Integer Circuit Evaluation is PSPACE-complete

Ke Yang

Computer Science Department, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, USA

E-mail: yangke@cmu.edu

Key Words: PSPACE, Integer Circuit, Chinese Remainder Theorem

Wagner [10] introduced the Integer Circuit Evaluation problem. Informally, the problem concerns a circuit that takes singleton sets, each containing one integer, and combines them using three types of set operations: $A \cup B$, $A \otimes B = \{a \cdot b \mid a \in A,\ b \in B\}$, and $A \oplus B = \{a + b \mid a \in A,\ b \in B\}$. The problem is to determine whether the set output by the circuit contains a particular integer. In this paper we show that the Integer Circuit problem is PSPACE-complete, resolving an open problem posed by McKenzie, Vollmer and Wagner [7].

1. INTRODUCTION

The Integer Circuit Evaluation problem was first introduced by Wagner [10] (though in a different representation). Roughly speaking, an Integer Circuit (IC) takes singleton sets, each containing one integer, as input, and has three types of set operations as gates: the union gate — $A \cup B$; the pair-wise multiplication gate — $A \otimes B = \{a \cdot b \mid a \in A,\ b \in B\}$; the pair-wise addition gate — $A \oplus B = \{a + b \mid a \in A,\ b \in B\}$. The Integer Circuit Evaluation problem (ICE) is given an integer X, a circuit and its inputs, to determine whether or not X is contained in the set output by the circuit. Wagner proved that this problem is in PSPACE [10]. In this paper we demonstrate a polynomial-time algorithm that reduces the Quantified Boolean Formula (QBF) problem to the Integer Circuit Evaluation problem, thus proving the Integer Circuit Evaluation problem is PSPACE-complete. This result resolves one of the open problems posed in [7].

Circuit computation is thought to be strictly more powerful than formula computation. For example, polynomial-size boolean formula evaluation is in NC^1, while polynomial-size boolean circuit evaluation is P-complete. As early as 1973, Stockmeyer and Meyer studied the complexity of integer expressions [9], which can be rephrased as the "formula version" of the Integer Circuit Evaluation problem. They proved that $\{\cup, +\}$-formula problem (where one only has union and pair-wise addition operations) is NP-complete. Their result easily extends to $\{\cup, \times, +\}$-formula problem (where one allows pair-wise multiplication operations). Furthermore, they observed that, if one allows negation, namely, operations that negates all elements in a set, the problem is already PSPACE-complete. But they didn't consider integer circuit evaluation problems.

The complexity of circuit evaluation has always been a point of interests and different models of circuit evaluation are studied. For instance, Beaudry, McKenzie, Péladeau, and Thérien proved that circuit evaluation over nonsolvable monoids is P-complete, and circuit evaluation over solvable monoids can be evaluated in $DET \subseteq NC^2$ [3]. Allender et al. [1] discussed depth reduction for commutative and non-commutative arithmetic circuits. They proved that in the commutative setting, uniform semi-bounded arithmetic circuits of logarithmic depth are as powerful as uniform arithmetic circuits of polynomial degree (unbounded depth). They also proved that in the non-commutative case, over the algebra $(\Sigma^*, \max, \mathrm{concat})$, arithmetic circuits of polynomial size and polynomial degree can be reduced to $O(\log^2 n)$ depth.

Wagner in his paper [10] defined "hierarchical descriptions", which are equivalent to Integer Circuits without pair-wise multiplication gates. He proved that the membership problem for hierarchical descriptions is in PSPACE, and left open if this problem is PSPACE-complete.

Recently, McKenzie, Vollmer and Wagner in [7] defined "polynomial replacement systems", in which the Integer Circuits arise as a special case. One of the open problems in their paper is the PSPACE-completeness of the Integer Circuit Evaluation problem.

In the rest of this paper, section 2 contains the definitions and formal notations; section 3 contains the reduction from QBF to the Integer Circuit Problem. The reduction consists of 3 parts: pre-processing the QBF, core reduction to remove the quantifiers, and the post-processing to extract the results. Section 4 is the summary.

2. DEFINITIONS AND NOTATIONS

We give some definitions and notations to be used in the rest of the paper.

We use $\mathbb{N}$ to denote the set of all positive integers and $\mathbb{Z}$ to denote the set of all integers. When there is no danger of confusion, we use 1 and -1 to represent TRUE and FALSE, respectively. For an integer set A, we use $||A||$ to denote the maximum absolute value of elements in A, i.e., $||A|| = \max\{|x| \,:\, x \in A\}$. For a positive integer X, we use $[X]$ to denote the set $\{1, 2, ..., X\}$.

2.1. Quantified Boolean Formula

Quantified Boolean Formula (QBF) is a well-known problem. We consider the following **standard form** of QBF:

$$F = Q_1 x_1 Q_2 x_2 ... Q_n x_n \ \phi(x_1, x_2, ..., x_n)$$

where $Q_i = \forall$ or $\exists$, $i = 1, 2, ..., n$, and $\phi(x_1, ..., x_n)$ is a DNF of variables $x_1, x_2, ..., x_n$. We call this type of QBF "standard form QBF", since it has special properties that:

1. all variables are quantified, namely, there are no free variables.
2. ϕ is a DNF, i.e., ϕ is the OR of several clauses, while each clause is an AND of literals.

The QBF problem is: given a QBF F in standard form, decide if the F is TRUE or FALSE. It is well-known that the standard QBF is a PSPACE-complete problem[9].

2.2. Integer Circuits

We give the definition of Integer Circuits.

DEFINITION 2.1. [Integer Circuit] An **Integer Circuit** (IC) is a directed acyclic graph with n nodes of in-degree zero, called inputs and labeled $X_1, X_2, ..., X_n$, and with all other nodes of in-degree two. The nodes of in-degree two each has a label from $\{\cup, \oplus, \otimes\}$, and one of these nodes is specified as the output gate.

