HAL open science

Homing Sequence Derivation with Quantified Boolean Satisfiability

Hung-En Wang, Kuan-Hua Tu, Jie-Hong R. Jiang, Natalia Kushik

To cite this version:

Hung-En Wang, Kuan-Hua Tu, Jie-Hong R. Jiang, Natalia Kushik. Homing Sequence Derivation with Quantified Boolean Satisfiability. 29th IFIP International Conference on Testing Software and Systems (ICTSS), Oct 2017, St. Petersburg, Russia. pp.230-242, 10.1007/978-3-319-67549-7_14 . hal-01678974

HAL Id: hal-01678974

https://inria.hal.science/hal-01678974v1

Submitted on 9 Jan 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Homing Sequence Derivation with Quantified Boolean Satisfiability

Hung-En Wang[†], Kuan-Hua Tu[†], Jie-Hong R. Jiang[†,‡], and Natalia Kushik[§,¶]

[†]Graduate Institute of Electronics Engineering, National Taiwan University, Taiwan
[‡]Department of Electrical Engineering, National Taiwan University, Taiwan
[§]SAMOVAR, CNRS, Télécom SudParis, Université Paris-Saclay, France
[¶]Tomsk State University, Russia

Abstract. Homing sequence derivation for nondeterministic finite state machines (NFSMs) has important applications in system testing and verification. Unlike prior methods based on explicit tree based search, in this work we formulate the derivation of a preset homing sequence in terms of a quantified Boolean formula (QBF). The formulation allows implicit NFSM representation and compact QBF encoding for effective computation. Different encoding schemes and QBF solvers are evaluated for their suitability to homing sequence derivation. Experimental results show the generality and feasibility of the proposed method.

1 Introduction

Model based testing techniques rely on formal specifications of the system under test. Whenever such systems are reactive, i.e., are working in a request-response mode, one of the appropriate formal models to describe the system behaviour is the finite state machine (FSM). Therefore, a significant branch of research in model based testing is devoted to FSM based testing.

Classical FSM based testing techniques, which are known to start with the W-method [4, 18] are mostly based on three main assumptions/steps: 1) to reach a given state from the initial one, 2) to traverse the transitions under each input, and 3) to distinguish the state that was reached from all other FSM states. The derivation of the corresponding test sequences in this case is based on solving state identification problems for the specification FSM [11].

FSM state identification is performed via an application of either distinguishing (for the initial state) or homing/synchronizing (for the current or final FSM state) sequences. The length of these sequences as well as the complexity of their derivation significantly depend on the type of the specification FSM. For distinguishing sequences (DSs), even for complete and deterministic machines the decision problem of DS existence is PSPACE-complete. However, for homing and synchronizing sequences (HSs and SSs) for deterministic complete machines the upper bounds on the corresponding length are known to be polynomial [14].

For nondeterministic FSMs, the complexity upper bounds rise higher. The existence check becomes PSPACE-complete while the length of the shortest HS or

SS for the machine is exponential with respect to the number of states. Development of complex (embedded) systems that can have nondeterministic behaviour due to various reasons, such as limited controllability and observability, therefore motivates studying the possibilities for reducing the complexity, at least for specific FSM classes [19].

In this paper, we consider non-initialized complete nondeterministic FSMs and we propose to improve the performance of HS existence checking and derivation using scalable FSM representation. We note that existing solutions for this problem mostly rely on the derivation of the homing tree which is built based on the successor tree [8] with the proper usage of truncating/termination rules [14]. For nondeterministic machines, not only the width but the height of this tree can grow exponentially before the nodes are truncated and thus, any search of the shortest HS in the homing tree is either length-bounded or requires exponential number of steps. Note that in this case, one of the most costly operations is shown to be the computation of the set of successors of a subset of FSM states [5]. In this paper, we circumvent deriving the homing tree and computing successor state sets.

The enabling technology of our computation is quantified Boolean formula solving, which has been advanced in recent years [12, 6, 17]. Quantified Boolean formulas (QBFs) are an extension to propositional formulas for their allowance of universal and existential quantification over variables. The additional quantifiers make QBFs exponentially more succinct than quantifier-free formulas in encoding many decision problems. Essentially, quantified Boolean satisfiability (QSAT) is PSPACE-complete in contrast to the NP-completeness of its Boolean satisfiability (SAT) counterpart. The generality of QBF and advancement of QSAT motivate our study of HS existence checking and derivation with QBF solving.

We implicitly represent the specification FSM as a Boolean circuit/formula. The HS existence checking and derivation can thereby be reduced to the corresponding QBF solving. In addition, we propose several techniques to enhance the scalability for QBF solving of homing sequences. Experimental results show promising applicability of our method.

The rest of this paper is organized as follows. After introducing backgrounds of homing sequence and QBF in Section 2, we present the QBF encoding of homing sequence computation in Section 3. We discuss some crucial implementation issues in Section 4. Experimental evaluation is then given in Section 5. Finally, we conclude this paper and outline future work in Section 6.

