Reducing NEXP-complete problems to DQBF

Fa-Hsun Chen
National Taiwan University
r10944015@ntu.edu.tw

Shen-Chang Huang
National Taiwan University
b07902135@ntu.edu.tw

Yu-Cheng Lu
National Taiwan University
luyucheng@protonmail.com

Tony Tan
National Taiwan University
tonytan@csie.ntu.edu.tw

***Abstract*—We present an alternative proof of the NEXP-hardness of the satisfiability of *Dependency Quantified Boolean Formulas* (DQBF). Besides being simple, our proof also gives us a general method to reduce NEXP-complete problems to DQBF. We demonstrate its utility by presenting explicit reductions from a wide variety of NEXP-complete problems to DQBF such as (succinctly represented) 3-colorability, Hamiltonian cycle, set packing and subset-sum as well as NEXP-complete logics such as the Bernays-Schönfinkel-Ramsey class, the two-variable logic and the monadic class. Our results show the vast applications of DQBF solvers which recently have gathered a lot of attention among researchers.**

***Index Terms*—Dependency quantified boolean formulas (DQBF), NEXP-complete problems, polynomial time (Karp) reductions, succinctly represented problems**

I. Introduction

The last few decades have seen a tremendous development of boolean SAT solvers and their applications in many areas of computing [1]. Motivated by applications in verification and synthesis of hardware/software designs [2]–[8], researchers have recently looked at the generalization of boolean formulas known as *dependency quantified boolean formulas* (DQBF).

While solving boolean SAT is "only" NP-complete, for DQBF the complexity jumps to NEXP-complete [9]. This makes solving DQBF quite a challenging research topic. Nevertheless there has been exciting progress. See, e.g., [10]–[18] and the references within, as well as solvers such as iDQ [19], dCAQE [20], HQS [21], [22] and DQBDD [23]. A natural question to ask is if we can use DQBF solvers to solve any NEXP-complete problems – similar to how SAT solvers are used to solve any NP-complete problems.

In this short paper we show how to reduce a wide variety of NEXP-complete problems to DQBF, especially the succinctly represented problems that recently have found applications in hardware/software engineering [24]–[26]. We present another proof for the NEXP-hardness of DQBF. We actually give two proofs. The first is by a very simple reduction from *succinct 3-colorability* [27]. The second is by utilizing the notion that we call *succinct projection*. It is the second one that we view more interesting since it gives us a general method to reduce any NEXP-complete problem to DQBF.

The main idea is quite standard: We encode the accepting runs of a non-deterministic Turing machine (with exponential run time) with boolean functions of polynomial arities. However, we observe that the input-output relation of these functions can actually be "described" by small circuits/formulas. Succinct projections are simply deterministic algorithms that construct these circuits efficiently. This simple observation is a deviation from the standard definition of NEXP, that a language in NEXP is a language with an exponentially long certificate.

Using succinct projections, we present reductions from various NEXP-complete problems such as (succinct) *Hamiltonian cycle*, *set packing* and *subset sum*. We believe our technique can be easily modified for many other natural problems. Note that the reduction in [9] gives little insight on how it can be used to obtain explicit reductions from concrete NEXP-complete problems.

We also present the reductions from well known NEXP-complete logics such as *the Bernays-Schönfinkel-Ramsey class*, *two-variable logic* (FO^2) and *the Löwenheim class* [28]–[32]. In fact we show that they are essentially equivalent to DQBF. Note that these are logics that have found applications in AI [33], databases [34] and automated reasoning [35], but lack implementable algorithms. Prior to our work, the only algorithm known for these logics is to "guess" a model (of exponential size) and then verify that it is indeed a model of the input formula.

We hope that the technique introduced in this short paper can lead to richer applications of DQBF solvers as well as a wide variety of benchmarks which in turn can lead to further development. It is also open whether the class NEXP has a *bona-fide* problem [27]. Our paper demonstrates that DQBF can be a good candidate – akin to how boolean SAT is the central problem in the class NP.

This paper is organized as follows. In Sect. II we review some definitions and terminology. In Sect. III we reprove the NEXP-completeness of solving DQBF. In Sect. IV and V we present concrete reductions from some NEXP-complete problems and logics to DQBF instances. The full version of this paper can be found in [36].

II. Preliminaries

Let $\Sigma = \{0, 1\}$. We usually use the symbol $\bar{a}, \bar{b}, \bar{c}$ (possibly indexed) to denote a string in Σ^* with $|\bar{a}|$ denoting the length of $\bar{a}$. We use $\bar{x}, \bar{y}, \bar{z}, \bar{u}, \bar{v}$ to denote vectors of boolean variables. The length of $\bar{x}$ is denoted by $|\bar{x}|$. We write $C(\bar{u})$ to denote a (boolean) circuit C with input gates $\bar{u}$. When the input gates are not relevant or clear from the context, we simply write C. For $\bar{a} \in \Sigma^{|\bar{u}|}$, $C(\bar{a})$ denotes the value of C when we assign the input gates $\bar{u}$ with $\bar{a}$. All logarithms have base 2.

https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_26

A *dependency quantified boolean formula* (DQBF) in prenex normal form is a formula of the form:

$$\Psi := \forall x_1 \cdots \forall x_n \ \exists y_1(\bar{z}_1) \cdots \exists y_m(\bar{z}_m) \quad \psi \qquad (1)$$

where each $\bar{z}_i$ is a vector of variables from $\{x_1, \ldots, x_n\}$ and ψ, called *the matrix*, is a quantifier-free boolean formula using variables $x_1, \ldots, x_n, y_1, \ldots, y_m$. The variables $x_1, \ldots, x_n$ are called *the universal variables*, $y_1, \ldots, y_m$ *the existential variables* and each $\bar{z}_i$ *the dependency set* of y_i.

