBF. This set of assignments has to fulfill the constraints imposed by the quantification. As preprocessing techniques have been shown to be sound for the original as well as for the preprocessed formula such sets justifying a formula's truth value must exist.

In this work, we aim at the reconstruction of partial witnesses from preprocessed QBFs. Basically, there are two possibilities to realize this. Either the preprocessor could be equipped with the functionality to log all preprocessing steps and especially all produced clauses from which the witness is calculable. Then the preprocessing could be replayed in reverse order and the partial witness returned from the QBF solver could be adapted and extended according to the preprocessing steps. Alternatively, a "don't-touch" approach can be realized as used in [15] for SAT-based bounded model checking, where preprocessing is applied in a restricted manner such that the values of certain variables are not affected. In the context of solving ALLQBF problems, Becker et al. [2] propose a similar modification of the preprocessor which is part of the QBF solver QuBE. We follow the latter approach as it requires less modifications of the preprocessor and hardly any postprocessing after the solving is required. To this end, we rigorously establish don't touch criteria for the individual techniques, i.e., given an assignment satisfying/falsifying the preprocessed formula, we investigate which variables' truth values have to be modified and which remain untouched to get a satisfying/falsifying assignment of the original formula. Then we empirically evaluate the impact of our modifications on the solving behavior of a state-of-the-art solver.

A. General Observations

After preprocessing, a variable can be in one of the following five states: (1) it still occurs in the formula that is handed over to the solver, (2) it is assigned either true or false by the preprocessor, (3) it has disappeared as side effect of the elimination of other variables, (4) it has been eliminated, or (5) it is found to be equivalent with another variable.

In the first case, the variable is assigned a truth value by the solver, if it is quantified in the outermost quantifier block. We modify the preprocessor such that these assignments are part of the partial witness. The second case refers to techniques like unit or pure literal elimination which give a specific value to a variable. We will argue that such a value can be kept after preprocessing. In the third case, all occurrences of a variable disappear although it is not directly touched by any preprocessing technique. Then the value of the variable can be arbitrarily chosen. If a variable (occurrence) was explicitly eliminated, then the value can not be chosen arbitrarily, but it is forced by the values of other variables. Here, we have to investigate which variables may be eliminated. Finally, if a variable is found to be equivalent with another variable, it has to take the same value as this variable.

B. Satisfiability

For satisfiability, the techniques which remove clauses are relevant. We show which variables must be modified in a given satisfying assignment σ for a QBF ϕ', such that it satisfies a QBF ϕ which is obtained from ϕ' by undoing one preprocessing step. The aim of this investigation is *not* to actually perform this a-posteriori modification of the assignment in our implementation, because this would require to trace the history of the elimination steps. We rather conclude that the modification of the assignment affects only the variables that are addressed by the respective preprocessing rule. To this end, we reuse ideas presented in [10], [14]. Järvisalo and Biere [14] discuss solution reconstruction for preprocessing techniques in propositional logic. Van Gelder [10] treats full solution reconstruction for QBFs. In both cases, the preprocessor has to trace the history of the elimination steps for reconstructing solutions. In contrast, we aim at a light weight reconstruction where this tracing is not necessary and the modifications in the preprocessor are kept minimal.

Techniques which remove clauses include blocked clause elimination, variable expansion as well as variable elimination. The following example illustrates that it would not be sound to choose an assignment arbitrarily for eliminated variables.

Example 2. *The satisfiability of QBF $\phi = \exists x \exists y \forall a \exists z.((x \vee \neg y \vee a \vee z) \wedge (\neg x \vee y \vee \neg a \vee z) \wedge \neg z)$ can be shown by unit literal elimination of z and by removing the remaining clauses by QBCE with x as blocking literal. Obviously, an arbitrary assignment like $\sigma = \{x, \neg y\}$ is not a partial witness.*

As illustrated by the example above, the preprocessing techniques impose some constraints on the values the variables may take. In the following, we review how the different techniques impose constraints in order to get satisfying assignments. Please remember that we consider individual satisfying assignments which could be assembled to a full witness. We show which variables are touched when modifying a satisfying assignment of a preprocessed formula such that it becomes a satisfying assignment of the original formula. Also keep in mind that we do not aim to construct a full certificate, but that we are only interested in finding an assignment of the outermost variables such that the formula evaluates to true.