2 Preliminaries

2.1 Finite State Machine and Homing Sequence

A *finite state machine* (FSM) is a five tuple $M = (Q, Q_{init}, I, O, T)$, where Q is a finite set of states, $Q_{init} \subseteq Q$ is the set of initial states, I is the input alphabet, O is the output alphabet, and $T \subseteq Q \times I \times O \times Q$ is the transition relation.

In the sequel, we assume an FSM is uninitialized, that is, $Q_{init} = Q$. Since the initial state set is assumed to be all possible states, we omit specifying Q_{init} in the sequel. We write $|Q|$ to denote the cardinality of the state set Q; we write $|I|$ and $|O|$ to denote the sizes of the input and output alphabets, respectively; we write $|T|$ to denote the number of transitions in T. A *trace* is a sequence of the form q_0, i_1, o_1, q_1, i_2, o_2, $\ldots$, q_n, such that $(q_{k-1}, i_k, o_k, q_k) \in T$ for all $1 \leq k \leq n$.

A *deterministic* FSM (DFSM) is an FSM, where for each current-state input pair $(q, i) \in Q \times I$, there exists at most one output next-state pair $(o, q') \in O \times Q$ such that $(q, i, o, q') \in T$. Otherwise, the machine is a *nondeterministic* FSM (NFSM). A finite state machine is *complete* if for each current-state input pair (q, i), there exists at least one output next-state pair (o, q') such that $(q, i, o, q') \in T$. A finite state machine is called *(fully) observable* if for each current-state input output triple (q, i, o), there exists at most one next-state q' such that $(q, i, o, q') \in T$.

Given an FSM, a *homing sequence* (HS) is an input sequence such that after running the machine under this input sequence, by observing the corresponding output response, the final state after the execution can be uniquely determined. A homing sequence can be either *nonadaptive* (or called *preset*), which is a fixed input strategy regardless of the output response, or *adaptive*, which is an input strategy that determines the next input symbol based on the so-far observed output response. In this work, we consider the problem of finding a preset homing sequence for a complete NFSM.

An uninitialized complete NFSM has the following property.

Proposition 1. *Given an uninitialized complete NFSM, if there exists a homing sequence of length n, then there exists a homing sequence of length $n + 1$.*

It is because given an uninitialized complete NFSM, with a homing sequence of length n, we can easily extend it to a length $n+1$ homing sequence by adding an arbitrary input symbol to the head of the sequence. After taking the first state transition, the possible current states are a subset of all states. Hence, applying the original homing sequence of length n, the final state can be determined by observing the output sequence.

Note that Proposition 1 is especially interesting for non-observable FSMs for which a prolongation of a homing sequence is not necessarily a homing sequence itself. However, it shows that any prefix can be added to a given homing sequence without ruining the property of the final state identification via the observation of an output response.

2.2 Quantified Boolean Formula

A Boolean variable takes a value in the Boolean domain $\mathbb{B} = \{\bot, \top\}$, with $\bot$ and $\top$ representing FALSE and TRUE, respectively. A *Boolean formula* ϕ consists of Boolean variables and Boolean connectives, which we denote negation, conjunction, disjunction, implication, and equivalence by symbols $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$, respectively. A vector of Boolean variables is denoted by a letter in

bold, such as $\boldsymbol{x}$ of variables $(x_1, x_2, \ldots, x_n)$. Given two vectors of Boolean variables $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$, we use "$\boldsymbol{x} = \boldsymbol{y}$" to denote $\bigwedge_{i=1}^{n} x_i \leftrightarrow y_i$, the bit-wise equivalence between $\boldsymbol{x}$ and $\boldsymbol{y}$.

For a Boolean formula ϕ and a Boolean variable x, we use $\phi|_x$ to denote the induced formula obtained from ϕ by assigning variable x to $\top$. Similarly, $\phi|_{\neg x}$ denotes the formula obtained from ϕ by assigning variable x to $\bot$. A *satisfying assignment* is a complete assignment of truth values to each variable that makes the formula evaluate to $\top$. The *on-set* of a Boolean formula ϕ is the collection of its satisfying assignments to ϕ.

A *literal* ℓ is either a Boolean variable x or its negation $\neg x$. A *clause* is a disjunction of literals. A Boolean formula is in the *conjunctive normal form* (CNF) if it is a conjunction of clauses.

