Formal Methods for NFA Equivalence: QBFs, Witness Extraction, and Encoding Verification*

Edith Hemaspaandra[1][0000−0002−7115−626X] and David E. Narváez[2][0000−0003−3704−1060]

[1] Department of Computer Science, Rochester Institute of Technology, Rochester NY, USA 14623
eh@cs.rit.edu

[2] Department of Computer Science, University of Rochester, Rochester NY, USA 14627
david.narvaez@rochester.edu

Abstract. Nondeterministic finite automata (NFAs) are an important model of computation. The equivalence problem for NFAs is known to be PSPACE-complete, and several specialized algorithms have been developed to solve this problem. In this paper, we approach the equivalence problem for NFAs by means of another PSPACE-complete problem: quantified satisfiability (QSAT). QSAT asks whether a quantified Boolean formula (QBF) is true. We encode the NFA equivalence problem as a QBF which is true if and only if the input NFAs are not equivalent. In addition to determining the equivalence of NFAs, we are also able to decode strings that witness the inequivalence of two NFAs by looking into the solving certificate. This is a novel application of certified QSAT solving. Lastly, we formally verify key aspects of the encoding in the Isabelle/HOL theorem prover.

Keywords: QSAT · QBF · Finite Automata · Interactive Theorem Proving · Isabelle/HOL · Formal Methods.

1 Introduction

Finite automata are fundamental concepts in computer science. In theoretical computer science education, for example, the contrast between deterministic and nondeterministic finite automata is commonly used to introduce students to the idea of nondeterminism. In practical settings, examples are lexical analysis in compilers [1] and bounded model checking [19].

Equivalence of two finite automata is a classic computational problem. For *deterministic finite automata* (DFAs), Hopcroft's algorithm [12] runs in near-linear time. For *nondeterministic finite automata* (NFAs), the equivalence problem is PSPACE-complete. Recent work by Bonchi and Pous [3,4] uses bisimulation and coinduction to determine the equivalence of two NFAs. From the theoretical

* Research supported in part by NSF grant DUE-1819546 and NSF grant CCF-2030859 to the Computing Research Association for the CIFellows Project.

point of view, NFA equivalence being in PSPACE means it can be encoded as an instance of any other problem that is PSPACE-complete. One such problem is QSAT: the problem of determining whether a quantified Boolean formula (QBF) is true. The ideas behind the proof of PSPACE-completeness of QSAT were used by Jussila and Biere [19] to generate short QBFs encoding the bounded model checking problem where given one automaton one wants to check that no bad state is reachable. Ultimately, their findings suggested that QSAT solvers were not a feasible tool at the time. Nevertheless, in the last decade there has been increasing interest in developing new ideas for QSAT solving and much has been advanced in terms of algorithms and tools for this problem. In particular, new circuit-based formula formats have been introduced which aim at overcoming the limitations of clause-based QSAT solving [18]. Part of our research studies whether the state of the art in non-clausal QSAT solving supports an alternative method to solve NFA equivalence via a workflow that includes encoding the problem as a QBF, solving it through QSAT solvers, and obtaining domain-specific information from the solving process. Our results indicate that, despite the great advances in the area, most solvers struggle with even small instances of NFA equivalence problems. On the other hand, we show that in the case when two NFAs are found to be not equivalent using this method, it is possible to use current technology for QSAT certification to extract a witness string.

Another part of our research seeks to provide a verified encoder of the NFA equivalence problem which can output formulas that can then be passed to solvers. Such a development belongs to the very active area of research that seeks to prove not only that the conceptual ideas of encoding a problem using (in our case, quantified Boolean) constraints is correct but also providing an implementation that matches the conceptual ideas [14,6].

The rest of this paper is structured as follows. Section 2 goes through the definitions of DFAs and NFAs, plus the basic definitions of quantified Boolean formulas. Section 3 explains the details of the QBF encoding of the problem of inequivalence of NFAs. Section 4 explains the process of extracting a *witness string* out of the solving certificate generated by a QSAT solver. Section 5 presents some experimental results regarding solving these formulas in practice. Section 6 discusses the details of the proof of correctness of a key part of the encoding. Finally, we conclude in Section 7 and give some directions for future work.

2 Background

A *finite automaton* is a computational model defined by a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where Q is a set of states, Σ is the alphabet, δ is a transition function to move between the states as the automaton reads an input string, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states. M accepts string w if M ends in a final state after reading w. The *language* $L(M)$ of a finite automaton M is the set of all strings that M accepts, and two automata M and M' are *equivalent* if $L(M) = L(M')$. If two automata are not equivalent, there is a *witness string*

that one automaton accepts, but the other automaton rejects. The key difference between a *deterministic finite automaton* (DFA) and a *nondeterministic finite automaton* (NFA) is the nature of the transition function δ to move between the states. For DFAs, $\delta : Q \times \Sigma \to Q$ indicates the next state after reading a symbol σ at a state q. For NFAs, $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ indicates the set of possible next states after either reading a symbol $\sigma \in \Sigma$ at a state q for $\delta(q, \sigma)$, or not reading a symbol from the input string for $\delta(q, \epsilon)$ (these are called ϵ-transitions). Despite their difference, DFAs and NFAs accept the same class of languages since an NFA can be transformed into a DFA that accepts the same language, though this transformation may incur in an exponential blow-up in the size of the state set.