The semantics of the IC are as follows. Each input gate is assigned a singleton set of integer and an internal gate receiving sets A and B along its two input wires computes the following:

1. A $\cup$ gate (or *union gate*) computes $A \cup B$.
2. A $\oplus$ gate (or *addition gate*) computes $\{a + b \mid a \in A, b \in B\}$.
3. A $\otimes$ gate (or *multiplication gate*) computes $\{a \cdot b \mid a \in A, b \in B\}$.

The output of the circuit is the set computed by the output gate.

The **Integer Circuit Evaluation** problem (ICE) is defined as a set of tuples in the form of $\langle C, X, a_1, a_2, ..., a_n\rangle$, where C is an Integer Circuit and X, $a_1, a_2, ..., a_n$ are positive integers. The tuple is in the ICE if X belongs to the set computed by the circuit C when the singleton set $\{a_i\}$ is assigned to the input gate X_i for each $i = 1, 2, ..., n$.

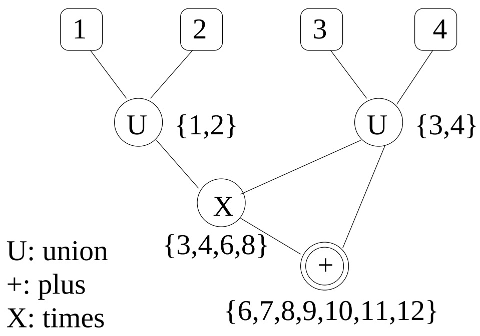

FIG. 1. An Integer Circuit

Figure 1 gives an example of an Integer Circuit.

Wagner proved that ICE is in PSPACE [10]. In this paper, we show it is actually PSPACE-complete. We prove this by showing a reduction from QBF to the ICE.

2.3. Integer Vector

We define integer vectors and the operations on them.

DEFINITION 2.2. [Integer Vector] An **n-dimensional Integer Vector** $\mathbf{v}$ is written as $\mathbf{v} = \langle v^1, v^2, ..., v^n \rangle$, where $v^i \in \mathbb{Z}$, for $i = 1, 2, ...n$. We denote the set of all n-dimensional integer vectors by $\mathbb{Z}^n$.

DEFINITION 2.3. [L_∞ Norm] We define the L_∞ **Norm** of an integer vector $\mathbf{v}$ to be:

$$||\mathbf{v}|| = \max \{|v^i| \ : i = 1, 2, ..., n\}$$

We define the L_∞ **Norm** of a finite set S of integer vectors to be:

$$||S|| = \max \{||\mathbf{v}|| \ : \ \mathbf{v} \in S\}$$

Notice this is consistent with the case when $n = 1$: now S is a set of integers and $||S||$ is exactly the maximum absolute value of the elements in S.

DEFINITION 2.4. [Neighbors] Two n-dimensional integer vectors $\mathbf{u}$ and $\mathbf{v}$ are **Neighbors in the k-th entry**, iff:

1. $u^k = -v^k$.
2. $u^i = v^i$, for all $i \neq k, 1 \leq i \leq n$.

We write $[\mathbf{v}]_k$ for the (unique) neighbor of $\mathbf{v}$ in the k-th entry, so that $\mathbf{u} = [\mathbf{v}]_k$ iff $\mathbf{v} = [\mathbf{u}]_k$.

DEFINITION 2.5. [Scalar Multiple] Given an integer a and an n-dimensional integer vector $\mathbf{v}$, their **Scalar Multiple** is defined as

$$a \cdot \mathbf{v} = \langle a \cdot v^1, a \cdot v^2, ..., a \cdot v^n \rangle$$

We denote $-1 \cdot \mathbf{v}$ by $-\mathbf{v}$.

Definition 2.6. [Addition and Multiplication] For n-dimensional integer vectors $\mathbf{u}$ and $\mathbf{v}$, we define their **Addition** and **Multiplication** as follows:

$$\begin{aligned}\mathbf{u}+\mathbf{v} &= \langle u^1+v^1, u^2+v^2, ..., u^n+v^n\rangle \\ \mathbf{u}\cdot\mathbf{v} &= \langle u^1\cdot v^1, u^2\cdot v^2, ..., u^n\cdot v^n\rangle\end{aligned}$$

Notice that the multiplication defined here is entry-wise multiplication, rather than the inner product.

Now we define some useful constant vectors.

Definition 2.7. We define the following constants:

- $\mathbf{1}^n = \langle 1, 1, .., 1\rangle$.
- $\mathbf{e}_k^n = \langle e_1, e_2, .., e_n\rangle$, where $e_k = 1$, and $e_i = 0$ for all $i \neq k, 1 \leq i \leq n$.
- $\mathbf{1}_k^n = \langle a_1, a_2, .., a_n\rangle$, where $a_k = -1$, and $a_i = 1$ for all $i \neq k, 1 \leq i \leq n$.

For example, $\mathbf{e}_2^5 = \langle 0, 1, 0, 0, 0\rangle$ and $\mathbf{1}_3^4 = \langle 1, 1, -1, 1\rangle$.

When the value of n is clear from the context and there is no danger of ambiguity, we normally eliminate the n — e.g., we write e_k rather than e_k^n.

2.4. Vector Integer Circuit

We will look at another kind of circuits — *Vector Integer Circuits* (VIC), which is closely related to the IC. Actually the only difference between VIC and IC is that in VIC, the operations are over sets of *integer vectors*, rather than sets of *positive integers* in IC.

Definition 2.8. [Vector Integer Circuit]

A **Vector Integer Circuit** is a directed acyclic graph with n nodes of in-degree zero, called inputs and labeled $X_1, X_2, ..., X_n$, and with all other nodes of in-degree two. The nodes of in-degree two each has a label from $\{\cup, \oplus, \otimes\}$, and one of these nodes is specified as the output gate.