A *quantified Boolean formula* (QBF) Φ can be expressed in a *prenex form* as follows.

$$Q_1 x_1, \ldots, Q_k x_k . \phi, \tag{1}$$

where $Q_i \in \{\exists, \forall\}$ is the quantifier over variable x_i, and ϕ is a quantifier-free Boolean formula over variables $x_1, \ldots, x_k$. A variable x_i with $Q_i = \exists$ (respectively $Q_i = \forall$) is referred to as an *existential variable* (respectively a *universal variable*). We call $Q_1 x_1, \ldots, Q_k x_k$ the *prefix* of Φ, denoted Φ.pfx, and call the quantifier-free formula ϕ the *matrix* of Φ, denoted Φ.mtx. A prenex-form QBF is called in the *prenex conjunctive normal form* (PCNF) if the matrix is expressed as a CNF formula. In the sequel, unless otherwise said, we assume a QBF is expressed in PCNF.

Given the QBF Φ of (1), the *quantification level* of variable x_i is defined to be the number of quantifier alternations between the quantifiers $\exists$ and $\forall$ from left (outer) to right (inner) plus one. A QBF is of l *quantification levels* if the number of quantifier alternations between $\exists$ and $\forall$ from Q_1 to Q_k is $l - 1$. In this work, our considered QBFs are of quantification levels 2 or 3.

The QBF $\exists x_1, Q_2 x_2, \ldots, Q_k x_k . \phi$ is true if one of $Q_2 x_2, \ldots, Q_k x_k . \phi|_{x_1}$ and $Q_2 x_2, \ldots, Q_k x_k . \phi|_{\neg x_1}$ is true. On the other hand, the QBF $\forall x_1, Q_2 x_2, \ldots, Q_k x_k . \phi$ is true if both $Q_2 x_2, \ldots, Q_k v_k . \phi|_{x_1}$ and $Q_2 x_2, \ldots, Q_k v_k . \phi|_{\neg x_1}$ are true. A QBF Φ is true if there exist *Skolem functions* for the existential variables of Φ such that substituting the existential variables with their corresponding Skolem functions in Φ.mtx makes the resultant formula a tautology. By duality, a QBF Φ is false if there exist *Herbrand functions* for the universal variables of Φ such that substituting the universal variables with their corresponding Herbrand functions in Φ.mtx makes the resultant formula unsatisfiable. A detailed exposition of Skolem and Herbrand functions can be found in [1].

3 QBF for Bounded-Length Homing Sequence Existence Checking and Derivation

Given a uninitialized complete NFSM $M = (Q, I, O, T)$, we aim at finding a shortest homing sequence. We search from length 1 to the theoretical upper

bound $2^{\binom{|Q|}{2}} - 1$ of a shortest homing sequence [10]. We present the QBF formulation of the bounded homing sequence checking as follows.

Since Q, I, O are all finite, we perform Boolean encoding on the states, input symbols, and output symbols with current-state variables $\boldsymbol{s}$, next-state variables $\boldsymbol{s'}$, input variables $\boldsymbol{x}$, and output variables $\boldsymbol{y}$. Then the transition relation T of the machine can be represented by the characteristic function $T(\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s'})$ in terms of the encoding Boolean variables. In our QBF formulation, we rely on time-frame expansion and denote the variables at the tth time-frame with a superscript index t.

Then the QBF corresponding to the existence of homing sequence of length n can be expressed as follows.

$$\exists \boldsymbol{X}, \forall \boldsymbol{Y}, \forall \boldsymbol{S}, \forall \boldsymbol{S^*}.[\Delta^{(n)}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{S}) \wedge \Delta^{(n)}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{S^*}) \rightarrow (\boldsymbol{s}^n = \boldsymbol{s}^{*n})], \qquad (2)$$

where variables $\boldsymbol{S} = (\boldsymbol{s}^0, \ldots, \boldsymbol{s}^n)$, $\boldsymbol{X} = (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n)$, $\boldsymbol{Y} = (\boldsymbol{y}^1, \ldots, \boldsymbol{y}^n)$, $\boldsymbol{S^*} = (\boldsymbol{s}^{*0}, \ldots, \boldsymbol{s}^{*n})$, and $\Delta^{(n)}$ is the conjunction of the transition relation of n time-frames, i.e., $\Delta^{(n)}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{S}) = \bigwedge_{k=1}^{n} T(\boldsymbol{s}^{k-1}, \boldsymbol{x}^k, \boldsymbol{y}^k, \boldsymbol{s}^k)$ and $\Delta^{(n)}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{S^*}) = \bigwedge_{k=1}^{n} T(\boldsymbol{s}^{*k-1}, \boldsymbol{x}^k, \boldsymbol{y}^k, \boldsymbol{s}^{*k})$. In the expression, the variables $\boldsymbol{s^*}$ are fresh variables as the instantiated versions of their counterparts $\boldsymbol{s}$.

The formula asks whether there exists an input sequence of length n, such that for any two traces with same output response, we can always conclude that the final states of the two traces are the same. Clearly, an input sequence satisfies such a constraint if and only if it is a homing sequence.

Proposition 2. *Formula* (2) *is true if and only if the underlying NFSM has a homing sequence of length* n.