Lemma 1 (Variable Elimination)**.** *Let $\phi = \Pi \exists X \cup \{x\}.\psi$ be a QBF where Π is an arbitrary prefix and let $\phi' = \Pi \exists X.\psi'$ be the equivalent QBF obtained by removing x from ϕ by variable elimination. If σ is a variable assignment satisfying ϕ', then there exists an l such that $\mathsf{var}(l) = x$ and the variable assignment $\sigma \cup \{l\}$ satisfies ϕ.*

Proof: Assume there is a clause C in ϕ which is not satisfied by σ. Since C obviously does not occur in ϕ', it contains a literal l with $\mathsf{var}(l) = x$. Let $\sigma' = \sigma \cup \{l\}$. Then σ' satisfies C. To show that σ' also satisfies ϕ, we have to show that there is no clause D with $\bar{l} \in D$ which is not satisfied by σ. If such a clause exists, there would be the resolvent $C \otimes_{\mathsf{var}(l)} D$ in ϕ'. However, this clause is not satisfiable by σ, which contradicts the assumption that σ satisfies ϕ'. ■

For reconstructing a satisfying assignment from VE, it is only necessary to chose the value of the eliminated variable. Hence, if we do not apply VE on the variables of the outermost quantifier block, the partial witnesses are not affected. Next, we consider QBCE.

Lemma 2 (Blocked Clause Elimination)**.** *Let C be a clause eliminated by QBCE from QBF $\Pi.\psi$ with blocking literal l and clashing literals $k_1, \ldots, k_n$. If there is an assignment σ such that $\Pi.\psi \backslash \{C\}$ is satisfiable, but such that $\Pi.\psi$ is unsatisfiable then there is an assignment σ_l satisfying $\Pi.\psi$.*

Proof: If the blocked clause C with $\{l, k_1, \ldots, k_n\} \subseteq C$ is false under σ, then all clauses D_i with $\{\bar{l}\} \in D_i$ are not only satisfied because of $\bar{l}$, but also because of some clashing literal $\overline{k_i}$. Therefore, σ_l satisfies not only C, but all clauses D_i with $\bar{l} \in D_i$. ■

According to the lemma above, the value of the blocking literal is determined by its polarity. Furthermore, the values of the clashing literals do not have to be changed. Next, we consider universal expansion. Although this technique removes universal literals, it eliminates clauses at the same time.

Lemma 3 (Universal Expansion)**.** *Assume that universal expansion creates QBF* $\phi' = \Pi\exists X\exists X'.(\psi[x|\bot] \wedge \psi'[x|\top] \wedge \chi)$ *from* $\phi = \Pi\forall x\exists X.(\psi \wedge \chi)$ *by expanding* x *where* χ *contains no variables from* $X \cup \{x\}$ *and* ψ' *is* ψ*, but each variable* $v' \in X'$ *replaces exactly one variable* $v \in X$*. If assignment* σ *satisfies* ϕ'*, then there exists an assignment* τ *satisfying* ϕ *such that for all variables* v *of* ϕ *with* $v \notin \{x\} \cup X$ *either* $v \in \sigma$ *and* $v \in \tau$ *or* $\neg v \in \sigma$ *and* $\neg v \in \tau$*.*

Proof: The clauses of χ are satisfied by σ as well as by τ. Assume that x shall be true in τ. Then $\tau = \sigma\backslash(X \cup X') \cup \{x\} \cup \{v|v' \in \sigma\} \cup \{\neg v|\neg v' \in \sigma\}$, which satisfies ψ. Otherwise, $\tau = \sigma\backslash X' \cup \{\neg x\}$ which also satisfies ψ. ■

For the calculation of assignments of the first quantifier block, the application of universal expansion is hence irrelevant, because only the values of the existential variables behind the expanded universal variable have to be modified.

Furthermore, we have to consider the standard simplification techniques. It is easy to show that the value determined by unit literal elimination or by pure literal elimination is part of a valid partial witness under the assumption that no occurrence of the eliminated variable has been a blocking literal during some application of QBCE. For subsumption and tautology detection it is not necessary to modify an assignment. Equivalence replacement naturally imposes constraints on the values the replaced variables may take.

