On Unordered BDDs and Quantified Boolean Formulas

Mikoláš Janota

IST/INESC-ID, Lisbon, Portugal

Abstract. This paper proposes to study the synthesis of unordered binary decision diagrams (BDDs) using solvers for Quantified Boolean Formulas (QBF). The synthesis of a BDD falls naturally in the realm of quantified formulas as we are typically looking for a BDD satisfying a certain specification. This means that we ask whether there exists a BDD such that for all inputs the specification is satisfied. We show that this query can be encoded naturally into QBF and experimentally evaluate these queries for the minority function.
The short paper should be seen as a challenge for further research on QBF solving.

1 Introduction

Reduced and ordered BDD (ROBDDs) [2,1] are well studied and appear often in practice due to their canonicity and ease of manipulation. At the same time, exponential lower bounds for ROBDDs are well-known [17]. In that aspect, BDDs (also called branching programs) are interesting because there are no known exponential lower bounds.[1] This means that BDDs are likely to give us small representations of Boolean functions while preserving some of the advantageous properties of ROBDDs. Namely, they can be naturally represented in hardware, where each node corresponds to a 2 to 1 multiplexer.
This short paper has the following two contributions.

1. We show that synthesizing a BDD can be naturally formulated as a quantified Boolean formula (QBF).
2. We perform preliminary evaluation of state-of-the-art QBF solvers on this formulation.

The preliminary evaluation shows that the nowadays solvers scale poorly. In some sense this is not surprising because synthesis of functions is inherently a hard problem. Already in the case of the synthesis of the disjunctive normal form (DNF) minimization is Σ_2^P-complete [18]. It is only to be expected to be harder for BDDs. Limitations of QBF solvers have also been observed in other related works on *synthesis of circuits* or *reactive systems* [7,11,4,10,3].

We believe that these poor results should not be seen as a deterrent but rather as a challenge for further QBF research.

[1] For existing lower bounds for BDD see a survey by Razborov [14].

2 Preliminaries

Standard concepts from propositional logic are assumed. Propositional formulas are built from variables, negation ($\neg$), and conjunction ($\wedge$). For convenience we also consider the constants $0, 1$ representing false and true, respectively. The results immediately extend to other connectives, e.g., $(\phi \Rightarrow \psi) = \neg(\phi \wedge \neg\psi)$, $(\phi \vee \psi) = \neg(\neg\phi \wedge \neg\psi)$. A *literal* is either a variable or its negation. An *assignment* is a mapping from variables to $\{0, 1\}$. Assignments are represented as sets of literals, i.e., $\{x, \neg y\}$ corresponds to $\{x \mapsto 1, y \mapsto 0\}$. For a formula ϕ and an assignment σ, the expression $\phi|_\sigma$ denotes *substitution*, i.e., the simultaneous replacement of variables with their corresponding value.

Quantified Boolean Formulas (QBF). QBFs [9] extend propositional logic by quantifiers over Boolean variables. Any propositional formula ϕ is also a QBF with all variables *free*. If Φ is a QBF with a free variable x, the formulas $\exists x.\, \Phi$ and $\forall x.\, \Phi$ are QBFs with x *bound*, i.e. not free. Note that we disallow expressions such as $\exists x. \exists x.\, x$. Whenever possible, we write $\exists x_1 \dots x_k$ instead of $\exists x_1 \dots \exists x_k$; analogously for $\forall$. Semantically a QBF corresponds to a compact representation of a propositional formula. In particular, the formula $\forall x.\, \Psi$ is satisfied by the same truth assignments as $\Psi|_{\{\neg x\}} \wedge \Psi|_{\{x\}}$ and $\exists x.\, \Psi$ by $\Psi|_{\{\neg x\}} \vee \Psi|_{\{x\}}$. Since $\forall x \forall y.\, \Phi$ and $\forall y \forall x.\, \Phi$ are semantically equivalent, we allow writing $\forall X$ for a set of variables X; analogously for $\exists$. A QBF with no free variables is *false* (resp. *true*), iff it is semantically equivalent to the constant 0 (resp. 1).

Binary Decision Diagrams (BDD). A BDD [2] is a rooted directed acyclic graph where each node has two outgoing edges except for two sinks. Each node is labeled by a Boolean variable and the outgoing edges are labeled by 1 and 0, respectively. The two sinks are labeled by 1 and 0, respectively. A BDD unequivocally represents a Boolean function: starting at the root take the 1 edge if the variable labeling the current node is true and take the 0 edge otherwise. Respond "true", if the 1-sink is reached; respond "false", if the 0-sink is reached.

