Non-prenex QBF Solving using Abstraction*

Leander Tentrup

Reactive Systems Group
Saarland University
tentrup@react.uni-saarland.de

Abstract. In a recent work, we introduced an abstraction based algorithm for solving quantified Boolean formulas (QBF) in prenex negation normal form (PNNF) where quantifiers are only allowed in the formula's prefix and negation appears only in front of variables. In this paper, we present a modified algorithm that lifts the restriction on prenex quantifiers. Instead of a linear quantifier prefix, the algorithm handles tree-shaped quantifier hierarchies where different branches can be solved independently. In our implementation, we exploit this property by solving independent branches in parallel. We report on an evaluation of our implementation on a recent case study regarding the synthesis of finite-state controllers from ω-regular specifications.

1 Introduction

In recent work [18], we introduced an algorithm for solving quantified Boolean formulas (QBF) in prenex negation normal form (PNNF). For each maximal consecutive block of quantifiers of the same type, we build an *abstraction*, i.e., a propositional formula that combines valuations of inner and outer quantifier blocks into valuations of special literals, called *interface literals*. The algorithm employs a counterexample guided abstraction refinement (CEGAR) loop that does recursion over the quantifier blocks. In every block, we use a SAT solver as an oracle to generate new abstraction entries and to provide us with witnesses for unsatisfiable queries. This algorithm, however, is limited to prenex QBF where quantifier are only allowed in the formula's prefix. This can be problematic for non-prenex formulas since the task of prenexing a QBF is non-deterministic and different prenexing strategies lead to different solving times [3]. On the other hand, miniscoping can be used to translate prenex formulas into non-prenex form. We have observed [17] that this is very effective for splitting instances into independent parts on some benchmark families.

In this paper, we extend our previous algorithm to handle non-prenex QBFs in negation normal form. Instead of a linear quantifier prefix, the algorithm is optimized to handle tree-shaped quantifier hierarchies. These optimizations include identifying parts of the formula that belongs only to one quantifier block,

* This work was partially supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

hence, eliminating the need for interface literals. Further, for a branching node, i.e., a quantifier block which has multiple children, it is possible to solve the children independently. Our implementation exploits this independence by solving the different branches in parallel.

2 Quantified Boolean Formulas

A quantified Boolean formula (QBF) is a propositional formula over a finite set of variables $\mathcal{X}$ with domain $\mathbb{B} = \{0,1\}$ extended with quantification. The syntax is given by the grammar

$$\varphi \coloneqq x \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists x.\, \varphi \mid \forall x.\, \varphi \ ,$$

where $x \in \mathcal{X}$. For readability, we lift the quantification over variables to the quantification over sets of variables and denote a maximal consecutive block of quantifiers of the same type $\forall x_1.\, \forall x_2.\cdots\forall x_n.\, \varphi$ by $\forall X.\, \varphi$ and $\exists x_1.\, \exists x_2.\cdots\exists x_n.\, \varphi$ by $\exists X.\, \varphi$, accordingly, where $X = \{x_1, \ldots, x_n\}$.

Given a subset of variables $X \subseteq \mathcal{X}$, an *assignment* of X is a function $\alpha : X \to \mathbb{B}$ that maps each variable $x \in X$ to either true (1) or false (0). We identify α as the conjunctive formula $\bigwedge_{x \in X \mid \alpha(x)=1} x \wedge \bigwedge_{x \in X \mid \alpha(x)=0} \neg x$. When the domain of α is not clear from context, we write α_X. A partial assignment $\beta : X \to \mathbb{B} \cup \{\bot\}$ may additional set variables $x \in X$ to an undefined value $\bot$. We say that β is *compatible* with α, $\beta \sqsubseteq \alpha$ for short, if they have the same domains ($\mathrm{dom}(\alpha) = \mathrm{dom}(\beta)$) and $\alpha(x) = \beta(x)$ for all $x \in X$ where $\beta(x) \neq \bot$. For two assignments α and α' with disjoint domains $X = \mathrm{dom}(\alpha)$ and $X' = \mathrm{dom}(\alpha')$ we define the combination $\alpha \sqcup \alpha' : X \cup X' \to \mathbb{B}$ as $\alpha \sqcup \alpha'(x) = \begin{cases} \alpha(x) & \text{if } x \in X, \\ \alpha'(x) & \text{otherwise.} \end{cases}$ We define the *complement* $\overline{\alpha}$ to be $\overline{\alpha}(x) = \neg\alpha(x)$ for all $x \in \mathrm{dom}(\alpha)$. The *set of assignments* of X is denoted by $\mathcal{A}(X)$.