Example 3. *Given QBF* $\exists x\exists y.((x \vee \neg y) \wedge (\neg x \vee y) \wedge (x \vee y))$*, then equivalence substitution replaces* y *by* x *resulting in* $\exists x.(x)$ *forcing the value of* x*. The only partial witness is* $\{x, y\}$*.*

Järvisalo and Biere [14] show how to integrate solution reconstruction with QBCE for propositional logic. This argument may be directly generalized for QBFs. It is straight forward to show that the different techniques may be arbitrarily interleaved. Finally, we are able to formulate the following proposition which allows us to extend a QBF preprocessor to generate partial witnesses.

Proposition 1. *Let* $\phi = \exists X\Pi.\psi$ *be a QBF and let* ϕ' *be the QBF obtained from* ϕ *after application of the previously discussed preprocessing rules with the restriction that no variable* $x \in X$ *is removed either by variable elimination or by blocked clause elimination. If* τ' *is the assignment of variables of* X *determined by pure and unit elimination during preprocessing, if* τ'' *is the assignment of the variables of* X *removed by equivalence substitution such that the equivalences are preserved, and if* σ *is a partial witness of* ϕ' *if the outermost quantifier block is existential, or the empty set otherwise, and if* τ''' *is an arbitrary assignment of the remaining variables of* X *not in* τ', τ'', σ*, then* $\tau' \cup \tau'' \cup \tau''' \cup \sigma$ *is a partial witness for* ϕ*.*

Example 4. *The QBF* $\exists x_1, x_2, x_3, x_4\forall a\exists y\exists z.\{\{x_1, a\}, \{\neg x_1, x_2\}, \{x_1, \neg x_2\}, \{x_3, z\}, \{x_4, \neg z\}, \{\neg x_3, x_4\}, \{\neg x_4, \neg x_3\}\}$ *could be preprocessed as follows.* x_1 *and* x_2 *are equivalent, so we replace* x_2 *by* x_1*. In the first clause, literal* a *is removable by universal reduction, what makes the clause unit. Hence,* x_1 *is set to true. Due to the equivalent substitution, also* x_2 *is true. Next,* z *is eliminated resulting in the matrix* $\{\{x_3, x_4\}, \{\neg x_3, x_4\}, \{\neg x_4, \neg x_3\}\}$*. One partial witness for this formula is* $\{\neg x_3, x_4\}$ *which can be found by a QBF solver. Then a partial witness for the original formula is* $\{x_1, x_2, \neg x_3, x_4\}$*.*

C. Unsatisfiability

In case a formula is unsatisfiable, a variable assignment is of interest if the outermost quantifier block is universal. Then such an assignment is the basis for a witness falsifying the formula no matter how innermore variables are assigned. As before, we review how the different techniques contribute to the evaluation of the formula. Due to space limitations, we give only an informal discussion in the following. When considering formulas in CNF, then a formula is falsified by an assignment if there is a clause which is unsatisfiable under this assignment. In propositional logic, not much attention is given to this situation, because if a formula is unsatisfiable there is no assignment. To parallelize SAT solvers [18], it has been investigated how conflicts can be learned when in-processing is enabled. Inprocessing refers to the controlled application as preprocessing techniques during solving. To the best of our knowledge, no work related to QBF solving has been presented.

Tautology removal and subsumption as well as existential pure literal elimination reduce the number of clauses but do not change the size of a clause. Therefore, these techniques never contribute in producing a conflict. Unit literal elimination also reduces the size of clauses, but it only sets the values of existential variables. Also equivalent reasoning induces only the value of existentially quantified variables. Therefore, none of these techniques is relevant for getting the values of the universal variables in the outermost quantifier block. If a clause produced by variable elimination is conflicting, then obviously one of the antecedent clauses is conflicting. As the pivot element is existentially quantified, its value is not of interest for our purposes.

The only remaining techniques are universal reduction, universal expansion, and universal pure literal elimination. If a variable of the outermost quantifier block is eliminated by universal reduction, this immediately results in a conflict, because this means that the clause does not contain any existential variables. Then the variable is set to true if it is negated in the clause, otherwise it is set to false. For universal expansion, we have argued in the previous section that in the case of satisfiability only the truth values of the universal variable as well as the following existential variables have to be adopted in order to get a satisfying assignment. The same line of argumentation may be applied in the case of unsatisfiability. Also here, only the values of the expanded variable as well as the values of the existential variables following the expanded variable in the prefix have to be modified. That such a modification exists is implied by the soundness of universal expansion. In consequence, it can be easily shown that if we restrict the application of universal expansion on variables not quantified in the outermost quantifier block, already established variable values can be kept. Due to these restrictions, universal pure literal elimination sets truth values for the partial witnesses.