A DQBF Ψ in the form (1) is *satisfiable*, if for every $1 \leqslant i \leqslant m$, there is a function $s_i : \Sigma^{|\bar{z}_i|} \to \Sigma$ such that by replacing each y_i with $s_i(\bar{z}_i)$, the formula ψ becomes a tautology. The function s_i is called the *Skolem function* for y_i. In this case, we also say that Ψ is satisfiable by the Skolem functions $s_1, \ldots, s_m$. The problem SAT(DQBF) is defined as: On input DQBF Ψ in the form (1), decide if it is satisfiable.

Since many NEXP-complete problems use circuits as the succinct representations of the inputs, we allow the matrix ψ to be in *circuit form*, i.e., ψ is given as a (boolean) circuit with input gates $x_1, \ldots, x_n, y_1, \ldots, y_m$. This does not effect the generality of our results, since every DQBF in circuit form can be converted to one in the standard formula form as stated in Proposition 1.

Proposition 1. *Every DQBF Ψ in the form of (1) in circuit form can be converted in polynomial time into an equisatisfiable DQBF formula Ψ' whose matrix is in DNF. Moreover, Ψ and Ψ' have the same existential variables (with the same dependency set).*

The proof is by standard Tseitin's transformation [37]. As an example, consider the following DQBF.

$$\forall x_1 \forall x_2 \ \exists y_1(x_1) \exists y_2(x_2) \ \neg\big(x_2 \vee (y_1 \wedge x_1 \wedge y_2)\big)$$

It is equisatisfiable with the following DQBF.

$$\forall x_1 \forall x_2 \ \forall u_1 \forall u_2 \forall u_3 \ \forall v_1 \forall v_2 \ \exists y_1(x_1) \exists y_2(x_2)$$
$$\left(\begin{array}{l} (v_1 \leftrightarrow y_1) \wedge (v_2 \leftrightarrow y_2) \wedge (u_1 \leftrightarrow v_1 \wedge x_1 \wedge v_2) \\ \wedge (u_2 \leftrightarrow x_2 \vee u_1) \wedge (u_3 \leftrightarrow \neg u_2) \end{array} \right) \rightarrow u_3$$

Intuitively, we use the extra variable v_1 to represent the value y_1, v_2 the value y_2, u_1 the value $y_1 \wedge x_1 \wedge y_2$, u_2 the value $x_2 \vee (y_1 \wedge x_1 \wedge y_2)$ and u_3 the value $\neg(x_2 \vee (y_1 \wedge x_1 \wedge y_2))$. Note that the matrix can be easily rewritten into DNF.

III. The NEXP-completeness of SAT(DQBF)

In this section we present two new proofs that SAT(DQBF) is NEXP-complete, originally proved in [9].

Theorem 2. [9] *SAT(DQBF) is* NEXP*-complete.*

Note that the membership is straightforward. So we will focus only on the hardness.

A. The first proof: Reduction from succinct 3-colorability

The reduction is from the problem *graph 3-colorability* where the input graphs are given in a succinct form [24]. A (boolean) circuit $C(\bar{u}, \bar{v})$, where $|\bar{u}| = |\bar{v}| = n$, represents a graph $G(C) = (V, E)$ where $V = \Sigma^n$ and $(\bar{a}, \bar{b}) \in E$ iff $C(\bar{a}, \bar{b}) = 1$. The problem *succinct 3-colorability* is defined as: On input circuit C, decide if $G(C)$ is 3-colorable. This problem is NEXP-complete [27].

The reduction to SAT(DQBF) is as follows. Let $C(\bar{u}, \bar{v})$ be the input circuit, where $|\bar{u}| = |\bar{v}| = n$. We represent a 3-coloring of $G(C)$ as a function $g : \Sigma^n \to \{01, 10, 11\}$ which can be encoded by the following DQBF.

$$\Psi := \forall \bar{x}_1 \forall \bar{x}_2 \ \exists y_1(\bar{x}_1) \exists y_2(\bar{x}_1) \ \exists y_3(\bar{x}_2) \exists y_4(\bar{x}_2)$$
$$\bar{x}_1 = \bar{x}_2 \ \rightarrow \ (y_1, y_2) = (y_3, y_4) \qquad (2)$$
$$\wedge \ (y_1, y_2) \neq (0, 0) \ \wedge \ (y_3, y_4) \neq (0, 0) \qquad (3)$$
$$\wedge \ C(\bar{x}_1, \bar{x}_2) = 1 \ \rightarrow \ (y_1, y_2) \neq (y_3, y_4) \qquad (4)$$

Intuitively, we use y_1, y_2 and y_3, y_4 to represent the first and the second bits of the image $g(\bar{x}_1)$ and $g(\bar{x}_2)$, respectively. Lines (2) and (3) state that (y_1, y_2) and (y_3, y_4) must represent the same function from Σ^n to Σ^2 and that their images do not include 00. Line (4) states that the colors of two adjacent vertices must be different. Thus, $G(C)$ is 3-colorable iff Ψ is satisfiable.

B. The second proof: Reduction via succinct projections

Our second proof uses the notion of *succinct projection*. We need some terminology. Let $C(\bar{u}_1, \bar{v}_1, \bar{u}_2, \bar{v}_2)$ be a circuit with input gates $\bar{u}_1, \bar{v}_1, \bar{u}_2, \bar{v}_2$ where $|\bar{u}_1| = |\bar{u}_2| = n$ and $|\bar{v}_1| = |\bar{v}_2| = m$. We say that a function $g : \Sigma^n \to \Sigma^m$ *agrees* with the circuit C, if $C(w_1, g(w_1), w_2, g(w_2)) = 1$, for every $w_1, w_2 \in \Sigma^n$. In this case, we also say that the circuit C *describes* the function g. In the following whenever we say that a function $g : \Sigma^n \to \Sigma^m$ agrees with $C(\bar{u}_1, \bar{v}_1, \bar{u}_2, \bar{v}_2)$, we implicitly assume that $n = |\bar{u}_1| = |\bar{u}_2|$ and $m = |\bar{v}_1| = |\bar{v}_2|$.