The semantics of the VIC are as follows. Each input gate is assigned a singleton set of integer vector and an internal gate receiving sets A and B along its two input wires computes the following:

1. A $\cup$ gate (or *union gate*) computes $A \cup B$.
2. A $\oplus$ gate (or *addition gate*) computes $\{\mathbf{a}+\mathbf{b} \mid \mathbf{a} \in A, \mathbf{b} \in B\}$.
3. A $\otimes$ gate (or *multiplication gate*) computes $\{\mathbf{a}\cdot\mathbf{b} \mid \mathbf{a} \in A, \mathbf{b} \in B\}$.

The output of the circuit is the set computed by the output gate.

The **Vector Integer Circuit Evaluation** problem (VICE) is defined as a set of tuples in the form of $\langle C, \mathbf{X}, \mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n\rangle$, where C is a Vector Integer Circuit and $\mathbf{X}$, $\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n$ are integer vectors. The tuple is in the VICE if $\mathbf{X}$ belongs to the set computed by the circuit C when the singleton set $\{\mathbf{a}_i\}$ is assigned to the input gate X_i for each $i = 1, 2, ..., n$.

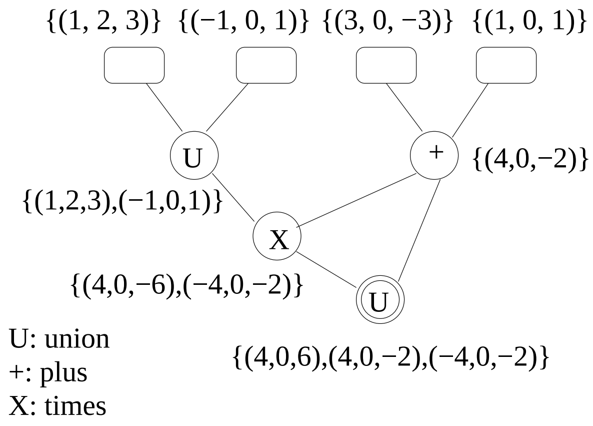

FIG. 2. An Vector Integer Circuit

Figure 2 gives an example of a Vector Integer Circuit.

A Vector Integer Circuit has the same structure as an Integer Circuit, but the elements under operation are sets of integer vectors, rather than sets of positive integers, and they have their own version of addition and multiplication. Notice that an IC is a special case of a VIC since an integer can be regarded as a 1-dimensional vector. Actually, as we will use Chinese Remainder Theorem to prove later, they are actually equivalent.

The reason we introduce the VICE is that we will first reduce the QBF to VICE, and then we further reduce VICE to ICE, thus showing ICE is PSPACE-hard.

3. REDUCING QBF TO ICE

In this section we show how to reduce QBF to VICE. Our reduction contains three parts: pre-processing the DNF formula to build a truth table; core reduction to remove the quantifiers; and post-processing to extract the result. The first two parts reduce QBF to VICE, and the third part reduces VICE to ICE.

3.1. Part 1: Pre-processing the DNF Formula

We begin with some definitions.

It is intuitively clear that we can use integer vectors to represent truth assignments, and we give a formal definition here:

DEFINITION 3.1. [Truth Assignment Vector] An n-dimensional integer vector $\mathbf{v}$ is a **Truth Assignment Vector**, if all its entries are 1 or -1, i.e., $v^i = \pm 1$, for $i = 1, 2, ...n$.

Claim. There are 2^n truth assignment vectors in $\mathbb{Z}^n$.

DEFINITION 3.2. [Satisfying Vector] For an n-dimensional truth assignment vector $\mathbf{v} = \langle v_1, v_2, ..., v_n \rangle$ and a boolean formula ϕ of n variables $x_1, ..., x_n$, we say **v satisfies** ϕ, if $\phi(v^1, v^2, .., v^n)$ is TRUE.

Notice that we can add "dummy variables" to a boolean formula, and thus a formula of n variables can also be regarded as a formula of m variables for $m > n$. For example, we can view formula $(x_1 \wedge x_2) \vee \neg x_3$ as a formula of variables x_1, x_2, x_3, x_4, and so vector $\langle 1, 1, 1, -1 \rangle$ is a satisfying vector for this formula. Moreover, we can also view the constants TRUE and FALSE as boolean formulas of n variables. Therefore any n-dimensional truth assignment vector satisfies TRUE and no truth assignment vector satisfies FALSE.

From now on, unless otherwise stated, we assume we are working with a universe of n variables: $x_1, x_2, ..., x_n$, and all boolean formulas are regarded as formulas of n variables. When there is no danger of confusion, we often remove n from our statements.

DEFINITION 3.3. [Truth Table] For a boolean formula ϕ, we define its **Truth Table** to be the set of all the truth assignments that satisfy ϕ:

$$T(\phi) = \{\mathbf{v} \mid \mathbf{v}\ \texttt{satisfies}\ \phi,\ \mathbf{v} \in \mathbb{Z}^n\}$$

In particular, the truth table of TRUE is the set of all n-dimensional truth assignment vectors and the truth table of FALSE is the $\emptyset$.

Our goal in this subsection is to construct a truth table for a DNF formula, using a Vector Integer Circuit.

Notice if ϕ is a DNF formula, then it can be written as

$$\phi = C_1 \vee C_2 \vee \cdots \vee C_m$$

where each C_i is a conjunctive clause. So we know

$$T(\phi) = \bigcup_{i=1}^{m} T(C_i) \tag{1}$$

which can be implemented by VIC pretty easily, as shown in Figure 3.