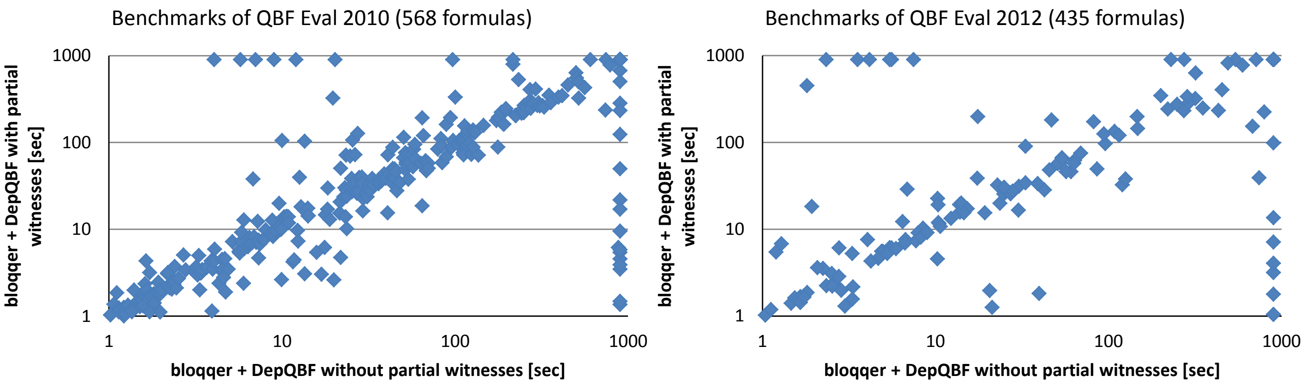

Fig. 1. Comparison of bloqqer Versions

V. Experimental Evaluation

We extended the preprocessor bloqqer[1] with an option to provide values for the variables of the outermost quantifier block. For getting the partial witness of the preprocessed formula, we integrated the QBF solver DepQBF [16] which provides an API function for getting partial witnesses and which has very good synergies with bloqqer [17], [4]. Our experiments first evaluate the impact of our modifications on the solving performance. Then we present experimental results of using preprocessing in synthesis. Without our extension bloqqer would not be applicable as this approach requires partial witnesses of the QBFs.

A. Benchmarks from the QBF Evaluations

In our first experiment, we test the impact of the modification of bloqqer on the preprocessing and solving performance. We evaluated the benchmark sets used in the QBF Eval 2010 and QBF Eval 2012 [17] with and without the generation of partial witnesses. The experiments were run on a cluster of equal machines with Intel Core 2 Quad CPUs having 2.83 GHz and 8 GB main memory. Time limits were set to 900 seconds for solving and preprocessing time and memory was restricted to 7 GB. The outcome of our experiments is shown in the scatter plots of Fig. 1. In general, the solving behavior is not much affected by our modification as we find most runtimes close to the diagonal for both benchmark sets. In the case of the benchmark set from 2010, DepQBF solves 6 more formulas with the partial witness producing version of bloqqer. This is because our modifications affect the internal behavior of bloqqer such that different techniques might be applied. This can be beneficial for certain kinds of formulas. It is the biu family which is very strongly present in the benchmark set and which benefits from our modifications. In contrast, only 3 less formulas are solved in the benchmark set of 2012. Although bloqqer is applied in a more restrictive way, there is hardly any negative impact.