Proposition 3. *If Formula* (2) *is true, then the Skolem functions for variables* $\boldsymbol{X}$ *correspond to a homing sequence of the underlying NFSM.*

4 Implementation

In this section, we discuss some implementation details in generating Formula (2) for QBF solving.

4.1 Input Symbol Encoding

The size of input alphabet may not necessarily be in the form of 2^j for some j. If some binary code is unused in representing any input symbol, the QBF solver may assign the unused code for the existential variables. In this case, the solver can falsify the transition relation and make Formula (2) true. However, the unused code does not correspond to any input symbol and cannot form a 'legal' homing sequence. Hence unused codes for input symbol encoding should be avoided.

There are two methods to eliminate unused input codes. The first one is to modify the matrix of Formula (2) by restricting $\boldsymbol{x}^t$, for $t = 1, \ldots, n$, in Formula (2) to only used codes. Essentially, the characteristic functions expressing

the used codes of $\boldsymbol{x}^1, \ldots, \boldsymbol{x}^n$ are conjuncted with Formula (2). The second one is to assign two or more codes to the same input symbol to make all codes used, which can avoid adding more clauses to Formula (2), and gain flexibility in circuit minimization.

In our implementation, we used $\lceil \log_2 |I| \rceil$ bits to encode the input symbols. Consider the input alphabet I with j symbols. It requires $\lceil \log_2 j \rceil$ bits for the encoding. We let each of the first $2^{\lceil \log_2 j \rceil} - j$ symbols be associated with two consecutive codes, and let each of the rest be associated with one of the remaining codes. For instance, if the input alphabet is $\{a, b, c\}$, both codes "00" and "01" are associated with 'a', and "10" and "11" are associated with 'b' and 'c', respectively.

4.2 Minimization of Transition Relation

To improve the efficiency of QBF solving, it is desirable to simplify the matrix of a QBF. Therefore, minimizing the transition relation of the NFSM under homing sequence derivation helps to simplify Formula (2) and improve QBF solving efficiency.

The characteristic function of the transition relation can be naively built by the on-set of T, i.e., by disjoining the characteristic function of each transition, which corresponds to a conjunction of literals of state, input, and output variables. It can be represented as a Boolean formula or a logic circuit. Two-level or multi-level logic minimization algorithms can be applied to reduce the size of the formula/circuit.

To simplify the QBF matrix, one may also exploit different state encoding methods. In our implementation, we study the effects of binary encoding and onehot encoding[1]. The empirical results on our generated benchmark instances are shown in Table 1, where Column "$|T|$" shows the number of transitions in T of each NFSM, Columns "#gates (bin)" and "#gates (1hot)" show the numbers of gates in the final simplified circuits under binary state encoding and onehot encoding, respectively, and Column "ratio (bin/1hot)" shows the ratio of the gate count of binary encoding to the gate count of onehot encoding. In the experiments, the input encoding method described at the end of Section 4.1 is applied, with the same encoding strategy applied on output symbols. Also, circuit minimization is applied on each case. Note that unlike input encoding, unused state codes do not affect the correctness of QBF analysis. We do not assign multiple codes for one state; otherwise, this encoding may introduce state equivalence in our formula and complicate the homing sequence derivation. Encoding for output symbols has no such an unused code problem, too. In the experiment, however, output is encoded in the same way as the input. As can be seen, binary encoding yields gate counts about 70% to 90% of those yielded by onehot encoding.

[1] By onehot encoding, n states $q_1, q_2, \ldots, q_n$ are encoded with n bits $b_1, b_2, \ldots, b_n$, one for each state, such that state q_i, $i \in \{1, \ldots, n\}$, is coded with $b_i = 1$ and $b_j = 0$ for $j \neq i$.

Table 1. Gate count comparison under different state encodings

case	$\|Q\|/\|I\|/\|O\|$	$\|T\|$	#gates (bin)	#gates (1hot)	ratio (bin/1hot)
0	5/2/2	13	43	64	0.67
1	5/2/2	17	39	60	0.65
2	5/2/2	18	50	76	0.66
3	5/2/2	17	38	54	0.70
4	5/2/2	14	37	58	0.64
5	10/5/5	153	480	531	0.90
6	10/5/5	139	466	566	0.82
7	10/5/5	147	451	527	0.86
8	10/5/5	154	475	591	0.80
9	10/5/5	142	459	536	0.86
10	13/7/7	371	1071	1169	0.92
11	13/7/7	385	1092	1214	0.90
12	13/7/7	384	1067	1197	0.89
13	13/7/7	381	1046	1172	0.89
14	13/7/7	394	1073	1200	0.89
15	15/8/8	517	1435	1758	0.82
16	15/8/8	567	1507	1760	0.86
17	15/8/8	528	1463	1677	0.87
18	15/8/8	539	1421	1723	0.82
19	15/8/8	523	1451	1700	0.85
20	20/10/10	1087	3211	3563	0.90
21	20/10/10	1147	3243	3539	0.92
22	20/10/10	1071	3130	3692	0.85
23	20/10/10	1094	3101	3482	0.89
24	20/10/10	1116	3234	3637	0.89