Boolean *satisfiability* (SAT) is the problem of determining whether a propositional Boolean formula is satisfiable, i.e., whether there exists an assignment of the variables of a formula φ that makes φ evaluate to *true*. This is arguably the most famous NP-complete problem. Propositional Boolean formulas are typically described, as we do in Section 3, using conventional operations like $\wedge$ (logical and), $\vee$ (logical or), and $\to$ (implication). An alternative way to think about propositional Boolean formulas is as combinational circuits. The inputs of a circuit representing a propositional Boolean formula φ are the variables of φ, and the satisfiability problem translates to determining whether there is an assignment of values to the inputs of the circuit such that the output of the circuit is *true*.

By including the universal quantifier $\forall$ (meaning *for all*) and the existential quantifier $\exists$ (meaning *there exists*), we get a *quantified boolean formula* (QBF). The problem of determining whether a QBF is satisfiable (QSAT) is complete for PSPACE, a problem class that is assumed to strictly contain NP [30].

As we mentioned before, determining if two NFAs are equivalent is a PSPACE problem, so QBFs can be used to encode this problem. We are interested in comparing the bisimulation approach to the QSAT approach for different classes of NFAs, starting with randomly generated NFAs.

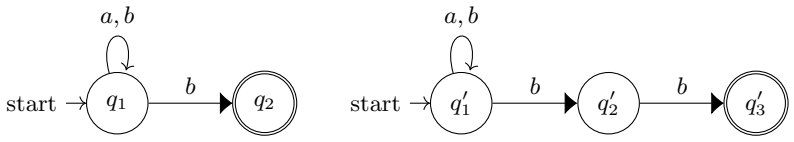

Fig. 1: NFAs N (left) and N' (right). The alphabet Σ is $\{a, b\}$, and final states are drawn with a double circle. These NFAs are inequivalent since N accepts the witness string ab, but N' does not.

3 QBF Encoding

As mentioned in the introduction, NFA equivalence is PSPACE-complete, and so this problem can be encoded in polynomial time as a QBF. In this section, we will show that it is remarkably straightforward to encode NFA inequivalence into a QBF formula. This is particularly remarkable since NFA inequivalence is a nondeterministic problem at heart. Our formula is closely related to the one Stockmeyer and Meyer used to prove PSPACE-hardness of QSAT in their seminal 1973 paper [30]. That paper points out the similarity between their construction and the one from Savitch's proof showing that NPSPACE = PSPACE [29], and the approach we use reveals that the construction from [30] also shows that QBF is NPSPACE-hard, and so provides an alternative proof of PSPACE = NPSPACE. We point out that the formula we use is closely related to the QBF encoding of Savitch's Theorem from [27] (the naïve encoding of Savitch's Theorem produces an existential formula of exponential length). Finally, for readers with background in bounded model checking, this encoding will resemble a double application of the *non-copying iterative squaring method* [19] as we are interested in simultaneously solving reachability problems in two automata.

Consider two NFAs $N = (Q, \Sigma, \delta, q_0, F)$ and $N' = (Q', \Sigma, \delta', q_0', F')$. For simplicity, in this explanation we assume that these NFAs do not have ϵ-transitions (those can easily be handled by a quick preprocessing step that does not increase the number of states). For $S \subseteq Q$ and $\sigma \in \Sigma$ we let $\delta(S, \sigma) = \bigcup_{s \in S} \delta(s, \sigma)$ (and similarly for N'), and we extend the definition of δ and δ' to strings in the obvious and standard way.

For $S, T \subseteq Q$ and $S', T' \subseteq Q'$, we can construct a QBF $\varphi_k(S, S', T, T')$ that is true if and only if there exists a string w of length at most k such that $\delta(S, w) = T$ and $\delta(S', w) = T'$.

N and N' are inequivalent if and only if there is a witness string w of length at most ℓ, the choice of which is discussed below, that is accepted by one NFA and not by the other. This is to say that N and N' are inequivalent if and only if

$$\exists T.\ \exists T'.\ [\varphi_\ell(\{q_0\}, \{q_0'\}, T, T') \wedge (T \cap F = \emptyset \text{ iff } T' \cap F' \neq \emptyset)].$$

For simplicity, we take k to be a power of 2. We define φ_k recursively as follows. For the case $k > 1$, $\varphi_k(S, S', T, T')$ is true if and only if there exist intermediate sets of states R, R' such that for all sets of states X, X', Y, Y', we have[3]

$$\begin{aligned}\varphi_k(S, S', T, T') := \exists R, R'.\ \forall X, X', Y, Y'.\ [((S, S', R, R') = (X, X', Y, Y') \\ \vee\ (X, X', Y, Y') = (R, R', T, T')) \\ \rightarrow \varphi_{k/2}(X, X', Y, Y')]. \qquad (1)\end{aligned}$$

[3] When we write $A = B$ for sets A and B, we mean that the standard representation of A and B as arrays of Boolean values is equal, i.e., $A = B \equiv \bigwedge(a_i \leftrightarrow b_i)$. Similarly, when we quantify over a set we mean that we quantify over the Boolean variables representing the set.