3 Encoding

We assume that we are given N nodes $\mathcal{N} = \{n_1, \dots, n_N\}$ and additionally the sink nodes $\mathsf{sink}_1, \mathsf{sink}_0$. Further, there is a finite set of input variables $\mathcal{I}$. The desired semantics of the resulting BDD is assumed to be specified as a Boolean formula on the input variables. Hence, the objective is to construct a BDD of size N representing the same function as the given Boolean formula. For the purpose of this paper we focus on the minority function, i.e. the formula $\phi(i_1, \dots, i_m)$ that is true if and only if a minority of the input variables are set to 1.[2]

[2] The majority function is obtained by swapping the semantics of 0 and 1, which is easy to do in BDDs.

We present the encoding in three conceptually separate steps: (1) encoding of topology (Section 3.1), (2) encoding of BDD's semantics (Section 3.2), (3) encoding of specification (Section 3.3). This section is concluded by Section 3.3, which discusses some of the technical details of the encoding.

3.1 Topology

To avoid cycles in the constructed BDD we apply the following trick. Without a loss of generality, we assume that a node n_i can be only connected to a node m if $m = n_j$ for $j < i$ or m is one of the sinks. Like so we ensure there is a topological ordering on the resulting graph and at the same time, we are not losing any graphs because a topological ordering has to exist (there are no cycles). For convenience, we define the notation $\mathsf{neighbors}(n_i)$ to denote the set $\{n_j \mid j \in 1..i-1\} \cup \{\mathsf{sink}_0, \mathsf{sink}_1\}$.

The space of BDDs is modeled by two sets of Boolean variables. The variables $l_{n,x}$ represent that the node $n \in \mathcal{N}$ is labeled by the input variable $x \in \mathcal{I}$. The variables $c^e_{n,m}$ represent that there is an edge from n to m labeled by $e \in \{0,1\}$. To ensure that each node is labeled by one and only one input variable and that each edge goes into one and only one node we output the following constraints.

$$\sum_{x \in \mathcal{I}} l_{n,x} = 1, \text{for each } n \in \mathcal{N}$$

$$\sum_{m \in \mathsf{neighbors}(n)} c^e_{n,m} = 1, \text{for each } n \in \mathcal{N}, e \in \{0,1\}$$

These constraints together with the restriction that a node can only connect to a sink or one of the preceding nodes yield BDDs with the correct topology.

3.2 Semantics

The semantics of the BDD under a given input is captured by assigning a value to each sub-BDD. This is done recursively as follows. The sink_0 is false, the sink_1 is true. A node n labeled by x is true iff the corresponding neighbor is true.

We introduce the following auxiliary formulas. For a node $n \in \mathcal{N}$ the formula $\mathcal{V}_n$ represents the value of the labeled variable, i.e. the formula is true if the node is labeled by one of the variables x and that variable is true at the same time. Formally defined as:

$$\mathcal{V}_n \triangleq \bigvee_{x \in \mathcal{I}} (l_{n,x} \wedge x)$$

For each sub-BDD rooted in some node $n \in \mathcal{N} \cup \{\mathsf{sink}_0, \mathsf{sink}_1\}$ the formula $\mathcal{T}_n$ represents the truth value of that sub-BDD. For each node $n \in \mathcal{N}$ the formula

$\mathcal{B}_n^e$ represents the truth value of the neighbor of n on the edge e.

$$\begin{aligned}
\mathcal{T}_{\mathsf{sink}_0} &\triangleq 0 \\
\mathcal{T}_{\mathsf{sink}_1} &\triangleq 1 \\
\mathcal{B}_n^e &\triangleq \bigvee_{m \in \mathsf{neighbors}(n)} (c_{n,m}^e \wedge \mathcal{T}_m) \\
\mathcal{T}_n &\triangleq (\neg \mathcal{V}_n \wedge \mathcal{B}_n^0) \vee (\mathcal{V}_n \wedge \mathcal{B}_n^1)
\end{aligned}$$

3.3 Specification

We assume that we are given a formula ϕ on the input variables $\mathcal{I}$ that specifies the behavior of the BDD that we wish to construct, i.e. the constructed BDD should be true if and only if ϕ is true on any given input. This is now easily expressed as a QBF with two levels of quantification where the first level is over the labeling and connecting variables and the second level is over the input variables. In the parlance of the above-defined concepts, we need to ensure that the truth value of the root node is equal to the truth value of ϕ for any input.