A quantifier $Q\, x.\, \varphi$ for $Q \in \{\exists, \forall\}$ *binds* the variable x in the *scope* φ. Variables that are not bound by a quantifier are called *free*. The set of free variables of formula φ is defined as $\mathit{free}(\varphi)$. We assume the natural semantics of the satisfaction relation $\alpha_X \vDash \varphi$ for QBF φ and assignments α_X where X are the free variables of φ. *QBF satisfiability* is the problem to determine, for a given QBF φ, the existence of an assignment α for the free variables of φ, such that the relation $\vDash$ holds.

A *closed* QBF is a formula without free variables. Closed QBFs are either true or false. A formula is in prenex form, if the formula consists of a quantifier prefix followed by a propositional formula.

A *literal* l is either a variable $x \in X$, or its negation $\neg x$. Given a set of literals $\{l_1, \ldots, l_n\}$, the disjunctive combination $(l_1 \vee \ldots \vee l_n)$ is called a *clause*. Given a literal l, the polarity of l, $\mathit{sign}(l)$ for short, is 1 if l is positive and 0 otherwise. The variable corresponding to l is defined as $\mathit{var}(l) = x$ where $x = l$ if $\mathit{sign}(l) = 1$ and $x = \neg l$ otherwise.

A QBF is in negation normal form (NNF) if negation is only applied to variables. Every QBF can be transformed into NNF by at most doubling the

size of the formula and without introducing new variables. For formulas in NNF, we treat literals as atoms.

3 Algorithm

We introduce additional notation to facilitate working with arbitrary Boolean formulas. Let $\mathcal{B}$ be the set of quantified Boolean formulas and let $\mathit{sf}(\varphi) \subset \mathcal{B}$ ($\mathit{dsf}(\varphi) \subset \mathcal{B}$) be the set of (direct) subformulas of φ (note that $\varphi \in \mathit{sf}(\varphi)$ but $\varphi \notin \mathit{dsf}(\varphi)$). Further, $\mathit{dqsf}(\varphi) \subset \mathcal{B}$ denotes the direct quantified subformulas of φ, i.e., a quantifier $Q\,X'.\,\psi$ is in $\mathit{dqsf}(\varphi)$ if $Q\,X'.\,\psi$ is in the scope of φ and there is no other quantifier $Q\,X''.\,\psi'$ such that $Q\,X''.\,\psi'$ is in the scope of φ and $Q\,X'.\,\psi$ is in the scope of ψ'. For a subformula ψ, $\mathit{type}(\psi) \in \{\mathit{lit}, \vee, \wedge, Q\}$ returns the Boolean connector if ψ is not a literal nor a quantifier. For example, given $\psi = \exists x.\,(\forall y.\,\exists z.\,(x \vee y \vee \neg x)) \vee (\forall y.\,(y \wedge x))$, it holds that $\mathit{type}(\psi) = Q$, $\mathit{dsf}(\psi) = \{\forall y.\,\exists z.\,(x \vee y \vee \neg x)) \vee (\forall y.\,(y \wedge x)\}$, and $\mathit{dqsf}(\psi) = \{\forall y.\,\exists z.\,(x \vee y \vee \neg x), \forall y.\,(y \wedge x)\}$.

For this section, we assume w.l.o.g. that all quantifier blocks in the QBF are strictly alternating, even for quantifiers not in the prefix. That means that for every quantified formula $Q\,X.\,\psi$, the quantifier type of all $\psi' \in \mathit{dqsf}(Q\,X.\,\psi)$ is $\overline{Q}$. We use a generic solving function $\textsc{sat}(\theta, \alpha)$ for propositional formula θ under assumptions α, that returns whether $\theta \wedge \alpha$ is satisfiable and either a satisfying assignment α_V for variables $V \subseteq \mathit{free}(\theta)$ or a partial assignment $\beta_{\mathit{failed}} \sqsubseteq \alpha$ such that $\theta \wedge \beta_{\mathit{failed}}$ is unsatisfiable. Further, we define SAT_Q to be SAT if $Q = \exists$ and UNSAT otherwise (UNSAT_Q analogously).