Definition 3. A *succinct projection* for a language L is a polynomial time deterministic algorithm $\mathcal{M}$ such that on input $w \in \Sigma^*$, $\mathcal{M}$ outputs a circuit C such that $w \in L$ iff there is a function g that agrees with C.

Intuitively, we can view the function g as the certificate for the membership of w in L and the circuit C as the succinct description of g. Since succinct projection runs in polynomial time, the output circuit can only have polynomially many gates. The following theorem is a new characterization of languages in NEXP.

Theorem 4. *A language $L \in$ NEXP iff it has a succinct projection.*

Proof. (if) Suppose that L has a succinct projection. Consider the following algorithm. On input w, first use the succinct projection to construct the circuit C. Then, guess a function g (of exponential size) and verify that it agrees with C. It is obvious that it runs in non-deterministic exponential time. That it is correct follows from the definition of succinct projection.

(only if) It is essentially the Cook-Levin reduction disguised in the form of function certificates. We only sketch it here. Let $L \in$ NEXP and M be a 1-tape NTM that accepts L in time $2^{p(n)}$ for some polynomial $p(n)$. For a word $w \in L$ of

length n, its accepting run can be represented as a function $g : \Sigma^{p(n)} \times \Sigma^{p(n)} \to \Sigma^{\ell}$, where $g(i, j)$ denotes the content of cell i in time j. The tuples in the codomain Σ^{ℓ} encode the states and the tape symbols of M. To verify that g represents an accepting run, it is sufficient to verify that for every $i_1, j_1, i_2, j_2 \in \Sigma^{p(n)}$, the tuple $(i_1, j_1, g(i_1, j_1), i_2, j_2, g(i_2, j_2))$ satisfies a certain property P which depends only on the input word w and the transitions of M. The desired succinct projection constructs in polynomial time a circuit C describing this property P. □

The second proof of the NEXP*-hardness of* SAT(DQBF)*:* Let $L \in$ NEXP. The polynomial time (Karp) reduction from L to SAT(DQBF) is described as Algorithm 1 below.

Algorithm 1: Reducing $L \in$ NEXP to SAT(DQBF)

Input: $w \in \Sigma^*$.

1: Run the succinct projection of L on w.

2: Let $C(\bar{x}_1, \bar{y}_1, \bar{x}_2, \bar{y}_2)$ be the output circuit where $|\bar{x}_1| = |\bar{x}_2| = n$, $|\bar{y}_1| = |\bar{y}_2| = m$, $\bar{y}_1 = (y_{1,1}, \ldots, y_{1,m})$ and $\bar{y}_2 = (y_{2,1}, \ldots, y_{2,m})$.

3: Output the following DQBF Ψ:

$$\forall \bar{x}_1 \forall \bar{x}_2 \, \exists y_{1,1}(\bar{x}_1) \cdots \exists y_{1,m}(\bar{x}_1) \, \exists y_{2,1}(\bar{x}_2) \cdots \exists y_{2,m}(\bar{x}_2) \; C(\bar{x}_1, \bar{y}_1, \bar{x}_2, \bar{y}_2) \; \wedge \; \big(\bar{x}_1 = \bar{x}_2 \to \bar{y}_1 = \bar{y}_2\big)$$

We show $w \in L$ iff Ψ is satisfiable. Suppose $w \in L$. Let $g : \Sigma^n \to \Sigma^m$ be a function that agrees with C. For each $1 \leqslant i \leqslant m$, define the Skolem function $s_i : \Sigma^n \to \Sigma$ where $s_i(\bar{a})$ is the i-th component of $g(\bar{a})$, for every $\bar{a} \in \Sigma^n$. It is routine to verify that Ψ is satisfiable with each s_i being the Skolem function for $y_{1,i}$ and $y_{2,i}$.

Conversely, suppose Ψ is satisfiable. Let $s_{j,i} : \Sigma^n \to \Sigma$ be the Skolem function for $y_{j,i}$, where $1 \leqslant j \leqslant 2$ and $1 \leqslant i \leqslant m$. Since $\bar{x}_1 = \bar{x}_2 \to \bar{y}_1 = \bar{y}_2$, the functions $s_{1,i}$ and $s_{2,i}$ must be the same, for every $1 \leqslant i \leqslant m$. Define $g : \Sigma^n \to \Sigma^m$ where $g(\bar{a}) = (s_1(\bar{a}), \ldots, s_{1,m}(\bar{a}))$ for every $\bar{a} \in \Sigma^n$. Since $C(\bar{a}_1, g(\bar{a}_1), \bar{a}_2, g(\bar{a}_2))$ is true for every $\bar{a}_1, \bar{a}_2$, the function g agrees with C. That is, there is a function that agrees with C. Hence, $w \in L$. This completes the second proof.

Remark 5. Observe that when Theorem 4 is applied to languages in NP, the accepting run of a non-deterministic Turing machine with polynomial run time $p(n)$ is represented as a function $g : \Sigma^{\log p(n)} \times \Sigma^{\log p(n)} \to \Sigma^{\ell}$ and the succinct projection outputs a circuit $C(\bar{x}_1, \bar{y}_1, \bar{x}_2, \bar{y}_2)$ where $|\bar{x}_1| = |\bar{x}_2| = \log p(n)$ and $|\bar{y}_1| = |\bar{y}_2| = \ell$. Thus, for $L \in$ NP, the DQBF output by Algorithm 1 has $4 \log p(n)$ universal variables and 2ℓ existential variables.

IV. Some concrete reductions

In this section we show how to utilize succinct projection to obtain the reductions from concrete NEXP-complete problems to SAT(DQBF). These are (succinct) *Hamiltonian cycle*, *set packing* and *subset sum* [27]. We use the notion of succinctness from [24] which has been explained in Sect. III-A. By Algorithm 1, it suffices to present only the succinct projections.