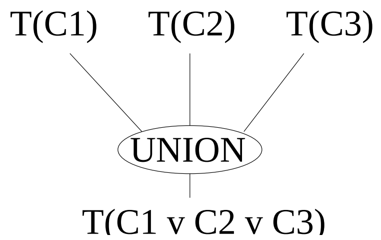

FIG. 3. The truth table of a DNF

Now each $T(C_i)$ is easy to compute: WLOG we assume

$$C_i = x_1 \wedge x_2 \wedge \cdots \wedge x_k \wedge \neg x_{k+1} \wedge \cdots \wedge \neg x_{k+l}$$

then

$$\begin{aligned} T(C_i) &= \{\langle \underbrace{1,1,..,1}_{k}, \underbrace{-1,-1,...,-1}_{l}, 0,...,0\rangle\} \\ &\oplus \{\mathbf{e}_{k+l+1}, -\mathbf{e}_{k+l+1}\} \\ &\oplus \cdots \\ &\oplus \{\mathbf{e}_n, -\mathbf{e}_n\} \end{aligned} \tag{2}$$

here the $\oplus$ is the addition operation in a Vector Integer Circuit. Figure 4 is a example for the clause $x_1 \wedge x_2$ when $n = 4$.

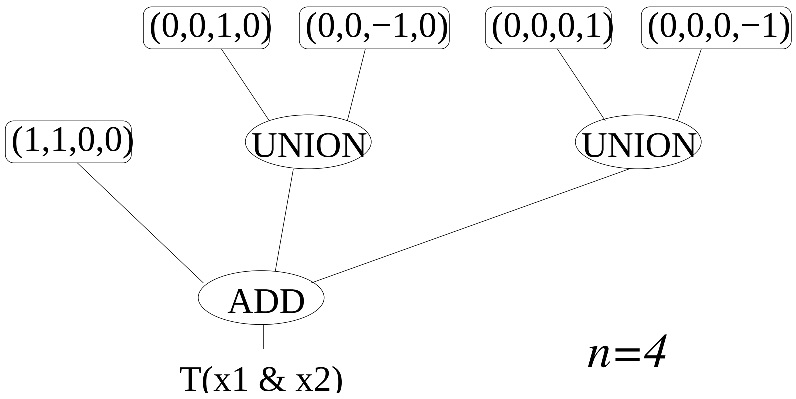

FIG. 4. The truth table of a clause

Thus we have

LEMMA 3.1. *There is a polynomial time algorithm A_1, that takes in a DNF formula ϕ as input, and outputs a Vector Integer Circuit C along with its input x, such that $C(x) = T(\phi)$.*

Proof. Suppose the DNF formula ϕ takes the form

$$\phi = C_1 \vee C_2 \vee \cdots \vee C_m$$

The algorithm A_1 does the following: First it uses Equation 2 to generate sub-circuits that compute $T(C_i)$ for $i = 1, 2, .., m$, then it uses Equation 1 to generate sub-circuits that compute $T(\phi)$, and finally composes the two sub-circuits to generate a circuit that computes the truth table of ϕ. The total size of the circuit is polynomial in n and m, and thus is polynomial in $|\phi|$. ■

3.2. Part 2: Core Reduction — Remove the Quantifiers

This part is the main part of the reduction.

Notice that the QBF we are studying takes the form

$$F = Q_1 x_1 Q_2 x_2 ... Q_n x_n \ \phi(x_1, ..., x_n)$$

By Lemma 3.1, we know we can compute the truth table for ϕ. Now we want to compute the truth table of F. Also notice that since F is **closed**, i.e. there are no free variables in F, its value is either TRUE or FALSE, and thus its truth table is either the set of all n-dimensional truth assignment vectors or the empty set.

Let's look at a QBF with a single quantifier:

DEFINITION 3.4. [Equivalent Formula] Let $F = Q_m x_m\ \phi(x_1, ..., x_m)$ be a QBF with one quantifier, where ϕ is a boolean formula. We define its **Equivalent Formula** to be

$$\phi' = \begin{cases} \phi(x_1, ..., x_{m-1}, 1) \wedge \phi(x_1, ..., x_{m-1}, -1) & \text{if } Q = \forall \\ \phi(x_1, ..., x_{m-1}, 1) \vee \phi(x_1, ..., x_{m-1}, -1) & \text{if } Q = \exists \end{cases}$$

which is also a boolean formula.

Then we obviously have

Claim. Let boolean formulas F and ϕ' be defined as in Definition 3.4. Then F and ϕ' are logically equivalent, i.e., for all $x_1, x_2, ..., x_{m-1} \in \{1, -1\}$, $F(x_1, ..., x_{m-1}) = \phi'(x_1, ..., x_{m-1})$. ∎

We can also demonstrate the relationship between the truth tables of ϕ and ϕ':

LEMMA 3.2. *Let $F = Q_m x_m\ \phi(x_1, ..., x_m)$ be a QBF with one quantifier, and ϕ' be its equivalent formula. We denote the truth table of ϕ by $T = T(\phi)$. Then we have*

•*If $Q_m = \forall$, then $T(\phi') = \{\mathbf{v}\ :\ \mathbf{v} \in T \wedge\ [\mathbf{v}]_m \in T\}$.*
•*If $Q_m = \exists$, then $T(\phi') = \{\mathbf{v}\ :\ \mathbf{v} \in T \vee\ [\mathbf{v}]_m \in T\}$.*

Proof. Immediate from the definition of ϕ and ϕ'. ∎

Conceptually, the above lemma gives a way to remove one quantifier from a boolean formula. Now, given a QBF with n quantifiers, we can remove the quantifiers one by one, from inside to outside.