For the base case of $k = 1$, we have

$$\varphi_1(S, S', T, T') := \Big[S = T \wedge S' = T'\Big] \\ \vee \bigvee_{\sigma \in \Sigma} \Big[\delta(S, \sigma) = T \wedge \delta'(S', \sigma) = T'\Big]. \tag{2}$$

Equation 2 encodes that either the sets S, T (resp. S', T') are equal (and thus $\delta(S, \epsilon) = S = T$ and $\delta(S', \epsilon) = S' = T'$) or that for some $\sigma \in \Sigma$, $\delta(S, \sigma) = T$ and $\delta'(S', \sigma) = T'$.

It is easy to see that $2^{n+n'} - 1$ is an upper bound for ℓ, where n and n' are the number of states in N and N', respectively. Note that this implies that φ_ℓ can be computed in polynomial time, since the depth of recursion is at most $n + n'$ and the size of φ_k is at most $c(n + n')$ plus the size of $\varphi_{k/2}$, where c is a fixed constant.

We obtain this $2^{n+n'} - 1$ upper bound by converting N and N' to equivalent DFAs (of size at most 2^n and $2^{n'}$, respectively) using the subset construction [26] and then using the Cartesian product construction (see [13]) to obtain a DFA for the symmetric difference of size at most $2^n 2^{n'} = 2^{n+n'}$. If this DFA accepts anything, it will accept a string of length at most $2^{n+n'} - 1$. It is important to note that the upper bounds on the number of states are tight (for the subset construction [21,23] and for the Cartesian product construction [34]).

We can do a bit better by not converting the NFAs to DFAs. Though the Cartesian product construction does not give an NFA for the symmetric difference of two NFAs, it does work just fine for the intersection of two NFAs. Note that x is in the symmetric difference of $L(N)$ and $L(N')$ if and only if $x \in L(N) \cap \overline{L(N')}$ or $x \in \overline{L(N)} \cap L(N')$. An NFA that accepts $\overline{L(N)}$ has size at most 2^n. This gives an NFA of size at most $n2^{n'}$ for $L(N) \cap \overline{L(N')}$ and an NFA of size at most $2^n n'$ for $\overline{L(N)} \cap L(N')$, and so a witness string of length at most $\max(n2^{n'}, 2^n n') - 1$. Here also the upper bounds on the number of states are tight (for computing the complement of an NFA [17] and for the intersection of two NFAs [11]).

Our arguments above use the number of states - 1 as an upper bound on the length of a shortest witness. State complexity, whether deterministic or nondeterministic, is well-studied. But it is not inconceivable that the length of a shortest witness is less than the number of states - 1. Compared to state complexity, little is known about this very natural problem. The only paper that looks at this problem for basic operations on finite automata is [2], where it is shown that for all $m, m' \geq 1$, there exist two deterministic finite automata M and M' with m and m' states respectively such that the length of a shortest string in $L(M) \cap L(M') = mm' - 1$. It is an interesting open question to look at the length of a shortest witness string for the other operations we are looking at, such as disjoint union or complementation.

4 Reading a Witness String from the Certificate

The QBF described in Section 3 is satisfiable if and only if there is a witness string that is accepted by one of the two input NFAs and rejected by the other. In the event that the solver finds the formula to be satisfiable, it would be desirable to obtain such a witness string from the solving process. Nevertheless, unlike SAT solvers, which will output a satisfying assignment if an input formula is found to be satisfiable, the yes/no output of a QSAT solver does not really convey the reason why a formula is satisfiable. (Some QSAT solvers do output the settings for the top-level variables in the case that the top-level quantifier is existential.) This, in the context of our application, means that decoding a witness string solely from the solver output is impossible.

Fortunately, the QSAT community, in an effort to improve the reliability of solvers and tools, has invested in developing ways to certify the execution of QSAT solvers [20,9]. This is akin to, and draws from, the similar effort that happened in the SAT solver community in order to certify unsatisfiability [7,10,33].[4] The additional information obtained from the certificate of satisfiability output by a QSAT solver execution on our QBFs contains the information necessary to construct a witness string. In this section we detail the process of extracting such a string.

4.1 What is a QSAT certificate?

In order to certify the satisfiability of a QBF it is enough to provide so-called Skolem functions to replace variables quantified by an existential quantifier. The certification process consists of carrying out the replacement and verifying that the resulting formula (which would only contain variables quantified by universal quantifiers, but essentially a propositional Boolean formula) is a tautology.

In practice, the Skolem functions of the certificate are output as a (combined) circuit. The inputs of the circuit are the universal variables, and the outputs of the circuit are the existential variables. The output corresponding to an existential variable x depends only on the values of the inputs corresponding to universal variables that precede x in the quantifier. In particular, if the top-level quantifier of a QBF is existential (as is the case for the formulas we deal with in this paper), then the outputs corresponding to top-level variables depend on no input and are thus constant.

4.2 Processing the Certificate

Recall that φ_k was defined as follows

$$\begin{aligned}\varphi_k := \exists R, R'.\ \forall X, X', Y, Y'.\ [((S, S', R, R') = (X, X', Y, Y')\\ \vee\ (X, X', Y, Y') = (R, R', T, T'))\\ \rightarrow \varphi_{k/2}(X, X', Y, Y')].\end{aligned}$$

[4] It is relevant to point out that satisfiable (propositional) Boolean formulas do not require certificates as the satisfying assignment is in itself the certificate.