Some useful notations: For an integer $k \geqslant 1$, $[k]$ denotes the set $\{0, \ldots, k-1\}$. For $i \in [2^n]$, $\mathsf{bin}_n(i)$ is the binary representation of i in n bits. The number represented by $\bar{a} \in \Sigma^n$ is denoted by $\mathsf{num}(\bar{a})$. For $\bar{a}, \bar{b} \in \Sigma^n$, if $\mathsf{num}(\bar{a}) = \mathsf{num}(\bar{b}) + 1 \pmod{2^n}$, we say that $\bar{a}$ *is the successor of* $\bar{b}$, denoted by $\bar{a} = \bar{b} + 1$. Note that successor is applied only on two strings with the same length and the successor of 1^n is 0^n. It is not difficult to construct a circuit $C(\bar{x}, \bar{y})$ (in time polynomial in $|\bar{x}| + |\bar{y}|$) such that $C(\bar{a}, \bar{b}) = 1$ iff $\bar{a} = \bar{b} + 1$.

Reduction from succinct Hamiltonian cycle: Succinct Hamiltonian cycle is defined as follows. The input is a circuit $C(\bar{u}, \bar{v})$. The task is to decide if there is a Hamiltonian cycle in $G(C)$.

Let $C(\bar{u}, \bar{v})$ be the input circuit where $|\bar{u}| = |\bar{v}| = n$. We use a function $g : \Sigma^n \to \Sigma^n$ to represent a Hamiltonian cycle $(\bar{b}_0, \ldots, \bar{b}_{2^n - 1})$ where $g(\mathsf{bin}_n(i)) = \bar{b}_i$, for every $i \in [2^n]$. To correctly represent a Hamiltonian cycle, the following must hold for every $\bar{a}_1, \bar{a}_2 \in \Sigma^n$.

(H1) If $\bar{a}_1 \neq \bar{a}_2$, then $g(\bar{a}_1) \neq g(\bar{a}_2)$.

(H2) If $\bar{a}_2 = \bar{a}_1 + 1$, then $(g(\bar{a}_1), g(\bar{a}_2))$ is an edge in $G(C)$.

The succinct projection for succinct Hamiltonian cycle simply outputs the circuit that expresses (H1) and (H2), i.e., it outputs the following circuit $D(\bar{x}_1, \bar{y}_1, \bar{x}_2, \bar{y}_2)$ where $|\bar{x}_1| = |\bar{x}_2| = |\bar{y}_1| = |\bar{y}_2| = n$:

$$\big(\bar{x}_1 \neq \bar{x}_2 \to \bar{y}_1 \neq \bar{y}_2\big) \wedge \big(\bar{x}_2 = \bar{x}_1 + 1 \to C(\bar{y}_1, \bar{y}_2) = 1\big)$$

Obviously, a function $g : \Sigma^n \to \Sigma^n$ represents a hamiltonian cycle in $G(C)$ iff it agrees with D.

Reduction from succinct set packing: In the standard representation the problem *set packing* is defined as follows. The input is a collection $\mathcal{K}$ of finite sets $S_1, \ldots, S_\ell \subseteq \Sigma^m$ and an integer k. The task is to decide whether $\mathcal{K}$ contains k mutually disjoint sets. We assume each S_i has a "name" which is a string in $\Sigma^{\log \ell}$.

The succinct representation of the sets $S_1, \ldots, S_\ell$ is a circuit $C(\bar{u}, \bar{v})$ where $|\bar{u}| = m$ and $|\bar{v}| = \log \ell$. A string $\bar{a} \in \Sigma^m$ is in the set $S_{\bar{b}}$, if $C(\bar{a}, \bar{b}) = 1$. We denote by $\mathcal{K}(C)$ the collection of finite sets defined by the circuit C. The problem *succinct set packing* is defined analogously where the input is the circuit $C(\bar{u}, \bar{v})$ and an integer k (in binary).

We now describe its succinct projection. Let $C(\bar{u}, \bar{v})$ and k be the input where $|\bar{u}| = m$ and $|\bar{v}| = n$. We first assume that k is a power of 2. We represent k disjoint sets $S_1, \ldots, S_k$ in $\mathcal{K}(C)$ as a function $g : \Sigma^{\log k} \times \Sigma^m \to \Sigma^n$ where $g(\mathsf{bin}(i), \bar{a})$ is the name of the set S_i. Note that the string $\bar{a}$ is actually ignored in the definition of g.

For a function $g : \Sigma^{\log k} \times \Sigma^m \to \Sigma^n$ to correctly represent k disjoint sets, the following must hold for every $(\bar{a}_1, \bar{b}_1), (\bar{a}_2, \bar{b}_2) \in \Sigma^{\log k} \times \Sigma^m$.

(P1) If $\bar{a}_1 = \bar{a}_2$, then $g(\bar{a}_1, \bar{b}_1) = g(\bar{a}_2, \bar{b}_2)$. That is, the function g does not depend on $\bar{b}_1$ and $\bar{b}_2$.

(P2) If $\bar{a}_1 \neq \bar{a}_2$ and $\bar{b}_1 = \bar{b}_2$, then $C(\bar{b}_1, g(\bar{a}_1, \bar{b}_1)) = 0$ or $C(\bar{b}_1, g(\bar{a}_2, \bar{b}_2)) = 0$. That is, the element $\bar{b}_1$ is not in the sets whose names are $g(\bar{a}_1, \bar{b}_1)$ and $g(\bar{a}_2, \bar{b}_2)$.

It is routine to verify that g represents k disjoint sets iff (P1) and (P2) hold for every $(\bar{a}_1, \bar{b}_1), (\bar{a}_2, \bar{b}_2) \in \Sigma^{\log k} \times \Sigma^m$.