To formalize, let $\mathcal{C}$ be the set of variables $c_{n,m}^e$ and let $\mathcal{L}$ be the set of variables $l_{n,x}$. The resulting QBF is defined as follows.

$$\exists \mathcal{C}\, \mathcal{L} \forall \mathcal{I}\,.\, \mathcal{T}_{n_N} \Leftrightarrow \phi$$

In particular, to obtain the minority function over the inputs, we output the following QBF.

$$\exists \mathcal{C}\, \mathcal{L} \forall \mathcal{I}\,.\, \mathcal{T}_{n_N} \Leftrightarrow \left(\sum_{x \in \mathcal{I}} x \leq \lfloor |\,\mathcal{I}\,|/2 \rfloor \right)$$

3.4 Technical Details of The Encoding

The above-defined formulas contain some constructs that are typically not supported by QBF solvers. In the implementation, we use the circuit-like language *QCIR* supported by a number of tools [8]. QCIR supports only typical Boolean connectives. This means that the language does not have native support for the constructs $\sum_{x \in s} x = 1$ or $\sum_{x \in s} x \leq k$. These constraints can be converted to a circuit form in a number of ways. Since we are dealing with rather modest numbers we use the pairwise (quadratic) encoding for "exactly one" and sequential counter [15] for $\mathsf{at\text{-}most}_k()$.

$$\sum_{x \in x_1, \ldots, x_n} x = 1 \triangleq \bigvee_i x_i \vee \bigwedge_{i<j} \neg x_i \vee \neg x_j$$

$$\begin{aligned}
\mathsf{at\text{-}most}_0(S) &\triangleq \bigwedge_{x \in S} \neg x_i \\
\mathsf{at\text{-}most}_k(x_1, \ldots, x_n) &\triangleq x_1 \wedge \mathsf{at\text{-}most}_{k-1}(x_2, \ldots, x_n) \vee \neg x_1 \wedge \mathsf{at\text{-}most}_k(x_2, \ldots, x_n)
\end{aligned}$$

$\lvert\mathcal{I}\rvert/k$	rareqs	qfun	qute	quabs
3/1	0.04	**0.02**	0.18	0.32
4/2	**0.11**	**0.11**	2.28	3.34
5/2	22.55	**17.35**	T/O	T/O
6/3	T/O	T/O	T/O	T/O

(a) Times in seconds

$\lvert\mathcal{I}\rvert/k$	rareqs	qfun	qute	quabs
3/1	4	4	4	4
4/2	6	6	6	6
5/2	9	9	7	8
6/3	10	10	6	7

(b) Lowerbounds for #nodes

Fig. 2. Results and example BDD

3.5 Experimental Evaluation

The encoding to QBF was implemented as a Python script allowing for different QBF solvers to be used as the back-end. The script uses encoding given in Section 3 with an increasing N (the number of nodes). Consequently, the majority of the QBF calls are UNSAT and the synthesized BDD is guaranteed to be the smallest in number of nodes (if found).

The targeted function was the minority function, in fact a special case of $\mathsf{at\text{-}most}_k()$ constraint, with $k = \lfloor\lvert\mathcal{I}\rvert/2\rfloor$.[3] Figure 1 shows one of the synthesized BDDs for $\lvert\mathcal{I}\rvert = 5$, $k = 2$ with 9 BDD nodes (plus the sinks). Interestingly, this BDD is in fact ordered, but to our best knowledge, it is not known whether the smallest BDD for the minority function can be always ordered.

The following QBF solvers were used: Qute [12]; QFUN [5]; QuAbS [16]; and RAReQS [6]. The tables in Figure 2 summarize the results. Figure 2(a) shows the running times (with the timeout 30 minutes). Figure 2(b) shows the lower bound for number of nodes. The solvers QFUN and RAReQS go up to 5 inputs while the solvers Qute and QuAbS stop already on 4 inputs. Interestingly enough, QFUN and RAReQS are still quite fast on 5 inputs.