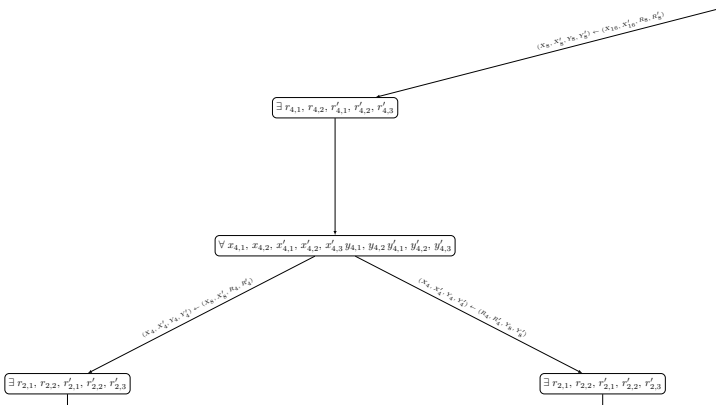

Fig. 2: Detail of the recursion tree for the example in Figure 1

where at every level of the recursive call we are essentially trying to find intermediate state sets R and R' and, once we find them, we will only recurse into $\varphi_{k/2}$ for the specific values of X, X', Y and Y' that coincide with either S, S', R, and R', or with R, R', T, and T'. The values of X, X', Y, and Y' will take the roles of S, S', T, and T' in the next level, and so on until reach φ_1.

Since at every universal level we are only interested in two settings of its variables, we can think of the processing of the certificate of satisfiability as a tree where every node is associated with a quantification level, every existential node has one child and every universal node has two children. This tree has the following invariant: every universal node has $2(n+n')$ variables, its parent node has $n+n'$, and its grandparent node has $2(n+n')$ variables (recall n and n' are the number of states in N and N'). One exception is the top-level: the total number of variables under the top level existential quantifier is $3(n+n')$ but for practical purpose we break this into two levels, one with $2(n+n')$ variables representing the bit vectors $\{q_0\}$, T, $\{q_0'\}$, and T', and another level of $n+n'$ variables representing the sets R and R' from φ_k where k is the upper bound on the length of the witness string. The variables in levels with $n+n'$ variables encode two bit vectors, one for the state set R of the first machine and one for the state set R' of the second machine. Levels with $2(n+n')$ variables encode four bit vectors for either S, S', T, and T', or X, X', Y, and Y', depending on how they are being interpreted (recall the roles of universal variables change at every recursion level).

We process the tree depth-first. At every universal level, we distinguish the left and right children. As we leave the node to arrive at the left child, we set the variables in the current universal level to the following settings:

$$X \leftarrow S \qquad X' \leftarrow S' \qquad Y \leftarrow R \qquad Y' \leftarrow R', \tag{3}$$

and as we leave the current node to arrive at the right child, we set the variables in the current universal level to the following settings:

$$X \leftarrow R \qquad X' \leftarrow R' \qquad Y \leftarrow T \qquad Y' \leftarrow T'$$

Note that the values of S, S', T, T', R, and R' have been set recursively by this process. To illustrate this point, we label the variables at every level according to the following scheme: Note that the upper bound k on the length of a witness string is a power of 2, and at every level of the recursive construction we halve the value of the length of the witness. Then every vector S, S', T, T', R, R', X, X', Y, and Y' is in the context of a length $k' \leq k$ where k' is a power of 2, and we label the variables at every node according to the vector and the length in context, e.g., $x'_{4,2}$ is the second bit of the bit vector representing X' in the context of length 4. Figure 2 shows an example of a recursion tree corresponding to the processing of the certificate of the example in Figure 1. Then the assignments in Equation 3 at level 4 traversing into the left child turn into the following assignments:

$X_4 \leftarrow S_8$	$X'_4 \leftarrow S'_8$	$Y_4 \leftarrow R_4$	$Y'_4 \leftarrow R'_4$
$x_{4,1} \leftarrow x_{8,1}$	$x'_{4,1} \leftarrow x'_{8,1}$	$y_{4,1} \leftarrow r_{4,1}$	$y'_{4,1} \leftarrow r'_{4,1}$
$x_{4,2} \leftarrow x_{8,2}$	$x'_{4,2} \leftarrow x'_{8,2}$	$y_{4,2} \leftarrow r_{4,2}$	$y'_{4,2} \leftarrow r'_{4,2}$
	$x'_{4,3} \leftarrow x'_{8,3}$		$y'_{4,3} \leftarrow r'_{4,3}$

The process continues until a leaf node is reached. At the leaf node, the formula is roughly of the following form. (Note that this formula includes additional constraints we did not include in Formula 2 for simplicity, but that are needed to precisely extract the witness string from the certificate.)

$$\begin{aligned}\varphi_1(S,S',T,T') =& [(S,S') = (T,T')] \\ & \vee \exists \sigma_1, \sigma_2, \ldots, \sigma_{|\Sigma|}. \\ & \left(\textsc{ExactlyOne}(\{\sigma_1, \ldots, \sigma_{|\Sigma|}\}) \wedge \delta(S,\sigma_i) = T \wedge \delta(S',\sigma_i) = T'\right),\end{aligned}$$

where ExactlyOne is the Boolean constraint that sets exactly one of the variables in the parameter set to *true*. (Here we overload the transition function δ to take Boolean variables σ_i as parameters in the place of a symbol in the alphabet Σ, i.e., we assume a one-to-one mapping $\mu : \Sigma \to \{\sigma_1, \ldots, \sigma_{|\Sigma|}\}$ and $\delta(S, \sigma_i)$ stands for $\delta(S, \mu^{-1}(\sigma_i))$.)