4.3 QBF Negation for Quantification Level Minimization

Simplifying transition relation is in general desirable. It is unclear, however, whether to represent the transition relation in two-level or multi-level circuits, especially when Tseitin transformation [16] is applied to convert a circuit into a CNF formula for PCNF-based QBF solvers. Tseitin transformation[2] uses intermediate variables in circuit-to-CNF conversion. It makes the final QBF having an extra innermost layer of existential quantification over these intermediate variables. That is, Formula (2), which is of two quantification levels, will become a QBF with three quantification levels of the following form

$$\exists \boldsymbol{X}, \forall \boldsymbol{Y}, \forall \boldsymbol{S}, \forall \boldsymbol{S^*}, \exists Z.\phi, \tag{3}$$

[2] In Tseitin transformation, an intermediate variable is introduced for each internal gate output, and a number of clauses are generated to characterize the relation of consistent valuations between input and output variables of each gate. For example, the circuit in Figure 1(a) can be converted into the CNF formula $(x \vee \neg u) \wedge (y \vee \neg u) \wedge (\neg x \vee \neg y \vee u) \wedge (\neg u \vee w) \wedge (\neg z \vee w) \wedge (u \vee z \vee \neg w)$, in which two intermediate variables u and w are used and the first (resp. last) three clauses describe $u \leftrightarrow (x \wedge y)$ (resp. $w \leftrightarrow (u \vee z)$).

where ϕ is a CNF formula converted from a circuit representing $[\Delta^{(n)}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{S}) \wedge \Delta^{(n)}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{S^*}) \rightarrow (\boldsymbol{s}^n = \boldsymbol{s^*}^n)]$ and variables Z are the intermediate variables introduced in the CNF conversion. Having many such intermediate variables introduced by Tseitin transformation for each internal gate output of the logic circuit may degrade QBF solving performance.

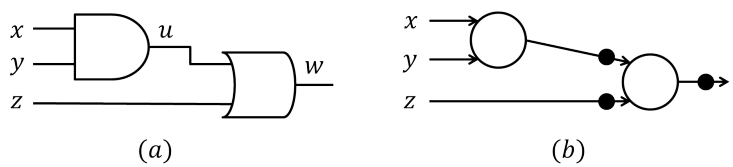

Fig. 1. (a) A logic circuit implementing function $(x \wedge y) \vee z$. (b) An AIG representing the circuit in (a), with each circle representing an AND gate, and a bubble on an edge representing an inverter.

The minimization procedure represents the transition relation in terms of an and-inverter graph (AIG) [13], which consists of 2-input AND gates and inverters. Figure 1(b) shows an example of AIG of the circuit in Figure 1(a), where a circle represents a 2-input AND gate and a bubble on an edge represents an inverter. AIGs allow compact representation of Boolean circuits and are widely used in logic synthesis and verification [7]. As shown in Table 1, since the number of gates in the minimized circuit (AIG) is about three times the number $|T|$ of transitions, the introduced extra variables will be more than those of the on-set approach. To be seen in the experiments in Section 5, the naive on-set representation of the transition relation, which corresponds to a circuit consisting of $|T|$ multi-input AND gates and 1 multi-input OR gate, has only $|T|$ intermediate variables and sometimes makes QBF solving more efficient.

It has been observed that a QBF and its negation often exhibit different solving characteristics [1]. Negating Formula (2) through Tseintin transformation yields

$$\forall \boldsymbol{X}, \exists \boldsymbol{Y}, \exists \boldsymbol{S}, \exists \boldsymbol{S^*}, \exists Z.\psi, \tag{4}$$

where ψ is a CNF formula converting from the circuit representing $\neg[\Delta^{(n)}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{S}) \wedge \Delta^{(n)}(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{S^*}) \rightarrow (\boldsymbol{s}^n = \boldsymbol{s^*}^n)]$ and variables Z are the intermediate variables by the Tseitin conversion. Observe that Formula (4) has only two quantification levels, in contrast to the three quantification levels of Formula (3). The experimental comparison will be shown in Section 5.