The succinct projection outputs the following circuit D that formalizes (P1) and (P2):

$$\big(\bar{x}_1 = \bar{x}_2 \to \bar{z}_1 = \bar{z}_2\big)$$
$$\wedge\big(\bar{x}_1 \neq \bar{x}_2 \wedge \bar{y}_1 = \bar{y}_2\big) \to \neg\big(C(\bar{y}_1, \bar{z}_1) = C(\bar{y}_1, \bar{z}_2) = 1\big)$$

If k is not a power of 2, we conjunct both atoms $\bar{x}_1 = \bar{x}_2$ and $\bar{x}_1 \neq \bar{x}_2$ with a circuit that tests whether the numbers represented by the bits $\bar{x}_1$ and $\bar{x}_2$ is an integer in $[k]$. Such a circuit can be easily constructed in polynomial time in $\lceil \log k \rceil$.

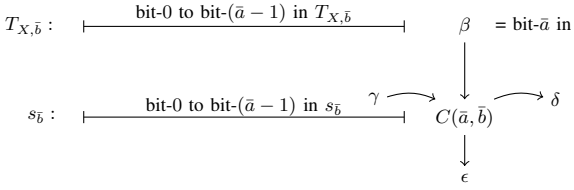
Reduction from succinct subset-sum: In the standard representation the instance of subset-sum is a list of positive integers $s_0, \ldots, s_{k-1}$ and t (all written in binary). The task is to decide if there is a subset $X \subseteq [k]$ such that $\sum_{i \in X} s_i = t$. Such X is called the subset-sum solution. The succinct representation is defined as two circuits $C_1(\bar{u}_1, \bar{v})$ and $C_2(\bar{u}_2)$, where $|\bar{u}_1| = \max_{i \in [k]} \log s_i$, $|\bar{v}| = \log k$ and $|\bar{u}_2| = \log t$. Circuit C_1 defines the numbers s_i's where $C_1(\bar{a}, \bar{b})$ is the i-th least significant bit of s_j, where $i = \mathsf{num}(\bar{a})$ and $j = \mathsf{num}(\bar{b})$. Circuit C_2 defines the number t where $C_2(\bar{a})$ is the i-th least significant bit of t, where $i = \mathsf{num}(\bar{a})$. The subset-sum instance represented by C_1 and C_2 is denoted by $\mathcal{N}(C_1, C_2)$. We will describe the succinct projection for succinct subset-sum.

Let $C_1(\bar{u}_1, \bar{v})$ and $C_2(\bar{u}_2)$ be the input where $|\bar{u}_1| = |\bar{u}_2| = n$ and $|\bar{v}| = m$. We need a few notations. Let $s_0, \ldots, s_{2^m-1}$ be the numbers represented by C_1 and t the number represented by C_2. For a set $X \subseteq [2^m]$, let $T_X = \sum_{i \in X} s_i$. For $0 \leqslant j \leqslant 2^m$, let $T_{X,j} = T_{X \cap [j]}$. Abusing the notation, for $\bar{b} \in \Sigma^m$, we write $s_{\bar{b}}$ and $T_{X,\bar{b}}$ to denote s_i and $T_{X,i}$, respectively, where $i = \mathsf{num}(\bar{b})$. For $\bar{a} \in \Sigma^n$, bit-$\bar{a}$ means bit-i where $i = \mathsf{num}(\bar{a})$.

We represent a set $X \subseteq [2^m]$ as a function $g : \Sigma^n \times \Sigma^m \to \Sigma^5$ where $g(\bar{a}, \bar{b}) = (\alpha, \beta, \gamma, \delta, \epsilon)$ such that:

- $\alpha = 1$ iff $s_{\bar{b}} \in X$.
- β is bit-$\bar{a}$ in $T_{X,\bar{b}}$.
- γ is the carry of adding $T_{X,\bar{b}}$ and $s_{\bar{b}}$ up to bit-$(\bar{a}-1)$.
- $\delta\epsilon = \beta + \gamma + C(\bar{a}, \bar{b})$, i.e., ϵ is the least significant bit of $\beta + \gamma + C(\bar{a}, \bar{b})$ and δ is the carry.

See the illustration below.



Intuitively, $g(\bar{a}, \bar{b})$ contains the information about the additions performed on bit-$\bar{a}$ in $s_{\bar{b}}$ (with respect to the set X). In particular, the bits of the number T_X are all contained in $g(\bar{a}, 1^m)$ for every $\bar{a} \in \Sigma^n$. These bits can then be compared to those in t by means of the circuit C_2.

Note that for a function $g : \Sigma^n \times \Sigma^m \to \Sigma^5$ to properly represent a number T_X, for some $X \subseteq [2^m]$, it suffices to check the values of g on "neighbouring" points in $\Sigma^n \times \Sigma^m$. More precisely, the following conditions must be satisfied for every $(\bar{a}_1, \bar{b}_1), (\bar{a}_2, \bar{b}_2) \in \Sigma^n \times \Sigma^m$, where $g(\bar{a}_1, \bar{b}_1) = (\alpha_1, \beta_1, \gamma_1, \delta_1, \epsilon_1)$ and $g(\bar{a}_2, \bar{b}_2) = (\alpha_2, \beta_2, \gamma_2, \delta_2, \epsilon_2)$.

(i) If $\bar{b}_1 = \bar{b}_2$, then $\alpha_1 = \alpha_2$. That is, the value α_1 depends only on the index of a number.
(ii) If $\alpha_1 = 0$, then $\gamma_1 = \delta_1 = 0$ and $\beta_1 = \epsilon_1$.
(iii) If $\alpha_1 = 1$, then $\gamma_1 + C(\bar{a}_1, \bar{b}_1) + \beta_1 = \delta_1\epsilon_1$.
(iv) If $\bar{a}_1 = 0^n$, then $\gamma_1 = 0$.
(v) If $\bar{a}_1 = 1^n$, then $\delta_1 = 0$.
(vi) If $\bar{b}_1 = 0^m$, then $\beta_1 = \gamma_1 = 0$.
(vii) If $\bar{b}_1 = 1^m$, then $\epsilon_1 = C_2(\bar{a}_1)$.
(viii) If $\alpha_1 = 1$ and $\bar{b}_1 = \bar{b}_2$ and $\bar{a}_2 = \bar{a}_1 + 1$, then $\delta_1 = \gamma_2$.
(ix) If $\alpha_1 = 1$ and $\bar{b}_2 = \bar{b}_1 + 1$ and $\bar{a}_2 = \bar{a}_1$, then $\epsilon_1 = \beta_2$.