At the time the process reaches a leaf node, the values of all the variables except the σ_i variables are set by the process at previous recursion levels. Thus the only thing we need to do at a leaf node is test whether $(S, S') = (T, T')$, in which case we do not output anything since it means the witness string at this position is ϵ. If it is not the case that $(S, S') = (T, T')$, then there is exactly one of the σ_i symbols that is *true*, and that will be the the symbol at this position of the string.

5 Experiments

All the experiments mentioned in this section ran on RIT's Research Computing cluster [28]. Each node is an Intel® Xeon® Gold 6150 CPU @ 2.70GHz. All jobs were limited to 2 GB of RAM memory. The code implementing the ideas in Sections 3 and 4, as well as the dataset and output from the runs described in Section 5.2 are all available in the supplemental materials of this paper[5].

We start this section with a description of the random model used to generate test instances for our QBF encoder and our witness string extraction algorithm.

5.1 Random Model

We used the model proposed by Tabakov and Vardi [31] to generate random NFAs. This model is parameterized by the number of states n, a *transition density* $r = k/n$ where k is the expected total number of transitions in the NFA that are labeled by a fixed symbol $\sigma \in \Sigma$ (hence the total expected number of transitions is $k|\Sigma|$), and a *final state density* $f = m/n$ where m is the expected number of final states. In order to compare our results to previous work by Bonchi and Pous [3], we fix $r = 1.25$ in our NFA generation process. This number was picked in [3] because "Tabakov and Vardi empirically showed that one statistically gets more challenging NFAs with this particular value," though this claim in [31] is with respect to the size of the minimum equivalent DFA problem. The relationship between minimum equivalent DFAs and the equivalence problem of NFAs (which is the problem we address in this paper) is unclear.

We also fix the alphabet $\Sigma = \{a, b\}$. We generated 35 NFAs using different numbers of states and values for f: 5 NFAs with $n = 2$ and $f = 0.05$[6], and 5 NFAs with $n =\in \{3, 4\}$ and $f \in \{0.25, 0.50, 0.75\}$. Per the Tabakov and Vardi model, the start state is also always a final state. We encoded the inequivalence problem of every (ordered) pair of (not necessarily distinct) NFAs in the dataset as a QBF for a total of 1225 QBFs.

As one reads the results in Section 5.2, it is important to keep in mind that the HKC algorithm by Bonchi and Pous, which is the current state of the art, processed all 1225 equivalence problem instances in our data set in less than 5 seconds.

5.2 Determining NFA Equivalence via QBFs

We ran the QSAT solvers from the QBF Eval 2020 competition[7] with default flags with some exceptions due to either availability or limitations in our experimental setup. The complete list of solvers used is: CQESTO [15] (in expert mode), QFUN [16] (in expert mode), QuAbS [32,8], and Qute [25],

[5] `https://doi.org/10.5281/zenodo.6896217`

[6] We generated only 5 NFAs with two states because r and f do not change a two-state NFA in a meaningful way.

[7] `http://www.qbflib.org/solver_view_domain.php?year=2020`

Table 1: Results with a timeout of 15 minutes per instance. OOM is out-of-memory.

solver	flags	solved	timeout	OOM	total	fraction solved
CQESTO	`-es`	550	675	0	1225	0.45
GhostQ		550	675	0	1225	0.45
QFUN	`-caps -i64 -n4 -b4 -S4`	61	135	1029	1225	0.05
QFUN	`-caps -i64 -n4 -b4 -S4 -d`	110	1113	2	1225	0.09
QuAbS		1000	225	0	1225	0.82
Qute		586	639	0	1225	0.48

Table 1 shows results for solving the NFA equivalence instances using solvers that take QCIR-14 as input. In general, most solvers determined the equivalence of about the same fraction of the problems (45%) within the timeout of 15 minutes. This fraction corresponds to the 550 instances where the number of combined states is at most 6. (In the case of QFun, we had to disable the learning feature to achieve comparable performance, since learning in these formulas seems to require too much memory.) One notable exception is QuAbS which solved 82% of the cases. Taking a close look at the number of instances in which QuAbS timed out, from Table 1 one can deduce that QuAbS timed out exactly on the 15×15 instances in which both NFAs had 4 states (i.e., QuAbS solved every instance in which the combined number of states is at most 7).