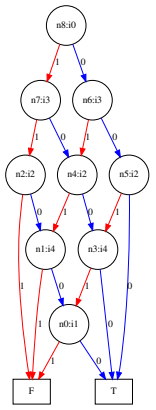

Fig. 1. Example BDD

4 Future Work

We hope that this study will motivate the investigation of other methods for QBF solving; possibly based on more stochastic approaches. The study should also be carried out for other types of functions. Functions from real world, which are typically not symmetrical, might give better picture of the performance and also might be a better target for BDDs.

Acknowledgments. This work was supported by national funds through FCT - Fundao para a Cincia e a Tecnologia with reference UID/CEC/50021/2019 and the project INFOCOS with reference PTDC/CCI-COM/32378/2017.

[3] Pudlák gives a $\Omega(n \lg n)$ lower-bound for this function [13].

References

1. Bahar, R.I., Frohm, E.A., Gaona, C.M., Hachtel, G.D., Macii, E., Pardo, A., Somenzi, F.: Algebraic decision diagrams and their applications. In: Proceedings International Conference on Computer-Aided Design. pp. 188–191 (1993)
2. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. Computers, IEEE Transactions on **100**(8), 677–691 (1986)
3. Faymonville, P., Finkbeiner, B., Rabe, M.N., Tentrup, L.: Encodings of bounded synthesis. In: Tools and Algorithms for the Construction and Analysis of Systems (TACAS). pp. 354–370 (2017). https://doi.org/10.1007/978-3-662-54577-5_20
4. Gascón, A., Subramanyan, P., Dutertre, B., Tiwari, A., Jovanovic, D., Malik, S.: Template-based circuit understanding. In: Formal Methods in Computer-Aided Design (FMCAD) (2014)
5. Janota, M.: Towards generalization in QBF solving via machine learning. In: AAAI Conference on Artificial Intelligence (2018)
6. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.: Solving QBF with counterexample guided refinement. Artificial Intelligence **234**, 1–25 (2016)
7. Jo, S., Matsumoto, T., Fujita, M.: SAT-based automatic rectification and debugging of combinational circuits with LUT insertions. In: Asian Test Symposium. pp. 19–24 (2012)
8. Jordan, C., Klieber, W., Seidl, M.: Non-CNF QBF solving with QCIR. In: Proceedings of BNP (Workshop) (2016)
9. Kleine Büning, H., Bubeck, U.: Theory of quantified boolean formulas. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 735–760. IOS Press (2009)
10. Maksimovic, D., Le, B., Veneris, A.G.: Multiple clock domain synchronization in a QBF-based verification environment. In: International Conference on Computer-aided Design (ICCAD) (2014)
11. Narodytska, N., Legg, A., Bacchus, F., Ryzhyk, L., Walker, A.: Solving games without controllable predecessor. In: Computer Aided Verification (CAV). pp. 533–540 (2014)
12. Peitl, T., Slivovsky, F., Szeider, S.: Dependency learning for QBF. In: Theory and Applications of Satisfiability Testing (SAT). pp. 298–313 (2017)
13. Pudlák, P.: A lower bound on complexity of branching programs. In: Chytil, M., Koubek, V. (eds.) Mathematical Foundations of Computer Science. vol. 176, pp. 480–489. Springer (1984). https://doi.org/10.1007/BFb0030331
14. Razborov, A.A.: Lower bounds for deterministic and nondeterministic branching programs. In: Fundamentals of Computation Theory, 8th International Symposium, FCT. pp. 47–60 (1991). https://doi.org/10.1007/3-540-54458-5_49
15. Sinz, C.: Towards an optimal CNF encoding of boolean cardinality constraints. In: Principles and Practice of Constraint Programming (CP). pp. 827–831. Springer (2005), https://doi.org/10.1007/11564751_73
16. Tentrup, L.: Non-prenex QBF solving using abstraction. In: Theory and Applications of Satisfiability Testing (SAT). pp. 393–401 (2016)
17. Tveretina, O., Sinz, C., Zantema, H.: An exponential lower bound on OBDD refutations for pigeonhole formulas. In: Athens Colloquium on Algorithms and Complexity, ACAC. pp. 13–21 (2009). https://doi.org/10.4204/EPTCS.4.2
18. Umans, C., Villa, T., Sangiovanni-Vincentelli, A.L.: Complexity of two-level logic minimization. IEEE Trans. on CAD of Integrated Circuits and Systems **25**(7), 1230–1246 (2006). https://doi.org/10.1109/TCAD.2005.855944