Intuitively, (ii) and (iii) state that the values of $(\alpha_1, \beta_1, \gamma_1, \delta_1, \epsilon_1)$ must have their intended meaning, i.e., when $\alpha_1 = 0$, no addition is performed and when $\alpha_1 = 1$, the addition $\gamma_1 + C(\bar{a}_1, \bar{b}_1) + \beta_1$ is performed and the result is $\delta_1\epsilon_1$. (iv) states that there is no carry from the previous bit when considering the least significant bit. (v) states that there shouldn't be any carry after adding the most significant bit (if we want T_X equals t). (vi) states that $T_{X,0}$ must be zero. (vii) states that bit-$\bar{a}$ in T_X must equal to bit-$\bar{a}$ in t. Finally, (viii) and (ix) state that when $(\bar{a}_1, \bar{b}_1)$ and $(\bar{a}_2, \bar{b}_2)$ are neighbors, the bits $\beta_1, \gamma_1, \delta_1, \epsilon_1$ and $\beta_2, \gamma_2, \delta_2, \epsilon_2$ must obey their intended meaning.

Obviously, if g satisfies (i)–(ix), then it represents a set X such that $T_X = t$. Conversely, if there is a set X such that $T_X = t$, then there is a function g that satisfies (i)–(ix). It is not difficult to design a succinct projection that constructs a circuit D that describes functions that satisfy (i)-(ix).

V. Reductions from other NEXP-complete logics

In this section we will consider the following fragments of relational first-order logic (with the equality predicate):

- The *Bernays-Schönfinkel-Ramsey* (BSR) class: The class of sentences of the form:
$$\Psi_1 := \exists x_1 \cdots \exists x_m \; \forall y_1 \cdots \forall y_n \; \psi$$
where ψ is a quantifier-free formula.
- The *two-variable logic* (FO^2): The class of sentences using only two variables x and y.
The classic result by Scott [38] states that every FO^2 sentence can be transformed in linear time into an equisatisfiable FO^2 sentence of the form:
$$\Psi_2 := \forall x \forall y \; \alpha(x, y) \; \wedge \bigwedge_{i=1}^{m} \forall x \exists y \beta_i(x, y)$$
for some $m \geqslant 1$, where $\alpha(x, y)$ and each $\beta_i(x, y)$ are quantifier free formulas.
- The *Löwenheim/monadic* class: The class of sentences using only unary predicate symbols. Sentences in this class are also known as monadic sentences.

Let $\mathsf{SAT}(\mathrm{BSR})$, $\mathsf{SAT}(\mathrm{Mon})$ and $\mathsf{SAT}(\mathsf{FO}^2)$ denote their corresponding satisfiability problems. It is well known that all of them are NEXP-complete [28]–[32]. The upper bound is usually established by the so called *Exponential Size Model* (ESM) property stated as follows.

- If the BSR sentence Ψ_1 is satisfiable, then it is satisfiable by a model with size at most $m+1$ [31, Prop. 6.2.17].
- If the FO^2 sentence Ψ_2 is satisfiable, then it is satisfiable by a model with size $m2^n$, where n is the number of unary predicates used [30].
- If a Löwenheim sentence is satisfiable, then it is satisfiable by a model with size at most $r2^n$, where r is the quantifier rank and n is the number of unary predicates [31, Prop. 6.2.1].

The main idea of the reduction to $\mathsf{SAT}(\mathrm{DQBF})$ is quite simple. We will represent the domain of a model with size at most N as a subset of Σ^t, where $t = \log N$ and use a function $f_0 : \Sigma^t \to \Sigma$ as the indicator whether an element is in the domain. Every predicate in the input formula can be represented as a function $f : \Sigma^{kt} \to \Sigma$ where k is the arity of the predicate. All these functions can then be encoded appropriately as existential variables in DQBF. Note that the universal FO quantifier $\forall x \cdots$ can be encoded as $\forall \bar{u}\ f_0(\bar{u}) \to \cdots$. The existential FO quantifier can first be Skolemized and then encoded as existential variables in DQBF.

The rest of this section is organized as follows. For technical convenience, we first introduce the logic Existential Second-order Quantified Boolean Formula ($\exists$SOQBF) – an alternative, but equivalent formalism of DQBF. The only difference between $\exists$SOQBF and DQBF is the syntax in declaring the function symbol. Then, we consider the problem that we call *Bounded FO satisfiability*, denoted by $\mathsf{Bnd\text{-}SAT}(\mathsf{FO})$, which subsumes all $\mathsf{SAT}(\mathrm{BSR})$, $\mathsf{SAT}(\mathsf{FO}^2)$ and $\mathsf{SAT}(\mathrm{Mon})$ and show how to reduce it to $\mathsf{SAT}(\exists\mathrm{SOQBF})$.

The logic $\exists$SOQBF: The class $\exists$SOQBF is the extension of quantified boolean formulas (QBF) with existential second-order quantifiers, i.e., formulas of the form:

$$\Psi := \exists f_1 \exists f_2 \cdots \exists f_p\ Q_1 v_1 \cdots\ Q_n v_n \quad \psi$$

where each $Q_i \in \{\forall, \exists\}$ and each f_i is a boolean function symbol associated with a fixed arity $\mathrm{ar}(f_i)$. The formula ψ is a boolean formula using the variables v_i's and $f(\bar{z})$'s, where $f \in \{f_1, \ldots, f_p\}$, $|\bar{z}| = \mathrm{ar}(f)$ and $\bar{z} \subseteq \{v_1, \ldots, v_q\}$. We call each $f(\bar{z})$ in ψ a *function variable*.