B. Benchmarks from Game-Based Synthesis

The importance of preprocessing becomes evident in the practical application of QBF. As case study, we consider synthesis of reactive hardware systems from safety specifications, i.e., the problem of synthesizing a system that will never visit an unsafe state. Synthesis problems can be seen as a game between two players: the system trying to satisfy the specification, and the environment trying to violate it. This competitive aspect can be handled by using different quantifiers for the two players, which gives a natural reduction to QBF. We use a learning-based approach as presented in [6]. It computes a *winning region*, i.e., the states from which the specification is fullfillable. For the details we kindly refer to [6]. Initially, the winning region F is set to the safe states P. In every iteration, we first compute a state from which the environment can enforce to leave the winning region (and, hence, reach an unsafe state eventually). Such a state is computed as satisfying assignment for the QBF $\exists X \exists J \forall C \exists X'.(F \wedge T \wedge \neg F')$. In this formula, the variables X (resp. X') model the current (resp. the successor) state, J are the variables controlled by the environment, C are the variables controlled by the system, and T denotes the transition relation. If such a state exists, it is removed from the winning region F (after generalizing it into a larger region of states that can be removed). This is repeated until the QBF becomes unsatisfiable (which means that there are no more states from which the environment can visit an unsafe state), or an initial state is removed (which means that the problem is unrealizable).

The learning-based synthesis algorithm strongly relies on a QBF solver for calculating problematic moves of the environment. To this end, the QBF solver has to provide partial witnesses for true QBFs. We also experimented with a second synthesis algorithm, which is based on templates [6]. Here, the search for a winning region is reduced to one single QBF-solver call. A satisfying assignment for the variables quantified existentially on the outermost level gives the solution. As in the previous section, we use the QBF solver DepQBF. The experiments were run on an Intel Xeon E5430 CPU with 4 cores (only one was used) running at 2.66 GHz, and a 64 bit Linux. We set a timeout of 10 000 seconds. We evaluated the approach on 207 specifications including specifications of an arbiter for ARMs AMBA AHB bus [5], parameterized on the number of masters it can handle, as well as specifications of a generalized buffer [5] connecting different numbers of senders to two receivers, as well as i-bit adders, i-bit multipliers, i-bit counters, and i-bit barrel shifters.

Figure 2 compares the runtimes for the two synthesis algorithms, with and without preprocessing. The blue crosses show the results of the learning-based synthesis approach, the red crosses show the results from the template-based approach.

[1] http://fmv.jku.at/bloqqer

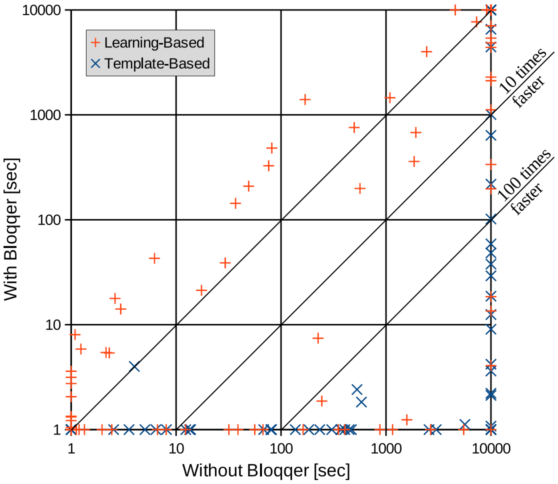

Fig. 2. Synthesis with QBF

For the learning-based synthesis approach, the number of solved instances increased from 98 to 110. The average synthesis time (counting only cases were both versions terminate) decreased from 308 to 187 seconds. For the template-based synthesis approach, the results are even more impressive. The number of solved instances increased from 73 to 113, while the average synthesis time decreased from 214 seconds to only 1 second.

VI. Conclusion and Future Work

In this paper, we showed how to equip a modern QBF preprocessor with the capability of providing variable assignments for the outermost quantifier block. We used this approach in the context of game-based synthesis where the outermost, existentially quantified variables help to find a winning strategy to synthesize safe programs. Furthermore, we compared the modified version of bloqqer with the publicly available version in order to find out if the calculation is imposing restrictions on the preprocessing and solving performance. It turned out that the necessary modifications only have moderate effects.

In future work, we plan to extend our approach to get full certificates for the QBF formulas in form of Skolem and Herbrand functions. For this purpose stronger modifications of the preprocessor will be required in terms of tracking the history of performed preprocessing steps. For the calculation of full certificates a complete Q-resolution proof has to be reconstructed whose leaves contain only formulas of the input formula. In the case of an unsatisfiable QBF this is directly possible as the clause elimination techniques are basically resolution based. The challenging case are the satisfiable formulas for which a cube resolution proof has to be provided. Here, first ideas have been presented in [10] but no implementation has been provided so far.

Overall, the possibility to extract solutions from a preprocessor provides a tighter integration of the different components of the QBF tool chain which is extremely valuable for solving application problems with QBF.