The analysis above focuses on the combined number of states and suggests that the hardness of these instances is solely tied to that parameter. However, some of the data we collected suggests otherwise: Qute, although not having a particularly impressive performance over the dataset used for the experiments in this paper, is able to outperform QuAbS consistently on the instance generated from the NFAs depicted in Figure 3. It is natural to think that we mean the two instances generated from the encoding the inequivalence of (N, N') and (N', N) are easy for Qute, but we do not. In fact, we mean the inequivalence problem of the NFAs in Figure 3 taken *in the left-to-right order* as they appear in the figure is easy for Qute, which can determine the satisfiability of the instance in under 5 minutes while QuAbS times out. If we reverse the order of the NFAs in the encoding, both QuAbS and Qute time out on the resulting instance. (Recall that our timeout is 15 minutes.) This supports a deliberate experiment design of ours: even though NFA equivalence is obviously a symmetric relation, we were interested in finding out if, in the practical setting, there is an order of the input NFAs for which solvers perform better. As it turns out, the 36 cases of $n_{\text{total}} = 7$ combined number of states that Qute was able to solve within our timeout are mostly cases (N, N') where (N', N) timed out—in fact, only 2 pairs were such that Qute was able to solve both orderings of the input within the timeout, though it did so in wildly different solving times. This suggest a future research path to identify what properties determine the hardness of an instance and an optimal ordering of the input parameters.

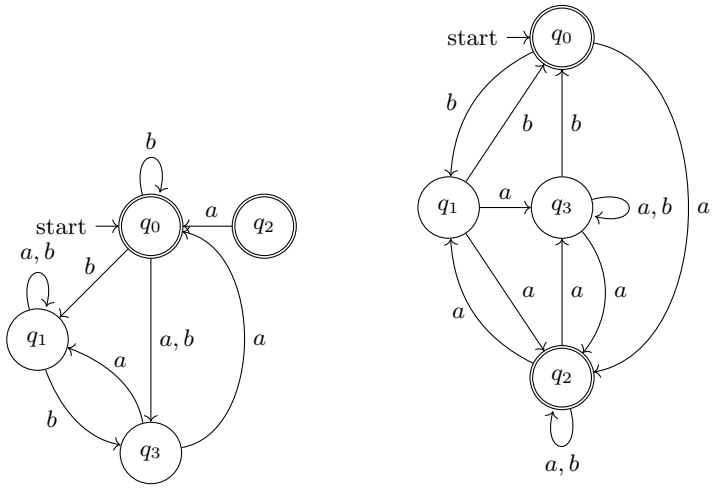

Fig. 3: Two NFAs for which the QBF encoding of their inequivalence problem is easy for Qute and hard for QuAbS.

5.3 Extracting Witness Strings

We implemented the process described in Section 4 and evaluated this implementation on the satisfiable instances of our dataset. Since QuAbS [32,8] is able to generate certificates and uses QCIR-14 as input format, we used it as the solver for these experiments. Table 2 shows the results.

Table 2: Count of witness length in a range k for top-down solving.

k	$[0, 10)$	$[10, 20)$	$[20, 30)$	$[30, 40)$	$[40, 50)$	$[50, 60)$	total
count	319	210	187	92	59	29	896

These results suggest that the majority of the inequivalent instances generated by our random method have witness strings that are much shorter than the upper bound described in Section 2. This in turn means that in order to determine satisfiability for these instances we only needed to generate QBFs that are much smaller than the default.

To exploit the idea of solving smaller formulas in the hope that these smaller formulas are found to be satisfiable, we implemented an iterative solving strategy that solves formulas generated from pairs of NFAs but incrementally choosing a level k from which to start the recursion. This experiment revealed that of the 896 instances found to be satisfiable, 736 of them had witness strings of length at most 1, 116 had witness strings of length at most 2 (but not of length at most 1), and 44 had witness strings of length at most 4 (but not of length at most

2) (Table 3). This suggests that in order to make the QSAT approach to NFA equivalence more feasible, it would be good to have a better understanding of witness strings and the relation between their length and the input NFAs.

Table 3: Count of witness strings of length of at most k obtained through an iterative solving strategy. This strategy generates formulas $\varphi_{2^k}(N, N')$ for $k = 0, 1, 2, \ldots$ until a formula is determined to be SAT.

witness string length	count
≤ 1	736
≤ 2	116
≤ 4	44
total	896

6 Verifying the Encoding in Isabelle/HOL

Despite the fact that the encoding we study in this paper is conceptually easy to describe, it involves several small details that become potential pitfalls when implementing such an encoding. In the supplemental materials[8] we provide a formalization of the base case of our recursive formula φ_k (i.e., φ_1) and we prove its correctness. Our formalization is heavily based on the Boolean Expression Checkers library [24] which provides a language to define Boolean constraints. Unfortunately at the time of this writing that language does not extend to quantified Boolean formulas, and this is the reason why we only prove the correctness of the base case (since it is a QBF with just the existential quantifier, so essentially a propositional Boolean formula). As future work, we intend to tackle extending the language of the Boolean Expression Checkers library to support quantifiers. Ultimately, our goal is to replace our current Java code that implements the encoding of the equivalence problem of two NFAs with verified code obtained directly from the code extraction mechanism in Isabelle/HOL. This will most likely require adjustments to the design decisions of our current formalizations, as some parts of our formalization rely on "abstract" datatypes that cannot be extracted as code.

Most notably, our formalization relies heavily on finite sets (the datatype `fset` in Isabelle/HOL) in an attempt to keep the definitions and theorems very close to those available in textbooks that cover NFAs. Unfortunately, the type system in Isabelle/HOL does not make the interaction between finite sets and sets very easy, and key concepts like transitive closures are only defined for (potentially infinite) sets and not for `fset`s. We expect that the completion of our formalization will in turn contribute a number of theorems that are missing in the Isabelle/HOL library regarding `fset`s.