The non-prenex algorithm works on the principle of communicating the satisfaction of subformulas between quantifier blocks in the QBF. This communication is realized by two special types of literals which we call *interface literals*. Only the valuation of those literals are communicated between the quantifier levels. For a given quantifier $Q\,X$ and a subformula ψ, the T literal t_ψ represents the assignments made by the outer quantifiers while the B literal b_ψ represents the assignments from the current quantifier $Q\,X$ including assumptions on the satisfaction of subformulas by inner quantifiers. Thus, t_ψ is true if ψ is satisfied by the outer quantifiers and a valuation that sets b_ψ to true indicates that ψ is satisfied by the quantifier $Q\,X$. Before going in more detail on the abstraction, we introduce the basic algorithm first.

The algorithm $\textsc{abstraction-qbf}$ is depicted in Algorithm 1. The algorithm uses a *dual abstraction* for optimization of abstraction entries [18]. Given a quantifier $Q\,X$, the sets T_X and B_X contain the T and B literals corresponding to this quantifier. When translating a B literal to a T literal, we use the the same index, e.g., in line 12, the B literals $b_{\psi''}$ are translated to T literals $t_{\psi''}$ of the inner quantifier. We initialize θ_X with the abstraction described below, which is a propositional formula over variables in X, as well as T and B literals. For the innermost quantifier, it holds that $B_X = \emptyset$. Further, the *dual abstraction* $\overline{\theta}_X$ is defined as the abstraction for $\overline{Q}\,X$.

Algorithm 1 Non-prenex Abstraction Based Algorithm

```
 1: procedure ABSTRACTION-QBF(Q X. ψ, α_{T_X})
 2:     while true do
 3:         result, α_X ⊔ α_{B_X}, β_failed ← SAT(θ_X, α_{T_X})          ▷ β_failed ⊑ α_{T_X}
 4:         if result = UNSAT then
 5:             return UNSAT_Q, β_failed
 6:         else if ψ is propositional then
 7:             result, _, β_failed ← SAT(θ̄_X, α_X ⊔ \overline{α_{T_X}})   ▷ result = UNSAT
 8:             return SAT_Q, \overline{β_failed}                          ▷ \overline{β_failed} ⊑ α_{T_X}
 9:         sub-result ← SAT_Q
10:         Let β_sub be the empty assignment
11:         for ψ' = Q̄ Y. ψ* in dqsf(Q X. ψ) where α_{B_X}(b_{ψ'}) = 0 do
12:             Define α_{T_Y} s.t. α_{T_Y}(t_{ψ''}) = ¬α_{B_X}(b_{ψ''}) for all t_{ψ''} ∈ T_Y
13:             result, β_{T_Y} ← ABSTRACTION-QBF(ψ', α_{T_Y})               ▷ β_{T_Y} ⊑ α_{T_Y}
14:             if result = UNSAT_Q then
15:                 θ_X ← θ_X ∧ (⋁_{b_{ψ''} ∈ B_X | β_{T_Y}(t_{ψ''})=1} b_{ψ''})
16:                 sub-result ← UNSAT_Q
17:             else
18:                 β_sub ← β_sub ⊔ β_{T_Y}
19:         if sub-result = SAT_Q then
20:             θ̄_X ← θ̄_X ∧ (⋁_{b_{ψ''} ∈ B_X | β_sub(t_{ψ''})=1} b_{ψ''})
21:             result, _, β_failed ← SAT(θ̄_X, α_X ⊔ \overline{β_sub})     ▷ result = UNSAT
22:             return SAT_Q, \overline{β_failed}                          ▷ \overline{β_failed} ⊑ α_{T_X}
```

In every iteration of the while loop, a B literal assignment α_{B_X} is generated according to the outer T assignment α_{T_X} (line 3). For every direct quantified subformula $\psi' = \overline{Q}\, Y.\, \psi^*$ (line 11) which is assumed to be satisifed ($\alpha_{B_X}(b_{\psi'}) = 0$), we translate α_{B_X} into a T assignment for the inner quantifier (line 12) and proceed recursively (line 13). If the recursive call is UNSAT (w.r.t the current quantifier), we refine the abstraction to exclude the counterexample β_{T_Y}, i.e., in the following iterations one of the $b_{\psi''}$ such that $\beta_{T_Y}(t_{\psi''}) = 1$ must be set to true. Due to the negation during translation, those $b_{\psi''}$ were set to false in α_{B_X}. In case the recursive call is SAT, we update β_{sub} to include the optimized T assignment returned from the inner quantifier. When all recursive calls returned SAT, we update the dual abstraction $\overline{\theta}_X$ and use it to optimize the witness β_{sub} for the outer scope. If the query in line 3 fails, UNSAT_Q together with an assignment $\beta_{failed} \sqsubseteq \alpha_{T_X}$ witnessing the unsatisfiability is returned.