DEFINITION 3.5. [Equivalent QBF Chain] Let $F = Q_1 x_1 Q_2 x_2 ... Q_n x_n\ \phi(x_1, ..., x_n)$ be a closed QBF. An **Equivalent QBF Chain** of F is a sequence of QBF's: $\{F_0, F_1, ..., F_n\}$, satisfying:

1. $F_n = F$.
2. $F_{k-1} = Q_1 x_1 ... Q_{k-1} x_{k-1}\ \phi_{k-1}$, and ϕ_{k-1} is the equivalent formula of $Q_k x_k \phi_k$, for $k = 1, 2, ..., n$, and ϕ_n is defined to be ϕ.

So all the QBF's in the equivalent QBF chain are closed, and in particular, $F_0 = \phi_0$ is a constant of value either TRUE or FALSE. Again it is immediate that all the QBF's in the chain are equivalent. So F is true iff F_0 is true iff the truth table of F_0 is the set of all n-dimensional truth assignment vectors.

Next we will see how we compute the truth tables of $\phi_n, \phi_{n-1}, ..., \phi_0$ using Vector Integer Circuits. We do so inductively. When we have the truth table of ϕ_0 (it is either the complete set or the empty set), we can tell if the original formula F is TRUE or FALSE.

We define a transition function as follows:

DEFINITION 3.6. [Transition Function] The **Transition Function** f is defined as follows;

$$f(Q,S,k,n)=\begin{cases}\{2\cdot\mathbf{1}\}\otimes(S\cup(S\otimes\{\mathbf{1}_k^n\})) & \text{if } Q=\exists\\ S\oplus(S\otimes\{\mathbf{1}_k^n\}) & \text{if } Q=\forall\end{cases}\tag{3}$$

where Q is a quantifier, T is a set of integer vectors, and $\mathbf{1}$ and $\mathbf{1}_k^n$ are defined as in Definition 2.7. Again, when there is no danger of confusion, we may omit the n.

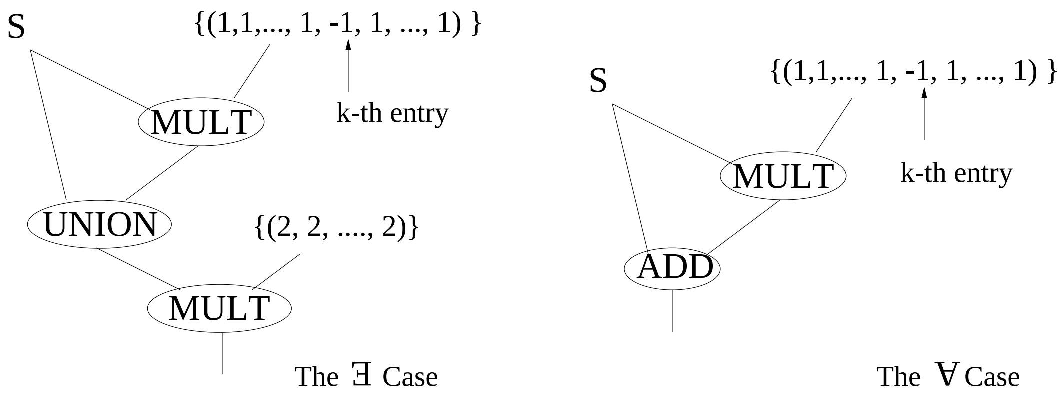

FIG. 5. The transition function

The transition function can be implemented by a Vector Integer Circuit: see Figure 5. Suppose now we have a QBF $F=Q_1x_1Q_2x_2...Q_nx_n\ \phi(x_1,...,x_n)$, where ϕ is a DNF, and $S=T(\phi)$ is the truth table of ϕ. Then we can repeatedly apply the transition f on S:

DEFINITION 3.7. [Operation Chain] For a closed QBF

$$F=Q_1x_1Q_2x_2...Q_nx_n\ \phi(x_1,...,x_n),$$

let S be the truth table of the DNF in F. We define $S_n=S$, and $S_{k-1}=f(Q_k,S_k,k,n)$, for $k=n,n-1,...,1$. The sequence $\{S_0,S_1,...,S_n\}$ is called the **Operation Chain** for F.

We have several observations:

LEMMA 3.3. $||S_k||\leq 2^{n-k}$

This can be easily proved by induction on k.

Next we define the good vectors — roughly speaking, good vectors are the ones whose each entry achieves its maximally possible absolute value.

DEFINITION 3.8. [Good Vectors, Bad Vectors] An integer vector $\mathbf{v} = \langle v^1, v^2, ..., v^n \rangle \in S_k$ is a **Good Vector** in S_k, if $|v^i| = 2^{n-k}$ for all $i = 1, 2, ..., n$. A vector is a **Bad Vector** if it is not a good vector.

The next lemma is an important one.

LEMMA 3.4. *If $\mathbf{v} \in S_{k-1}$ is a good vector in S_{k-1}, then:*

1.*If $Q_k = \forall$, then there exists $\mathbf{u} \in S_k$, such that:*

- $\mathbf{u}$ *is good in S_k.*
- $[\mathbf{u}]_k$*, which is also good, is also in S_k.*
- $\mathbf{v} = 2 \cdot \mathbf{u}$.
- $[\mathbf{v}]_k$*, which is also good, is also in S_{k-1}.*

2.*If $Q_k = \exists$, then there exists $\mathbf{u} \in S_k$, such that:*

- $\mathbf{u}$ *is good in S_k.*
- $\mathbf{v} = 2 \cdot \mathbf{u}$ *or* $\mathbf{v} = 2 \cdot \mathbf{u} \cdot \mathbf{1}_k$ *.*
- $[\mathbf{v}]_k$*, which is also good, is also in S_{k-1}.*

Proof. We prove the two statements of the lemma:

1. Suppose $Q_k = \forall$. Then, by Definition 3.6, there must exist vectors $\mathbf{u}, \mathbf{w} \in S_k$, such that $\mathbf{v} = \mathbf{u} + \mathbf{w} \cdot \mathbf{1}_k$. Notice $\mathbf{v}$ is good in S_{k-1}, so $|v^i| = 2^{n-k+1}$, for all $i = 1, 2, ..., n$. However, $v^i = u^i + w^i$ for $i \neq k$, and $v^k = u^k - w^k$. By Lemma 3.3, each entry of $\mathbf{u}$ and $\mathbf{w}$ is bounded by 2^{n-k}. Thus the only possibility $\mathbf{v}$ is good is that both $\mathbf{u}$ and $\mathbf{w}$ are good in S_k, $u^i = w^i$, for $i \neq k$, and $u^k = -w^k$. That means $\mathbf{w} = [\mathbf{u}]_k$. So $\mathbf{v} = \mathbf{u} + \mathbf{w} \cdot \mathbf{1}_k = \mathbf{u} + [\mathbf{u}]_k \cdot \mathbf{1}_k = 2 \cdot \mathbf{u}$. Furthermore, $[\mathbf{v}]_k = \mathbf{w} + \mathbf{u} \cdot \mathbf{1}_k \in S_{k-1}$.

2. Suppose $Q_k = \exists$. Then, again by Definition 3.6, there exists a vector $\mathbf{u} \in S_k$, such that $\mathbf{v} = 2 \cdot \mathbf{u}$ or $\mathbf{v} = 2 \cdot \mathbf{u} \cdot \mathbf{1}_k$. In either case, this $\mathbf{u}$ should be good, and notice $[\mathbf{v}]_k = \mathbf{v} \cdot \mathbf{1}_k$. So we have $[\mathbf{v}]_k = 2 \cdot [\mathbf{u}]_k \cdot \mathbf{1}_k \in S_{k-1}$ or $[\mathbf{v}]_k = 2 \cdot \mathbf{u} \cdot \mathbf{1}_k \cdot \mathbf{1}_k = 2 \cdot \mathbf{u} \in S_{k-1}$.
■

Lemma 3.4 implies that, if there is a good vector $\mathbf{v}$ in S_{k-1}, then there must be a good vector in S_k corresponding to it, and the neighbor of $\mathbf{v}$ in the k-th place is also in S_k.

In the subsequent discussion, we will only focus on the good vectors in $S_0, S_1, ..., S_n$.

DEFINITION 3.9. [Filtering Function and Filtered Operation Chain] We define a **Filtering Function** $\psi(\cdot, \cdot)$, as follows:

$$\psi(k, S) = \{\mathbf{v} \mid \mathbf{v} \in S,\ |v^i| = 2^{n-k},\ \forall i = 1, 2, ..., n\}$$

For the operation chain $\{S_0, S_1, ..., S_n\}$ of a QBF F we define $G_k = \psi(k, S)$, for $k = 0, 1, ..., n$, and call the sequence $\{G_0, G_1, ..., G_n\}$ the **Filtered Operation Chain** of F.

As its names suggests, $\psi(k, \cdot)$ "filters" all the bad vectors out of S_k.

Then immediately from Lemma 3.4, we have

Claim. $G_k = \psi(k, f(Q_{k+1}, G_{k+1}, k))$. ■

The intuition is: there are two ways to do the filtering: one way is that we filter out the bad vectors in each step of the transition, and only apply the next transition to the remaining good vectors; the other way is that we do all the transitions first, and then filter out the bad ones in the end. Actually these two ways of doing the filtering will yield the same result.

Now we are ready to prove the Main Lemma, which relates our filtered operation chain $\{G_k\}$, to the truth tables of the equivalent QBF chain.

LEMMA 3.5 (Main Lemma). *Let*

$$F = Q_1 x_1 Q_2 x_2 ... Q_n x_n \ \phi(x_1, x_2, ..., x_n)$$

be a QBF, and its equivalent QBF chain be $\{F_0, F_1, ..., F_n\}$. *For each* i, *write* ϕ_i *for the quantifier-free part of* F_i, *and denote the truth table of* ϕ_i *by* $T_i = T(\phi_i)$. *Let* $\{G_k\}$ *be the filtered operation chain of* F *as defined in Definition 3.9. Then we have*

$$G_k = 2^{n-k} \cdot T_k$$

for $k = 1, 2, ..., n$.

Proof.

We proceed by induction on k.

The base case is $k = n$. For $k = n$, Notice both T_n and S_n are truth tables of ϕ, so $T_n = S_n$. Each vector in the truth table is a truth assignment vector, i.e., their entries are all ± 1. Therefore all vectors in S_n are good, and thus $G_n = S_n = T_n$.

Now the inductive case: suppose the Main Lemma is true for k, we look at $k-1$. We inspect the relationship between T_{k-1} and T_k, the truth table for the formula ϕ_{k-1} and ϕ_k. ϕ_{k-1} is the equivalent Formula for $Q_k x_k \ \phi_k$. We look at the two possible cases $Q_k = \forall$ and $Q_k = \exists$, respectively.

• $Q_k = \forall$. For an arbitrary vector $\mathbf{v} \in G_{k-1}$, we know from Lemma 3.4, that there exists $\mathbf{u} \in G_k$, such that $\mathbf{v} = 2 \cdot \mathbf{u}$, and also $[\mathbf{u}]_k \in G_k$. By inductive hypothesis, $G_k \subseteq 2^{n-k} \cdot T_k$ so $\mathbf{u} \in 2^{n-k} \cdot T_k$ and $[\mathbf{u}]_k \in 2^{n-k} \cdot T_k$. By Lemma 3.2, we know $\mathbf{u} \in 2^{n-k} \cdot T_{k-1}$, which means $\mathbf{v} = 2 \cdot \mathbf{u} \in 2^{n-k+1} \cdot T_{k-1}$. Therefore $G_{k-1} \subseteq 2^{n-k+1} \cdot T_{k-1}$.