[8] `https://doi.org/10.5281/zenodo.6896217`

7 Conclusions and Future Work

To summarize, we consider the PSPACE-complete problem of NFA equivalence and evaluate whether tackling this problem through QSAT is feasible, and whether QSAT technology can be used to learn information about the input instance, in particular whether we can use a trace of the solver to learn a witness string that is accepted by one of the input NFAs and rejected by the other. The answer to the first research question is that these formulas still pose serious challenges to non-clausal QSAT solvers, in comparison to the performance of dedicated algorithms. Nevertheless, we show that satisfiability certificates provide a way to extract a witness string from the solving process.

To normalize the results among the different approaches, we will develop and use the same merit function[9] for each approach so that the programming language used (e.g., Bonchi and Pous used OCaml) does not change the results. Related to the HKC algorithm by Bonchi and Pous, we are interested in learning whether there are concepts in specialized algorithms (e.g., "congruence up to") that can be used to develop QSAT algorithms that are specific for the class of QBFs we introduce in this paper.

Regarding the formalization, as immediate future work we will extend the Boolean Expression Checkers library [24] to handle quantifiers and provide our formalization as a case study. It would also be interesting to improve the integration of our formalization with other libraries in Isabelle's Archive of Formal Proofs[10], for example the Transition Systems and Automata library [5].

Acknowledgements

We thank the referees for helpful comments and suggestions. We also thank the AAAI-21 Student Abstract referees for their helpful comments on our preliminary work on this topic [22].

References

1. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools. Addison Wesley, 2 edn. (Aug 2006)
2. Alpoge, L., Ang, T., Schaeffer, L., Shallit, J.O.: Decidability and shortest strings in formal languages. In: Holzer, M., Kutrib, M., Pighizzini, G. (eds.) Descriptional Complexity of Formal Systems - 13th International Workshop, DCFS 2011, Gießen/Limburg, Germany, July 25-27, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6808, pp. 55–67. Springer (2011). https://doi.org/10.1007/978-3-642-22600-7_5, https://doi.org/10.1007/978-3-642-22600-7_5
3. Bonchi, F., Pous, D.: Checking nfa equivalence with bisimulations up to congruence. In: Giacobazzi, R., Cousot, R. (eds.) POPL. pp. 457–468. ACM (2013). https://doi.org/10.1145/2429069.2429124

[9] For example, Bonchi and Pous track the number of processed pairs in their experimental work.

[10] https://www.isa-afp.org/

4. Bonchi, F., Pous, D.: Hacking nondeterminism with induction and coinduction. Commun. ACM **58**(2), 87–95 (2015). https://doi.org/10.1145/2713167
5. Brunner, J.: Transition systems and automata. Archive of Formal Proofs (Oct 2017), `https://isa-afp.org/entries/Transition_Systems_and_Automata.html`, Formal proof development
6. Cruz-Filipe, L., Marques-Silva, J., Schneider-Kamp, P.: Formally verifying the solution to the Boolean Pythagorean triples problem. Journal of Automated Reasoning **63**(3), 695–722 (2019). https://doi.org/10.1007/s10817-018-9490-4
7. Goldberg, E., Novikov, Y.: Verification of proofs of unsatisfiability for CNF formulas. In: Design, Automation and Test in Europe Conference and Exhibition. pp. 886–891. IEEE (Mar 2003). https://doi.org/10.1109/DATE.2003.1253718
8. Hecking-Harbusch, J., Tentrup, L.: Solving QBF by abstraction. In: Proceedings Ninth International Symposium on Games, Automata, Logics, and Formal Verification. Electron. Proc. Theor. Comput. Sci. (EPTCS), vol. 277, pp. 88–102. EPTCS, [place of publication not identified] (2018). https://doi.org/10.4204/EPTCS.277.7
9. Heule, M.J.H., Seidl, M., Biere, A.: A unified proof system for QBF preprocessing. In: International Joint Conference on Automated Reasoning. Lecture Notes in Comput. Sci., vol. 8562, pp. 91–106. Springer (2014). https://doi.org/10.1007/978-3-319-08587-6_7
10. Heule, M.J., Hunt, W.A., Wetzler, N.: Trimming while checking clausal proofs. In: Formal Methods in Computer-Aided Design. pp. 181–188. IEEE (Oct 2013). https://doi.org/10.1109/FMCAD.2013.6679408
11. Holzer, M., Kutrib, M.: Nondeterministic descriptional complexity of regular languages. Int. J. Found. Comput. Sci. **14**(6), 1087–1102 (2003). https://doi.org/10.1142/S0129054103002199, `https://doi.org/10.1142/S0129054103002199`
12. Hopcroft, J.: An $n \log n$ algorithm for minimizing states in a finite automaton. Tech. Rep. STAN-CS-71-190, Stanford University (1971)
13. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
14. Hoque, K.A., Mohamed, O.A., Abed, S., Boukadoum, M.: An automated sat encoding-verification approach for efficient model checking. In: 2010 International Conference on Microelectronics. pp. 419–422. IEEE (Dec 2010). https://doi.org/10.1109/ICM.2010.5696177, `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5696177;https://ieeexplore.ieee.org/document/5696177/`
15. Janota, M.: Circuit-based search space pruning in QBF. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) Theory and Applications of Satisfiability Testing – SAT 2018. pp. 187–198. Springer International Publishing, Cham (2018)
16. Janota, M.: Towards generalization in qbf solving via machine learning. In: McIlraith, S.A., Weinberger, K.Q. (eds.) AAAI. pp. 6607–6614. AAAI Press (2018), `http://dblp.uni-trier.de/db/conf/aaai/aaai2018.html#Janota18`
17. Jirásková, G.: State complexity of some operations on binary regular languages. Theor. Comput. Sci. **330**(2), 287–298 (2005). https://doi.org/10.1016/j.tcs.2004.04.011, `https://doi.org/10.1016/j.tcs.2004.04.011`
18. Jordan, C., Klieber, W., Seidl, M.: Non-CNF QBF solving with QCIR. In: Darwiche, A. (ed.) AAAI Workshop: Beyond NP. AAAI Workshops, vol. WS-16-05. AAAI Press (2016), `http://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12601`