5 Experimental Results

The proposed QBF method is tested on a Linux machine with Intel Xeon E5-2630 CPU (2.3 GHz) and 200 GB RAM. Several state-of-the-art QBF solvers are

tested and compared, including DepQBF [12], RAReQS [6], QELL [17], and the 2QBF solver in Berkeley ABC [3, 2]. We randomly generated 25 test cases by the tool FSMTest-1.0 [15] for performance evaluation.[3] Binary encoding is applied, and for input encoding, all the codes are used as discussed in Section 4.1. Circuit minimization is also applied to minimize the transition relation of each case. Then Tseitin transformation is applied to convert the formulas into PCNF for DepQBF, RAReQS, and QELL. For each case, its potential homing sequences of length k, for $k = 1, \ldots, 1023$, are tested under a timeout limit of 7200 seconds. In the experiments, we find a homing sequence by iteratively increasing the length k by one and solving the corresponding formula. This searching strategy ensures that the derived homing sequence is of the minimum length. For the cases where no-homing sequence is found, this searching strategy also guarantees that there exists no homing sequence of length up to the longest length k successfully checked before timeout. Note that one may exploit Proposition 1 to have a binary search-like strategy starting with some $k > 1$. If it finds a homing sequence under k, one can decrease k to look for a shorter homing sequence. Otherwise, one can increase k by some number to look for a longer homing sequence.

Table 2 shows the statistics of different QBF solvers on solving the 25 test cases. The number of states, and the sizes of input and output alphabets of each case are listed in Column "$|Q|/|I|/|O|$". For each solver, Columns "result" show the final answer, which is one of the three outcomes: "SAT" indicating homing sequence found, "UNSAT" indicating no homing sequence exists, and "TO" indicating timeout on testing homing sequence existence under a length greater than the number reported in Columns "len". Columns "time" show the total solving time (in seconds) of each solver up to the length reported in Columns "len". Columns "len" show the longest sequence length successfully checked before termination, which is the length of the found homing sequence for the SAT case, the length upper bound for the UNSAT case, and the last verified length for the TO case.

As can be seen from Table 2, most cases are reported timeout for each solver, with no homing sequence found within length 6 for the 10-state cases to length

[3] We note that the process of FSM generation can be seen as a simple task. However, deriving an FSM with the corresponding properties such as observability, degree of non-determinism, etc., makes this task more complex. In the FSM generation process in [15], a machine that is not observable was automatically dropped to generate another machine with the same cardinality of input/output alphabet and the same number of states, which is observable. Note that our QBF formulation is not limited to observable NFSMs. We experimented with observable FSMs only as even in the observable case the exponential upper bound on the length of homing sequence is known to be attainable. On the other hand, for simplicity, the number of outputs was chosen to be equal to the number of FSM inputs. In total, we generated 25 machines for which the number of inputs varied from 2 to 10 and the number of states was in the range from 5 to 20, correspondingly. Note that, in most of the cases, neither the number of states nor the number of inputs can be represented by an appropriate power of two. The latter allows to better experiment with our heuristics proposed for input/state encoding.

Table 2. Performance comparison of different QBF solvers

		DepQBF			RAReQS			QELL			ABC		
case	$\|Q\|/\|I\|/\|O\|$	result	time	len	result	time	len	result	time	len	result	time	len
0	5/ 2/ 2	SAT	0.07	3	TO	7200	2	SAT	0.04	3	SAT	0.28	3
1	5/ 2/ 2	TO	7200	560	TO	7200	2	TO	7200	70	TO	7200	53
2	5/ 2/ 2	SAT	11.32	5	TO	7200	2	SAT	0.15	5	SAT	0.41	5
3	5/ 2/ 2	TO	7200	9	TO	7200	2	TO	7200	133	UNSAT	5503	1023
4	5/ 2/ 2	TO	7200	7	TO	7200	2	TO	7200	13	TO	7200	13
5	10/ 5/ 5	TO	7200	4	TO	7200	1	TO	7200	5	TO	7200	6
6	10/ 5/ 5	TO	7200	4	TO	7200	1	TO	7200	5	SAT	818	6
7	10/ 5/ 5	TO	7200	4	TO	7200	1	TO	7200	5	TO	7200	6
8	10/ 5/ 5	TO	7200	4	TO	7200	1	TO	7200	5	SAT	5293	7
9	10/ 5/ 5	TO	7200	4	TO	7200	1	SAT	1122	5	SAT	30.18	5
10	13/ 7/ 7	TO	7200	3	TO	7200	1	TO	7200	4	TO	7200	5
11	13/ 7/ 7	TO	7200	3	TO	7200	1	TO	7200	4	TO	7200	5
12	13/ 7/ 7	TO	7200	3	TO	7200	1	TO	7200	4	TO	7200	5
13	13/ 7/ 7	TO	7200	3	TO	7200	1	TO	7200	4	TO	7200	5
14	13/ 7/ 7	TO	7200	3	TO	7200	1	TO	7200	4	TO	7200	5
15	15/ 8/ 8	TO	7200	3	TO	7200	1	TO	7200	3	TO	7200	4
16	15/ 8/ 8	TO	7200	3	TO	7200	1	TO	7200	3	TO	7200	4
17	15/ 8/ 8	TO	7200	3	TO	7200	1	TO	7200	3	TO	7200	4
18	15/ 8/ 8	TO	7200	3	TO	7200	1	TO	7200	3	TO	7200	4
19	15/ 8/ 8	TO	7200	3	TO	7200	1	TO	7200	3	TO	7200	4
20	20/10/10	TO	7200	2	TO	7200	1	TO	7200	3	TO	7200	4
21	20/10/10	TO	7200	2	TO	7200	1	TO	7200	3	TO	7200	4
22	20/10/10	TO	7200	2	TO	7200	1	TO	7200	3	TO	7200	4
23	20/10/10	TO	7200	2	TO	7200	1	TO	7200	3	TO	7200	4
24	20/10/10	TO	7200	2	TO	7200	1	TO	7200	3	TO	7200	4