On the other hand, For an arbitrary vector $\mathbf{u} \in 2^{n-k+1} \cdot T_{k-1}$, we write $\mathbf{u} = 2 \cdot \mathbf{w}$. Then we know $\mathbf{w} \in 2^{n-k} \cdot T_{k-1}$. Again by Lemma 3.2, we know both $\mathbf{w}$ and $[\mathbf{w}]_k$ are in $2^{n-k} \cdot T_k$. By inductive hypothesis, both $\mathbf{w}$ and $[\mathbf{w}]_k$ are in G_k. So $\mathbf{u} = 2 \cdot \mathbf{w} = \mathbf{w} + [\mathbf{w}]_k \cdot \mathbf{1}_k \in G_{k-1}$.Therefore $G_{k-1} \supseteq 2^{n-k+1} \cdot T_{k-1}$.

• $Q_k = \exists$.

For an arbitrary vector $\mathbf{v} \in G_{k-1}$, from Lemma 3.4 we can assume there exists $\mathbf{u} \in G_k$, such that $\mathbf{v} = 2 \cdot \mathbf{u}$ (if it is not the case, consider $[\mathbf{v}]_k$, which is also in

G_{k-1}). By induction hypothesis, we have $\mathbf{u} \in 2^{n-k} \cdot T_k$. By Lemma 3.2, we know both $\mathbf{u} \in 2^{n-k} \cdot T_{k-1}$ and $[\mathbf{u}]_k \in 2^{n-k} \cdot T_{k-1}$, i.e., both $\mathbf{v} = 2 \cdot \mathbf{u} \in 2^{n-k+1} \cdot T_{k-1}$ and $[\mathbf{v}]_k \in 2^{n-k+1} \cdot T_{k-1}$.Therefore $G_{k-1} \subseteq 2^{n-k+1} \cdot T_{k-1}$.

On the other hand, For an arbitrary vector $\mathbf{u} \in 2^{n-k+1} \cdot T_{k-1}$, we write $\mathbf{u} = 2 \cdot \mathbf{w}$. Then we have $\mathbf{w} \in 2^{n-k} \cdot T_{k-1}$. Again by Lemma 3.2, we know either $\mathbf{w}$ or $[\mathbf{w}]_k$ is (or both) in $2^{n-k} \cdot T_k$. By induction hypothesis, either $\mathbf{w}$ or $[\mathbf{w}]_k$ is in G_k, and thus either $\mathbf{u} = 2 \cdot \mathbf{w} \in G_{k-1}$, which means $\mathbf{u} = 2 \cdot \mathbf{w} \cdot \mathbf{1}_k \in G_{k-1}$.Therefore $G_{k-1} \supseteq 2^{n-k+1} \cdot T_{k-1}$.

■

The important consequence of the Main Lemma is that $G_0 = 2^n \cdot T_0$. From Definition 3.3, we know T_0 is either the set or all n-dimensional truth assignment vectors or the empty set. Therefore we know that G_0 is also either the set of all good vectors or the empty set — so we have:

COROLLARY 3.1 (All-or-Nothing Rule). *Let F and G_0 be as defined in Lemma 3.5. If F is TRUE, then G_0 contains all good vectors for S_0, namely, all the 2^n vectors whose entries are $\pm 2^n$; if F is FALSE, then G_0 is the empty set.*

If we combine lemma 3.1 with the All-or-Nothing Rule, we have

LEMMA 3.6. *There exists a polynomial time algorithm A_2, that takes a QBF in standard form, F, and outputs a Vector Integer Circuit C along with the input X, such that F is TRUE, iff $\langle 2^n, 2^n, ..., 2^n \rangle \in C(X)$.*

Proof. The Vector Integer Circuit is constructed as follows: first we apply Lemma 3.1 to construct a sub-circuit that outputs the truth table S of the DNF ϕ inside the QBF F. Then we construct another sub-circuit that applies the transition function defined in Definition 3.6 n times. Putting the two pieces together, we now have a circuit that outputs S_0. By the All-or-Nothing Rule, if F is TRUE, and the filtered set, $G_0 = \psi(0, S_0)$ contains all good vectors, and in particular, vector $\langle 2^n, 2^n, ..., 2^n \rangle$, thus S_0 also contain this vector; if F is FALSE, then G_0 is the empty set, and thus S_0 doesn't contain any good vector, including the vector $\langle 2^n, 2^n, ..., 2^n \rangle$. ■

Notice this result already implies

THEOREM 3.1. *VICE is PSPACE-hard.* ■

3.3. Part 3: Post-processing to extract the result

Now we will reduce the Vector Integer Circuit Evaluation problem to the Integer Circuit Evaluation problem, where each element in the sets is a positive integer.

We state the Chinese Remainder Theorem first: the version given here is adapted from [6].

THEOREM 3.2 (Chinese Remainder Theorem(CRT)). *If $M = \prod_{i=1}^{n} a_i$ and $GCD(a_i, a_j) = 1$ for $i \neq i, 1 \leq i \leq n$, and $1 \leq j \leq n$, then any solution of $f(x) \equiv 0 \pmod{M}$ is a* *simultaneous solution of the system*

$$f(x) \equiv 0 \ (\mathrm{mod}\ a_1)$$
$$f(x) \equiv 0 \ (\mathrm{mod}\ a_2)$$
$$\ldots$$
$$f(x) \equiv 0 \ (\mathrm{mod}\ a_n)$$

and conversely.

It it not hard to see the theorem above is equivalent to the following version of CRT (the main difference here is that we actually "pin down" the range of the residues for $a_1, a_2, ..., a_n$).