The semantics of Ψ is defined naturally. We say that Ψ is satisfiable, if there is an interpretation $F_i : \Sigma^{\mathrm{ar}(f_i)} \to \Sigma$ for each f_i such that $Q_1 v_1 \cdots\ Q_n v_n\ \psi$ is a true QBF. In this case we say that $F_1, \ldots, F_p$ *make* Ψ *true*. It is not difficult to see that DQBF and $\exists$SOQBF can be transformed to each other in linear time while preserving satisfiability.

Bounded FO satisfiability ($\mathsf{Bnd\text{-}SAT}(\mathsf{FO})$): The problem $\mathsf{Bnd\text{-}SAT}(\mathsf{FO})$ is defined as: On input relational FO sentence φ and a positive integer N (in binary), decide if φ has a model with cardinality at most N. It is a folklore that $\mathsf{Bnd\text{-}SAT}(\mathsf{FO})$ is NEXP-complete. Note that due to the ESM property, $\mathsf{Bnd\text{-}SAT}(\mathsf{FO})$ trivially subsumes all $\mathsf{SAT}(\mathrm{BSR})$, $\mathsf{SAT}(\mathsf{FO}^2)$ and $\mathsf{SAT}(\mathrm{Mon})$.

Reduction from $\mathsf{Bnd\text{-}SAT}(\mathsf{FO})$ to $\mathsf{SAT}(\exists\mathrm{SOQBF})$: Let φ and N be the input to $\mathsf{Bnd\text{-}SAT}(\mathsf{FO})$. We may assume that φ is in the Prenex normal form: $\varphi := Q_1 x_1 \cdots Q_n x_n\ \psi$, where each $Q_i \in \{\forall, \exists\}$ and ψ is quantifier-free formula. Adding redundant quantifier, if necessary, we assume that Q_1 is $\forall$. Then, we Skolemize each existential quantifier as follows. Let i be the minimal index where $Q_i = \exists$. We rewrite φ into:

$$\varphi' := \forall x_1 \cdots \forall x_{i-1}\ Q_{i+1} x_{i+1} \cdots Q_n x_n\ \forall z \\ z = g(x_1, \ldots, x_{i-1})\ \to\ \psi'$$

where z is a fresh variable, g is the Skolem function representing the existentially quantified variable x_i and ψ' is obtained from ψ by replacing every occurrence of x_i with z. Hence, we may assume that the input sentence φ is of form:

$$\varphi := \forall x_1 \cdots \forall x_n\ \psi \tag{5}$$

where ψ is quantifier-free formula where every (Skolem) function symbol $g(x_1, \ldots, x_{i-1})$ only occur in the equality predicate $z = g(x_1, \ldots, x_{i-1})$ and z is one of $x_i, \ldots, x_n$.

Let $g_1, \ldots, g_k$ be the Skolem function symbols in ψ and $P_1, \ldots, P_\ell$ be the predicates in ψ. Let $\mathrm{ar}(g_i)$ and $\mathrm{ar}(P_i)$ denote the arity of g_i and P_i. Let $t = \lceil \log N \rceil$. Construct the following $\exists$SOQBF formula:

$$\Phi := \exists f_0\ \exists f_{1,1} \cdots \exists f_{1,t} \cdots \exists f_{k,1} \cdots \exists f_{k,t}\ \exists f_{P_1} \cdots \exists f_{P_\ell} \\ \forall \bar{u}_1 \cdots \forall \bar{u}_n\ \left(\begin{array}{l} \bar{u}_1 = 0^t \to f_0(\bar{u}_1) \\ \wedge\ \bigwedge_{i=1}^{n} f_0(\bar{u}_i)\ \to\ \Psi \end{array} \right) \tag{6}$$

where:

- The arity of f_0 is t.
- For every $1 \leqslant i \leqslant k$, the arity of $f_{1,1}, \ldots, f_{1,t}$ is $t \cdot \mathrm{ar}(g_i)$.
- For every $1 \leqslant i \leqslant \ell$, the arity of $f_{P_1}, \ldots, f_{P_\ell}$ is $t \cdot \mathrm{ar}(P_i)$.
- For every $1 \leqslant i \leqslant n$, $|\bar{u}_i| = t$.

The formula Ψ is obtained from ψ as follows.

- Each predicate $P_i(x_{j_1}, \ldots, x_{j_m})$ is replaced with $f_{P_i}(\bar{u}_{j_1}, \ldots, \bar{u}_{j_m})$.
- Each predicate $x_j = g_i(x_{j_1}, \ldots, x_{j_m})$ is replaced with $\bar{u}_j = (f_{i,1}(\bar{u}_{j_1}, \ldots, \bar{u}_{j_m}), \ldots, f_{i,t}(\bar{u}_{j_1}, \ldots, \bar{u}_{j_m}))$
- Each predicate $x_j = x_i$ is replaced with $\bar{u}_j = \bar{u}_i$.

Intuitively, we use f_0 as the indicator to determine whether a string in Σ^t is an element in the model. To ensure that the model is not empty, we insist that 0^t belongs to the model, hence, the formula $\bar{u}_1 = 0^t \to f_0(\bar{u}_1)$. We use the vector of variables $\bar{u}_i$ to represent x_i. For every $1 \leqslant i \leqslant k$, the functions $f_{i,1}, \ldots, f_{i,t}$ represent the bit representation of $g_i(x_{j_1}, \ldots, x_{j_m})$. Finally, for every $1 \leqslant i \leqslant \ell$, the function f_{P_i} represents the predicate P_i. Note the part $\bigwedge_{i=1}^{n} f_0(\bar{u}_i) \to \Psi$ which means we require Ψ to hold only on the vectors $\bar{u}_1, \ldots, \bar{u}_n$ that "passes" the function f_0, i.e., they are elements of the model. It is routine to verify that the formula φ in Eq. (5) is satisfiable by a model with cardinality at most N iff the $\exists$SOQBF formula Φ in Eq. (6) is satisfiable.

Acknowledgement

We are very grateful to Jie-Hong Roland Jiang for many fruitful discussions on the preliminary drafts of this work. We also thank the anonymous reviewers for their constructive comments. We acknowledge the generous financial support of Taiwan National Science and Technology Council under grant no. 109-2221-E-002-143-MY3.