4 for the 20-state cases. We observed that for the cases with 5 states, each solver seems to show its own strength. DepQBF performs very well on case 1; ABC performs well on case 3; QELL yields a more balanced result compared to the other solvers. In overall performance, ABC outperforms other solvers, with at least one more length verified in each of the larger cases. The only one UNSAT case, reported by ABC, has no homing sequence within length upper bound 1023, and this is in fact the theoretical upper bound of shortest homing sequence ([10]) for a 5-state NFSM. The outstanding performance of ABC is not surprising as the homing sequence QBFs favor a circuit-based solver due to its natural circuit representation of transition relation.

Note that although all solvers timed out on all the cases with 13 and more states, the scalability of the proposed method can still be seen through the longest lengths that successfully verified before timeout in these cases from Table 2. For most of the cases, the successfully checked lengths seem to be small. It suggests that computing homing sequence for NFSM is challenging. In fact, there are exponentially many input sequences of a given length, and for a NFSM the problem of checking whether an input sequence is homing is known to be-

long to the PSPACE complexity class [9]. The complexity of checking if a given sequence is homing for nondeterministic machine is "hidden" in the costly operation of an i-successor [9] of a given state subset. Moreover, the higher is the nondeterminism degree of the machine, the slower is the check that for each state pair and each common output response at these states, the final state is unique. The latter makes it unpromising to directly apply any brute force search or even truncated successor tree approach in a large scale. In this paper, we discuss possible heuristics how this complexity can be reduced via the usage of FSM scalable representations and corresponding QBF solvers.

Table 3. Performance comparison under different formula construction methods

		DepQBF					RAReQS					QELL				
case	$\|Q\|$	m	o	m+c	o+c	o+b	m	o	m+c	o+c	o+b	m	o	m+c	o+c	o+b
1	5	560	14	1023	22	14	2	5	20	20	5	70	25	19	19	25
3	5	9	14	21	22	14	2	6	20	20	6	133	38	19	19	39
11	13	3	2	6	6	2	1	2	6	6	2	4	5	5	6	5
13	13	3	2	6	6	2	1	2	6	6	2	4	5	5	6	5
21	20	2	2	4	4	2	1	2	4	5	2	3	4	4	4	4
23	20	2	2	4	4	2	1	2	4	5	2	3	4	4	4	4

As discussed in Section 4, there can be different options in formula generation. Solver performance may also be affected by the chosen options, especially the PCNF-based solvers, DepQBF, RAReQS, and QELL. In Table 3, we compare solver performance in five different options of formula generation. Six test cases in the above experiment are selected, including two small ones with 5 states, two medium ones with 13 states, and two large ones with 20 states. The three PCNF-based solvers, DepQBF, RAReQS and QELL are compared. Since the 2QBF solver in ABC takes an AIG as its input, it does not need Tseitin transformation and the methods mentioned in Section 4 seem not affecting much the ABC performance. So ABC is excluded in this comparison.

In Table 3, each entry shows the verified length before the timeout, Columns "m" show the result of applying circuit minimization on transition relation without complementing the formula. They are also the results shown in the above experiment. Columns "o" show the results using the on-set of transition relation without minimization and having no formula negation. Columns "m+c" show the results using minimized circuits and applying formula negation. Columns "o+c" show the results using the on-set of transition relation without minimization, but with formula negation. Each of "m", "o", "m+c", "o+c" uses all codes for input and output encodings. On the other hand, Columns "o+b" do not use all codes for encoding, and clauses are added to constrain inputs to legal code assignments. Both circuit minimization and formula negation are not applied in Columns "o+b".

It can be seen that for DepQBF, transition relation minimization is beneficial in most cases. Also, formula negation substantially improves the performance,

with the verified lengths doubled within timeout, and even case 1 reached the pre-specified upper bound 1023 before timeout (about 1542 seconds for solving case 1). On the other hand, for RAReQS, using the onset of transition relation without circuit minimization is better in most of the cases. Moreover, solving the negated formula is also much faster than solving the original formula, with the verified lengths increased to at least 2.5 times. As for QELL, transition relation minimization or solving negated formula significantly improves the solving of the 5-state cases, but the verified lengths slightly drops in the larger cases. Comparing Column "o" and Column "o+b" in any of the three solvers, we see that the ways of handling unused codes in input encoding seem not having notable effects on the solver performance.