To ensure correctness, we need requirements on the abstraction being used. Given a quantifier $\exists X$, we say that a subformula ψ is good if it is not yet falsified ($type(\psi) = \wedge$), respectively satisfied ($type(\psi) = \vee$). For every quantifier $Q\, X$ and B literal b_ψ it must hold that if b_ψ is set to true then ψ is good for quantifier X. This gives us proper refinement semantics (line 15). The same property holds for t literals t_ψ. For every direct quantified subformula $\psi' = Q\, Y.\, \varphi^*$ in the current scope, the B literal $b_{\psi'}$ can be only set to true if the subformula ψ' is not assumed to be true. Intuitively, this means that the result of subformula ψ' is not used

to satisfy the abstraction θ_X. Further, for a quantifier alternation, it must hold that the set of outer B literals B_X matches the union of all inner T literals T_Y to enable the translation in line 12. Combining these properties gives us that a good subformula of quantifier $Q\,X$ is a bad subformula of quantifier $Q\,Y$ and vice versa. Termination then follows from progress due to refinements (line 15) and correctness can be showed by induction over the quantifiers.

Abstraction. We now give a formal definition of the abstraction. Given a QBF φ in NNF and a quantifier $\exists X.\,\varphi'$, we build the following propositional formula in conjunctive normal form representing the structural abstraction $\theta_X = \mathit{out}(\varphi) \wedge \bigwedge_{\psi \in sf(\varphi) \wedge type(\psi) \neq lit} \mathit{enc}(\psi)$ for this quantifier, where *out* encodes the entry point of the formula and *enc* defines a CNF formula that encodes the truth of subformula ψ with respect to the valuations of the current, inner and outer quantifiers represented by B and T literals, respectively:

$$\mathit{enc}(\psi) = \begin{cases} \bigwedge_{\psi' \in dsf(\psi)} (\neg b_\psi \vee \mathit{enc}_\psi(\psi')) & \text{if } \mathit{type}(\psi) = \wedge \\ \neg b_\psi \vee \bigvee_{\psi' \in dsf(\psi)} \mathit{enc}_\psi(\psi') & \text{if } \mathit{type}(\psi) = \vee \\ (b_\psi \vee \mathit{out}(\psi')) & \text{if } \psi = Q\,X.\,\psi' \end{cases}$$

$$\mathit{enc}_\psi(\psi') = \begin{cases} \psi' & \text{if } \mathit{type}(\psi') = \mathit{lit} \wedge \mathit{var}(\psi') \in X \\ t_\psi & \text{if } \mathit{type}(\psi') = \mathit{lit} \wedge \mathit{var}(\psi') \text{ bound by outer scope} \\ \neg b_\psi & \text{if } \mathit{type}(\psi') = \mathit{lit} \wedge \mathit{var}(\psi') \text{ bound by inner scope} \\ b_{\psi'} & \text{if } \mathit{type}(\psi') \neq \mathit{lit} \wedge \psi' \text{ only influenced by current or outer scope} \\ \neg b_{\psi'} & \text{if } \mathit{type}(\psi') = Q \\ \bot & \text{otherwise} \end{cases}$$

$$\mathit{enc}_\vee(\psi) = \bigvee_{\substack{\psi' \in dsf(\psi) \\ type(\psi') = lit}} \mathit{enc}_\psi(\psi') \vee \bigvee_{\substack{\psi' \in dsf(\psi) \\ type(\psi') = \wedge}} b_{\psi'} \vee \bigvee_{\substack{\psi' \in dsf(\psi) \\ type(\psi') = \vee}} \mathit{enc}_\vee(\psi') \vee \bigvee_{\substack{\psi' \in dsf(\psi) \\ type(\psi') = Q}} \neg b_{\psi'}$$

$$\mathit{out}(\psi) = \begin{cases} b_\psi & \text{if } \mathit{type}(\psi) = \wedge \\ \mathit{enc}_\vee(\psi) & \text{if } \mathit{type}(\psi) = \vee \\ \neg b_\psi & \text{if } \mathit{type}(\psi) = Q \end{cases}$$

An undefined result $\bot$ form $\mathit{enc}_\psi(\psi')$ means that the subformula ψ' is ignored in the encoding $\mathit{enc}(\psi)$. The abstraction of a scope $\forall X$ is defined as the existential abstraction for $\neg\varphi$. The dual abstraction $\overline{\theta}_X$ is defined as the abstraction for $\neg\,Q\,X$. Note that not every B literal that is used in the abstraction may be exposed as an interface literal.