References

[1] V. Balabanov and J. R. Jiang. Unified QBF certification and its applications. *Formal Meth. in Syst. Design*, 41(1):45–65, 2012.

[2] B. Becker, R. Ehlers, M. Lewis, and P. Marin. ALLQBF Solving by Computational Learning. In *Proc. of the Int. Symp. of Aut. Tech. for Verification and Analysis*, pages 370–384. Springer, 2012.

[3] M. Benedetti and H. Mangassarian. QBF-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.

[4] A. Biere, F. Lonsing, and M. Seidl. Blocked Clause Elimination for QBF. In *Proc. 23rd Int. Conf. on Automated Deduction (CADE)*, volume 6803 of *LNCS*, pages 101–115. Springer, 2011.

[5] R. Bloem, S. J. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Specify, Compile, Run: Hardware from PSL. *Electr. Notes Theor. Comput. Sci.*, 190(4):3–16, 2007.

[6] R. Bloem, R. Könighofer, and M. Seidl. Sat-based synthesis methods for safety specs. *CoRR*, abs/1311.3530, 2013.

[7] H. Kleine Büning and U. Bubeck. Theory of Quantified Boolean Formulas. In *HB of Sat.*, pages 735–760. IOS Press, 2009.

[8] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 4:394–397, 1962.

[9] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[10] A. Van Gelder. Certificate Extraction from Variable-Elimination QBF Preprocessors. In *Proc. of the 1st Int. Workshop on Quantified Boolean Formulas (QBF 2013)*, pages 35–39. http://fmv.jku.at/qbf2013/reportQBFWS13.pdf, 2013.

[11] A. Van Gelder, S. B. Wood, and F. Lonsing. Extended Failed-Literal Preprocessing for Quantified Boolean Formulas. In *Proc. of the 15th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2012)*, volume 7317 of *LNCS*, pages 86–99. Springer, 2012.

[12] E. Giunchiglia, P. Marin, and M. Narizzano. Reasoning with Quantified Boolean Formulas. In *HB of Sat.*, pages 761–780. IOS Press, 2009.

[13] E. Giunchiglia, P. Marin, and M. Narizzano. sQueezeBF: An Effective Preprocessor for QBFs Based on Equivalence Reasoning. In *Proc. of the 13th Int. Conf. on Theory and Applications of Sat. Testing (SAT 2010)*, pages 85–98. Springer, 2010.

[14] M. Järvisalo and A. Biere. Reconstructing Solutions after Blocked Clause Elimination. In *Proc. of th 13th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2010)*, volume 6175 of *Lecture Notes in Computer Science*, pages 340–345. Springer, 2010.

[15] S. Kupferschmid, M. Lewis, T. Schubert, and B. Becker. Incremental preprocessing methods for use in BMC. *Formal Meth. in Syst. Design*, 39(2):185–204, 2011.

[16] F. Lonsing and A. Biere. DepQBF: A dependency-aware QBF solver. *JSAT*, 7(2-3):71–76, 2010.

[17] F. Lonsing, M. Seidl, and A. Van Gelder. The QBF Gallery 2013. URL: http://kr.tuwien.ac.at/qbfgallery13, 2013.

[18] N. Manthey, T. Philipp, and C. Wernhard. Soundness of Inprocessing in Clause Sharing SAT Solvers. In *Proc. of the 16th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2013)*, volume 7962 of *LNCS*, pages 22–39. Springer, 2013.

[19] P. Marin, C. Miller, and B. Becker. Incremental QBF Preprocessing for Partial Design Verification . In *Proc. of the 15th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2012)*, volume 7317 of *LNCS*, pages 473–474. Springer, 2012.

[20] A. Niemetz, M. Preiner, F. Lonsing, M. Seidl, and A. Biere. Resolution-Based Certificate Extraction for QBF. In *Proc. of the 15th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2012)*, volume 7317 of *LNCS*, pages 430–435. Springer, 2012.

[21] D. A. Plaisted and S. Greenbaum. A Structure-Preserving Clause Form Translation. *Journal of Symb. Computation*, 2(3):293–304, 1986.

[22] H. Samulowitz, J. Davies, and F. Bacchus. Preprocessing QBF. In *Proc. of the 12th Int. Conf. on Principles and Practice of Constraint Programming (CP 2006)*, pages 514–529. Springer, 2006.