6 Conclusions

We have formulated the problem of finding preset homing sequence of an uninitialized NFSM as QBF solving. Different implementation issues in formula construction have been discussed. Experiments have been done comparing different QBF solvers on existence checking and derivation of homing sequences for NFSMs. The effects of circuit minimization and formula negation have been studied. The results have suggested that circuit-based QBF solver ABC is the most powerful one in our applications, while other solvers may not be as effective due to the Tseitin transformation overhead. On the other hand, for PCNF-based solvers, complementing Formula (2), which reduces the number of quantification levels, tends to improve solving efficiency. Moreover, different PCNF-based solvers may have their preferred encoding methods. We believe that the approach proposed in the paper should outperform the classical ones, based on the derivation of the truncated successor tree, but the comparison remains to be done.

For future work, we plan to conduct experiments comparing our approach against the classical methods. We will extend our formulation to finding adaptive homing sequences and to consider initialized NFSMs under partial observability. Moreover, it would be interesting to study how our proposed approach performs on 'hard' FSMs that are known to have the homing sequence but of an exponential length, i.e., to stress-test the QBF solvers over the machines for which the exponential upper bound is reachable. We therefore plan to implement the derivation of such machines, using for example an algorithm given in [10].

Acknowledgements

The authors are grateful to Prof. Nina Yevtushenko for initiating this work and for valuable discussions. This work was supported in part by the joint project between the Ministry of Science and Technology (MOST) of Taiwan and Russian Science Foundation (RSF) of Russia under grants MOST 105-2923-E-002-016-MY3 and RSF 16-49-03012.

References

1. Balabanov, V., Jiang, J.H.R.: Unified QBF certification and its applications. Formal Methods in System Design 41(1), 45–65 (2012)
2. Balabanov, V., Jiang, J.H.R., Scholl, C., Mishchenko, A., Brayton, R.K.: 2QBF: Challenges and solutions. In: SAT. pp. 453–469 (2016)
3. Brayton, R.K., Mishchenko, A.: ABC: An academic industrial-strength verification tool. In: CAV. pp. 24–40 (2010)
4. Chow, T.S.: Testing software design modeled by finite-state machines. IEEE Transactions on Software Engineering 4(3), 178–187 (1978)
5. Haddad, A.R.: Efficient Algorithms for Constructing Preset Distinguishing Sequences for Nondeterministic Finite State Machines. Master's thesis, American University of Sharjah (2016)
6. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.M.: Solving QBF with counterexample guided refinement. In: SAT. pp. 114–128 (2012)
7. Jiang, J.H.R., Devadas, S.: Logic synthesis in a nutshell. In: Wang, L.T., Chang, Y.W., Cheng, K.T. (eds.) Electronic Design Automation: Synthesis, Verification, and Test. Elsevier (2009)
8. Kohavi, Z.: Switching and Finite Automata Theory. McGraw-Hill (1978)
9. Kushik, N.G., Kulyamin, V.V., Evtushenko, N.V.: On the complexity of existence of homing sequences for nondeterministic finite state machines. Programming and Computer Software 40(6), 333–336 (2014)
10. Kushik, N., Yevtushenko, N.: On the length of homing sequences for nondeterministic finite state machines. In: CIAA. pp. 220–231 (2013)
11. Lee, D., Yannakakis, M.: Principles and methods of testing finite state machines - a survey. Proceedings of the IEEE 84(8), 1090–1123 (1996)
12. Lonsing, F., Biere, A.: DepQBF: A dependency-aware QBF solver. Journal on Satisfiability, Boolean Modeling and Computation 7(2-3), 71–76 (2010)
13. Mishchenko, A., Chatterjee, S., Jiang, J.H.R., Brayton, R.K.: FRAIGs: A unifying representation for logic synthesis and verification. In: ERL Technical Report. UC Berkeley (2005)
14. Sandberg, S.: Homing and synchronizing sequences. In: Model-Based Testing of Reactive Systems: Advanced Lectures, pp. 5–33. Springer Berlin Heidelberg (2005)
15. Shabaldina, N., Gromov, M.: FSMTest-1.0: A manual for researches. In: EWDTS. pp. 1–4 (2015)
16. Tseitin, G.: On the complexity of derivation in propositional calculus. In: Studies in Constructive Mathematics and Mathematical Logic, pp. 466–483 (1970)
17. Tu, K.H., Hsu, T.C., Jiang, J.H.R.: QELL: QBF reasoning with extended clause learning and levelized SAT solving. In: SAT. pp. 343–359 (2015)
18. Vasilevskii, M.: Failure diagnosis of automata. Kibernetika (4), 98–108 (1973)
19. Yenigün, H., Yevtushenko, N., Kushik, N.: Some classes of finite state machines with polynomial length of distinguishing test cases. In: SAC. pp. 1680–1685 (2016)