For disjunctive formulas, $\mathit{enc}(\psi)$ enforces that b_ψ can be only set to true if a direct subformula that is (1) a (possibly negated) variable of the current or outer scope, (2) a subformula that is not influenced by an inner variable, or (3) a quantified subformula, is set to true. The encoding $\mathit{enc}(\psi)$ of a conjunctive formula enforces likewise that if b_ψ is true, the encodings $\mathit{enc}_\psi(\psi')$ of all such

direct subformulas $\psi' \in dsf(\psi)$ are true. The abstraction has the required refinement semantics: If we want to ensure that one of the subformulas in a set $R = \{\psi_1, \dots, \psi_k\}$ is guaranteed to be true at the current scope, we add the clause $(b_{\psi_1} \vee \cdots \vee b_{\psi_k})$.

Optimizations. The optimizations from the prenex algorithm [18] can be applied to this algorithm as well. Additionally, we preprocess the formula using the well-known miniscoping rules in order to decompose quantifier blocks.

In the algorithm, satisfying results of direct quantified subformulas are discarded if one of them is UNSAT. Instead, we found that modifying the decision heuristic of the underlying SAT solver to regenerate this subassignment reduced the number of iterations overall.

4 Case Study: Reactive Synthesis

For our case study, we consider the *reactive synthesis* problem, i.e., the problem of synthesizing a finite-state controller from an ω-regular specification. Formally, we have a specification φ that defines a language $\mathcal{L}(\varphi) \subseteq (2^{I \cup O})^\omega$ over the atomic propositions that are partitioned into a finite set of inputs I to the controller and a finite set of outputs O of the controller. An implementation of a controller is a 2^O*-labeled* 2^I*-transition system* $\mathcal{S} = \langle S, s_0, \delta, l \rangle$ where S is a finite set of states, $s_0 \in S$ is the designated initial state, $\delta \colon S \times 2^I \to S$ is the transition function, and $l \colon S \to 2^O$ is the state-labeling. The run of $\mathcal{S}$ on a sequence $\pi \in (2^I)^\omega$ is $run(\mathcal{S}, \pi) = s_0 \pi_0 s_1 \pi_1 \cdots \in (S \cdot 2^I)^\omega$ where $s_{i+1} = \delta(s_i, \pi_i)$ for every $i \geq 0$. The corresponding trace, denoted by $trace(\mathcal{S}, \pi)$, is $(l(s_0) \cup \pi_0)(l(s_1) \cup \pi_1) \cdots \in (2^{I \cup O})^\omega$. A transition system $\mathcal{S}$ satisfies the specification φ if $trace(\mathcal{S}, \pi) \in \mathcal{L}(\varphi)$ for all input sequences $\pi \in (2^I)^\omega$. By bounding the number of states that the implementation of the controller may use, one can derive a QBF encoding [4] from this problem using the *bounded synthesis* approach [5]. The synthesis instances used in this case study where taken from the Acacia benchmark set [2].

The exact encoding is out of scope for this paper, so we are only giving a high level overview. The QBF query has a quantifier prefix of the form $\exists\forall\exists$. The variables in the top level existential correspond to a global constraint that cannot be split syntactically. However, the constraints regarding the inner quantifiers $\forall\exists$ are local to the state of the implementation, so one gets a QBF with a top level existentially quantifier and n independent $\forall\exists$ quantifiers below by using miniscoping rules, where n is the number of states in the implementation. This is merely a new observation and not particularly special for this kind of benchmark as we have made similar observations regarding competitive benchmark suites for CNF [17].

We implemented Algorithm 1 and its optimizations in a prototype tool called PQuAbS (Parallel Quantified Abstraction Solver)[1] that takes QBFs in the standard format QCIR [16]. We use PicoSAT [1] as the underlying SAT solver and

[1] Available at https://www.react.uni-saarland.de/tools/quabs/

Table 1. Cumulated solving time of PQuAbS with respect to number of used threads. There are 443 instances in total.