19. Jussila, T., Biere, A.: Compressing BMC encodings with QBF. Electron. Notes Theor. Comput. Sci. **174**(3), 45–56 (2007). https://doi.org/10.1016/j.entcs.2006.12.022, https://doi.org/10.1016/j.entcs.2006.12.022
20. Jussila, T., Biere, A., Sinz, C., Kröning, D., Wintersteiger, C.M.: A first step towards a unified proof checker for QBF. In: Theory and applications of satisfiability testing—SAT 2007. Lecture Notes in Comput. Sci., vol. 4501, pp. 201–214. Springer, Berlin (2007). https://doi.org/10.1007/978-3-540-72788-0_21
21. Meyer, A.R., Fischer, M.J.: Economy of description by automata, grammars, and formal systems. In: 12th Annual Symposium on Switching and Automata Theory, East Lansing, Michigan, USA, October 13-15, 1971. pp. 188–191. IEEE Computer Society (1971). https://doi.org/10.1109/SWAT.1971.11, https://doi.org/10.1109/SWAT.1971.11
22. Miller, H., Narváez, D.E.: Toward determining NFA equivalence via QBFs. In: AAAI-21 Student Abstract (2021), to appear
23. Moore, F.R.: On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata. IEEE Trans. Computers **20**(10), 1211–1214 (1971). https://doi.org/10.1109/T-C.1971.223108, https://doi.org/10.1109/T-C.1971.223108
24. Nipkow, T.: Boolean expression checkers. Archive of Formal Proofs (Jun 2014), https://isa-afp.org/entries/Boolean_Expression_Checkers.html, Formal proof development
25. Peitl, T., Slivovsky, F., Szeider, S.: Dependency learning for QBF. In: Gaspers, S., Walsh, T. (eds.) Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10491, pp. 298–313. Springer Verlag (2017). https://doi.org/10.1007/978-3-319-66263-3_19, http://www.ac.tuwien.ac.at/files/tr/ac-tr-17-011.pdf
26. Rabin, M.O., Scott, D.S.: Finite automata and their decision problems. IBM J. Res. Dev. **3**(2), 114–125 (1959). https://doi.org/10.1147/rd.32.0114, https://doi.org/10.1147/rd.32.0114
27. Rintanen, J.: Partial implicit unfolding in the davis-putnam procedure for quantified boolean formulae. In: Nieuwenhuis, R., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference, LPAR 2001, Havana, Cuba, December 3-7, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2250, pp. 362–376. Springer (2001). https://doi.org/10.1007/3-540-45653-8_25, https://doi.org/10.1007/3-540-45653-8_25
28. Rochester Institute of Technology: Research computing services (2019). https://doi.org/10.34788/0S3G-QD15, https://www.rit.edu/researchcomputing/
29. Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. J. Comput. Syst. Sci. **4**(2), 177–192 (1970). https://doi.org/10.1016/S0022-0000(70)80006-X, https://doi.org/10.1016/S0022-0000(70)80006-X
30. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time (preliminary report). In: Proceedings of the Fifth Annual ACM Symposium on Theory of Computing. pp. 1–9. STOC '73, ACM, New York, NY, USA (1973). https://doi.org/10.1145/800125.804029
31. Tabakov, D., Vardi, M.Y.: Experimental evaluation of classical automata constructions. In: Logic for Programming, Artificial Intelligence, and Reasoning. pp. 396–411. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

32. Tentrup, L.: Non-prenex QBF solving using abstraction. In: Theory and applications of satisfiability testing—SAT 2016, Lecture Notes in Comput. Sci., vol. 9710, pp. 393–401. Springer, [Cham] (2016). https://doi.org/10.1007/978-3-319-40970-2_24
33. Wetzler, N., Heule, M., Jr., W.A.H.: Drat-trim: Efficient checking and trimming using expressive clausal proofs. In: Sinz, C., Egly, U. (eds.) SAT. Lecture Notes in Computer Science, vol. 8561, pp. 422–429. Springer (2014). https://doi.org/10.1007/978-3-319-09284-3_31
34. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theor. Comput. Sci. **125**(2), 315–328 (1994). https://doi.org/10.1016/0304-3975(92)00011-F, `https://doi.org/10.1016/0304-3975(92)00011-F`