REFERENCES

[1] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability*. IOS Press, 2009.

[2] J. R. Jiang, "Quantifier elimination via functional composition," in *CAV*, 2009.

[3] V. Balabanov and J. R. Jiang, "Reducing satisfiability and reachability to DQBF," in *Talk given at QBF*, 2015.

[4] C. Scholl and B. Becker, "Checking equivalence for partial implementations," in *DAC*, 2001.

[5] K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, and B. Becker, "Equivalence checking of partial designs using dependency quantified boolean formulae," in *ICCD*, 2013.

[6] R. Bloem, R. Könighofer, and M. Seidl, "SAT-based synthesis methods for safety specs," in *VMCAI*, 2014.

[7] K. Chatterjee, T. Henzinger, J. Otop, and A. Pavlogiannis, "Distributed synthesis for LTL fragments," in *FMCAD*, 2013.

[8] A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai, "Robust boolean reasoning for equivalence checking and functional property verification," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 21, no. 12, pp. 1377–1394, 2002.

[9] G. Peterson and J. Reif, "Multiple-person alternation," in *FOCS*, 1979.

[10] V. Balabanov, H. K. Chiang, and J. R. Jiang, "Henkin quantifiers and boolean formulae: A certification perspective of DQBF," *Theor. Comput. Sci.*, vol. 523, pp. 86–100, 2014.

[11] A. Fröhlich, G. Kovásznai, and A. Biere, "A DPLL algorithm for solving DQBF," in *POS-12, Third Pragmatics of SAT workshop*, 2012.

[12] A. Ge-Ernst, C. Scholl, and R. Wimmer, "Localizing quantifiers for DQBF," in *FMCAD*, 2019.

[13] O. Kullmann and A. Shukla, "Autarkies for DQCNF," in *FMCAD*, 2019.

[14] R. Wimmer, C. Scholl, and B. Becker, "The (D)QBF preprocessor hqspre - underlying theory and its implementation," *J. Satisf. Boolean Model. Comput.*, vol. 11, no. 1, pp. 3–52, 2019.

[15] K. Wimmer, R. Wimmer, C. Scholl, and B. Becker, "Skolem functions for DQBF," in *ATVA*, 2016.

[16] R. Wimmer, S. Reimer, P. Marin, and B. Becker, "HQSpre – an effective preprocessor for QBF and DQBF," in *TACAS*, 2017.

[17] G. Kovásznai, "What is the state-of-the-art in DQBF solving," in *Join Conference on Mathematics and Computer Science*, 2016.

[18] C. Scholl and R. Wimmer, "Dependency quantified boolean formulas: An overview of solution methods and applications - extended abstract," in *SAT*, 2018.

[19] A. Fröhlich, G. Kovásznai, A. Biere, and H. Veith, "iDQ: Instantiation-based DQBF solving," in *POS-14, Fifth Pragmatics of SAT workshop*, 2014.

[20] L. Tentrup and M. Rabe, "Clausal abstraction for DQBF," in *SAT*, 2019.

[21] K. Gitina, R. Wimmer, S. Reimer, M. Sauer, C. Scholl, and B. Becker, "Solving DQBF through quantifier elimination," in *DATE*, 2015.

[22] R. Wimmer, A. Karrenbauer, R. Becker, C. Scholl, and B. Becker, "From DQBF to QBF by dependency elimination," in *SAT*, 2017.

[23] J. Síc and J. Strejcek, "DQBDD: an efficient bdd-based DQBF solver," in *SAT*, 2021.

[24] H. Galperin and A. Wigderson, "Succinct representations of graphs," *Inf. Control.*, vol. 56, no. 3, pp. 183–198, 1983.

[25] D. Kini, U. Mathur, and M. Viswanathan, "Data race detection on compressed traces," in *ESEC/SIGSOFT FSE*, 2018.

[26] A. Pavlogiannis, N. Schaumberger, U. Schmid, and K. Chatterjee, "Precedence-aware automated competitive analysis of real-time scheduling," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3981–3992, 2020.

[27] C. Papadimitriou and M. Yannakakis, "A note on succinct representations of graphs," *Inf. Control.*, vol. 71, no. 3, pp. 181–185, 1986.

[28] H. Lewis, "Complexity results for classes of quantificational formulas," *J. Comput. Syst. Sci.*, vol. 21, no. 3, pp. 317–353, 1980.

[29] M. Fürer, "The computational complexity of the unconstrained limited domino problem (with implications for logical decision problems)," in *Logic and Machines: Decision Problems and Complexity*, 1983, pp. 312–319.

[30] E. Grädel, P. Kolaitis, and M. Vardi, "On the decision problem for two-variable first-order logic," *Bull. Symbolic Logic*, vol. 3, no. 1, pp. 53–69, 3 1997.

[31] E. Börger, E. Grädel, and Y. Gurevich, *The Classical Decision Problem*. Springer, 1997.

[32] T. Lin, C. Lu, and T. Tan, "Towards a more efficient approach for the satisfiability of two-variable logic," in *LICS*, 2021.

[33] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[34] S. Itzhaky, T. Kotek, N. Rinetzky, M. Sagiv, O. Tamir, H. Veith, and F. Zuleger, "On the automated verification of web applications with embedded SQL," in *ICDT*, 2017, pp. 16:1–16:18.

[35] J. Robinson and A. Voronkov, Eds., *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[36] F.-H. Chen, S.-C. Huang, Y.-C. Lu, and T. Tan, "Reducing NEXP-complete problems to DQBF," *CoRR*, vol. abs/2208.06014, 2022. [Online]. Available: http://arxiv.org/abs/2208.06014

[37] G. Tseitin, "On the complexity of derivation in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logic, Part II*, 1968.

[38] D. Scott, "A decision method for validity of sentences in two variables," *The Journal of Symbolic Logic*, p. 377, 1962.