	1 thread	2 threads	3 threads	4 threads	prenex
# solved instances	397	403	407	409	325
cumulated solving time	100%	64.51%	54.15%	49.94%	-

the POSIX pthreads library for thread creation and synchronization. For every quantifier $Q\,X$ that branches more than once, we create a thread for each child quantifier. The loop in line 11 is then implemented by passing α_{T_Y} to the subquantifier and waking the corresponding thread. Before line 19, there is a barrier where we wait for all children to finish. For our experiments, we used a machine with a 3.6 GHz quad-core Intel Xeon processor and 32 GB of memory. The timeout was set to 10 minutes.

Table 1 shows the overall results of our experiments. It depicts the number of solved instances and the cumulated solving times with respect to the number of threads used. For comparison, we also included the number of solved instances from the single threaded version of PQuAbS without miniscoping, i.e., linear prenex solving. One cannot expect linear speedup due to the non-parallelizable parts, like preprocessing and solving of the top-level existential quantifier, as well as the fact that the solving time of the children $\forall\exists$ quantifiers are not uniform.

Nevertheless, already using 2 threads, the speedup compared to single thread solving is more than 1.5 and using 4 threads reduces the solving time by a factor of 2 on average. Table 2 gives detailed results for select instances from the scatter plot of Figure 1. These examples are the two "outliers" *load-full-6* and *ltl2dba-05*, the hardest commonly solved instance *ltl2dba-23*, and two instances with close to optimal speedup (*ltl2dpa-12* and *ltl2dpa-11*).

Table 2. Detailed solving results for example instances.

instance	branching	1 thread	2 threads	3 threads	4 threads
ltl2dba-23	10	598.20 s 100%	393.68 s 65.81%	335.59 s 56.10%	312.70 s 52.27%
ltl2dpa-12	15	521.35 s 100%	302.13 s 57.95%	233.98 s 44.88%	202.27 s 38.80%
ltl2dba-05	4	476.12 s 100%	359.40 s 75.49%	331.87 s 69.70%	322.59 s 67.75%
load-full-6	3	386.94 s 100%	332.15 s 85.84%	314.37 s 81.25%	321.75 s 83.15%
ltl2dpa-11	18	252.61 s 100%	143.13 s 56.67%	107.54 s 42.57%	92.43 s 36.59%

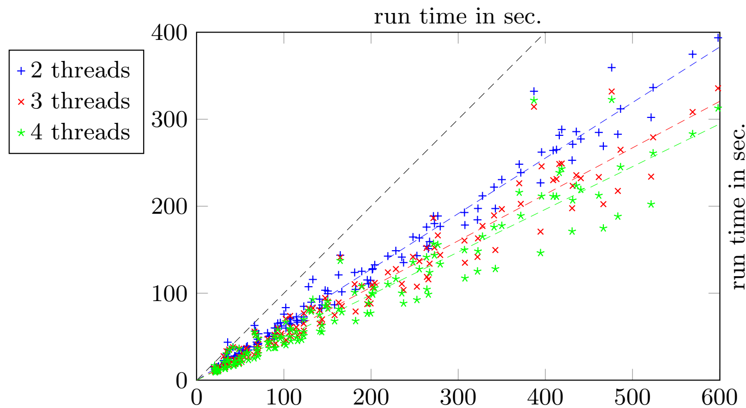

Fig. 1. Scatter plot of solving times with multiple threads against single thread baseline. Here, we consider only instances with more than 1 second of solving time.

5 Related Work

There are other solving techniques that use structural information, but they are conceptional very different, including DPLL like [3, 6, 10] and expansion [7, 13, 15]. Further, some of them can be applied to non-prenex setting as well [3, 10]. Employing SAT solver to solve propositional queries with quantifier alternations has been used before [7, 8, 17, 19]. We extend our own work on abstraction based QBF solving [18] that itself originated from techniques that communicate the satisfaction of clauses through a recursive refinement algorithm [8, 17] that were limited to conjunctive normal form. MPIDepQBF [9] is the most recent parallel solver for QBF. Their approach differs from our as they start instances of a sequential solver without synchronization. Da Mota et al. [14] proposed methods to split a QBF at the top level and solve the resulting QBF instances in parallel by a sequential CNF algorithm. In contrast, our approach can handle branches at every node in the quantifier hierarchy and our solving step is tightly integrated into the algorithm. Other parallel solving approaches [11, 12] are conceptionally very different to our solution.

6 Conclusion and Future Work

We presented a QBF solving algorithm for QBFs in negation normal form, which extends our previous algorithm [18] to non-prenex formulas together with new optimizations and parallelization. Our evaluation shows that the parallelization is beneficial for our case study and other experiments suggests this method is more broadly applicable. Adapting the certification from [18] for non-prenex formulas is left for future work. Further, it would be interesting to try non-syntactic unprenexing methods to improve the parallelization, e.g., expansion of variables that combine otherwise independent subformulas.

References

1. Biere, A.: PicoSAT essentials. JSAT 4(2-4), 75–97 (2008)
2. Bohy, A., Bruyère, V., Filiot, E., Jin, N., Raskin, J.: Acacia+, a tool for LTL synthesis. In: Proceedings of CAV. LNCS, vol. 7358, pp. 652–657. Springer (2012)
3. Egly, U., Seidl, M., Woltran, S.: A solver for QBFs in negation normal form. Constraints 14(1), 38–79 (2009)
4. Faymonville, P., Finkbeiner, B., Rabe, M.N., Tentrup, L.: Encodings of reactive synthesis. In: Proceedings of QUANTIFY (2015)
5. Finkbeiner, B., Schewe, S.: Bounded synthesis. STTT 15(5-6), 519–539 (2013)
6. Goultiaeva, A., Iverson, V., Bacchus, F.: Beyond CNF: A circuit-based QBF solver. In: Proceedings of SAT. LNCS, vol. 5584, pp. 412–426. Springer (2009)
7. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.M.: Solving QBF with counterexample guided refinement. In: Proceedings of SAT. LNCS, vol. 7317, pp. 114–128. Springer (2012)
8. Janota, M., Marques-Silva, J.: Solving QBF by clause selection. In: Proceedings of IJCAI. pp. 325–331. AAAI Press (2015)
9. Jordan, C., Kaiser, L., Lonsing, F., Seidl, M.: MPIDepQBF: Towards parallel QBF solving without knowledge sharing. In: Proceedings of SAT. LNCS, vol. 8561, pp. 430–437. Springer (2014)
10. Klieber, W., Sapra, S., Gao, S., Clarke, E.M.: A non-prenex, non-clausal QBF solver with game-state learning. In: Proceedings of SAT. LNCS, vol. 6175, pp. 128–142. Springer (2010)
11. Lewis, M.D.T., Schubert, T., Becker, B.: QmiraXT - A multithreaded QBF solver. In: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV), Berlin, Germany, March 2-4, 2009. pp. 7–16. Universitätsbibliothek Berlin, Germany (2009)
12. Lewis, M.D.T., Schubert, T., Becker, B., Marin, P., Narizzano, M., Giunchiglia, E.: Parallel QBF solving with advanced knowledge sharing. Fundam. Inform. 107(2-3), 139–166 (2011)
13. Lonsing, F., Biere, A.: Nenofex: Expanding NNF for QBF solving. In: Proceedings of SAT. LNCS, vol. 4996, pp. 196–210. Springer (2008)
14. Mota, B.D., Nicolas, P., Stéphan, I.: A new parallel architecture for QBF tools. In: Proceedings of HPCS. pp. 324–330. IEEE (2010)
15. Pigorsch, F., Scholl, C.: Exploiting structure in an AIG based QBF solver. In: Proceedings of DATE. pp. 1596–1601. IEEE (2009)
16. QBF Gallery 2014: QCIR-G14: A non-prenex non-CNF format for quantified Boolean formulas http://qbf.satisfiability.org/gallery/qcir-gallery14.pdf
17. Rabe, M.N., Tentrup, L.: CAQE: A certifying QBF solver. In: Proceedings of FMCAD. pp. 136–143. IEEE (2015)
18. Tentrup, L.: Solving QBF by abstraction. CoRR abs/1604.06752 (2016), https://arxiv.org/abs/1604.06752
19. Tu, K., Hsu, T., Jiang, J.R.: QELL: QBF reasoning with extended clause learning and levelized SAT solving. In: Proceedings of SAT. LNCS, vol. 9340, pp. 343–359. Springer (2015)