THEOREM 3.3 (CRT, version 2). *For any n positive numbers $a_1, a_2, ..., a_n$ which are pairwise relatively prime, that is, $GCD(a_i, a_j) = 1$, for all $1 \le i < j \le n$, let $M = \prod_{i=1}^n a_i$. Let*

$$\begin{aligned} V \;=\; & \{\mathbf{v} \mid \mathbf{v} \text{ is an Integer Vector and} \\ & -a_i/2 < v^i < a_i/2,\ i = 1, 2, ..., n\} \end{aligned}$$

Then there exists a 1-1, polynomial-time computable mapping h from V to $\{1, 2, ..., M\}$. And the mapping is homomorphic, in that for any $\mathbf{u}, \mathbf{v} \in V$, if $\mathbf{u} + \mathbf{v} \in V$ and $\mathbf{u} \cdot \mathbf{v} \in V$, then we have

$$\begin{aligned} h(\mathbf{u}) + h(\mathbf{v}) \equiv \; & h(\mathbf{u} + \mathbf{v}) \ \ (\mathrm{mod}\ M) \\ h(\mathbf{u}) \cdot h(\mathbf{v}) \equiv \; & h(\mathbf{u} \cdot \mathbf{v}) \ \ (\mathrm{mod}\ M) \end{aligned}$$

■

Now it is almost immediate for us to reduce the VICE problem to ICE problem.

LEMMA 3.7. *There exists a polynomial time algorithm A, which takes a QBF in standard form, F, and output a triple $\langle C, x, N\rangle$, where C is an Integer Circuit, x is the input to C, and N a positive integer, such that F is TRUE, iff $N \in C(x)$.*

Proof. First by Lemma 3.6, we have a Vector Integer Circuit C_0, whose output contains $\langle 2^n, 2^n, ..., 2^n\rangle$, iff F is TRUE.

Now we pick the smallest n distinct prime numbers: $p_1 = 2, p_2 = 3, ..., p_n$, and let $a_i = p_i^{n+1}$, for $i = 1, 2, .., n$. Then we know $a_1, a_2, ..., a_n$ are pairwise relatively prime. By the Prime Number Theorem (see [6]), we know $p_n < n^2$, and thus $a_i = p_i^{n+1} \le n^{2n} \le 2^{3n^2}$ and $a_i \ge 2^{n+1}$, for $i = 1, 2..., n$. Let $M = \prod_{i=1}^n a_i$. Then all the $a_1, a_2, ..., a_n$ and M can be generated in time polynomial in n. Now notice for each vector generated in circuit C_0, its entry is bounded by 2^n in absolute value (see Lemma 3.3). So by CRT, we have a 1-1 mapping $h(\cdot)$ from set of all integer vectors in the VIC C_0 to the set of all integers in the IC C_0 modulo M, and $h(\cdot)$ is homomorphic.

Then we construct an Integer Circuit C from C_0: for every input gate in C_0 with vector $\mathbf{v}$, we put an input gate in C, with number $h(\mathbf{v})$. For every computational gate in C_0, we put a computational gate in C in the corresponding place. The type of the computational gates in C_0 and C also correspond to each other: for a union gate in C_0, we put a union gate in C; for an addition gate in C_0, we put an addition gate in C; for a multiplication gate in C_0, we put a multiplication gate in C. Finally, we mark the "output" gate in C in the corresponding place of the "output" gate in C_0 — and we denote the output of this "output" gate by R — notice R is a set of positive integers

Then from CRT and Lemma 3.6, we know, $\exists m \in R, m \equiv 2^n \pmod M$ iff the original QBF, F is TRUE: Notice that $h(\langle 2^n, 2^n, ..., 2^n\rangle) = 2^n$. But we are not done yet: the 1-1 mapping h in the CRT is only homomorphic to modulo additions and multiplications, while Integer Circuits performs integer additions and multiplications. Actually the integers in the output set R can be quite large — much larger than M.

Now we need a way to find out if a set R contains a specific residue modulo M: in other words, we need a way to find out if R contains any integer from the set $\{x \mid x \equiv 2^n \pmod M)\}$.

Let $B = R \oplus \{M - 2^n\}$, then we know the original QBF is TRUE, iff B contains a multiple of M.

We show that each number of B is bounded by 2^{8n^4}:

Claim. $||B|| \leq 2^{8n^4}$.

Notice in the construction of the integer circuit C, the numbers in every input gate is bounded by $M = \prod_{i=1}^{n} a_i$, which in turn is bounded by 2^{3n^3}. When we look at the computations within the circuit C, in part 1, constructing the truth table, at most n multiplications (by a constant, which is bounded by M) are performed to each number; in part 2, there are n transition function operations, each has one multiplication with a constant and one addition or a doubling. Therefore the maximum number we can get is bounded by $(2M)^{2n+1} \leq 2^{7n^4}$ — that is a bound for $||A||$. Finally since $B = A \oplus \{M - \alpha\}$, we have $||B|| \leq 2^{8n^4}$.

Then, we show we can construct a set

$$L = \{k \cdot M \mid 1 \leq k \leq \lambda\}$$

for $\lambda = 2^{8n^4}$:

Notice that for an integer X, if X is odd, then $[X] = ([(X-1)/2] \oplus [(X-1)/2] \oplus \{1\}) \cup \{1, 2\}$; if X is even, then $[X] = ([X/2] \oplus [X/2]) \cup \{1\}$. Thus for any integer X, we can construct the set $[X]$ in $O(\log X)$ time. Therefore we can construct $L = [\lambda] \otimes \{M\}$ in time $O(n^4)$.

Then we know B contains a multiple of M, iff

$$(\lambda + 1) \cdot M \in (B \oplus L)$$

■

So now we have theorem

THEOREM 3.4. *ICE is PSPACE-hard.*

4. SUMMARY AND FURTHER QUESTIONS

We have shown a polynomial-time reduction from Quantified Boolean Formula (QBF) to the Integer Circuit Evaluation problem (ICE), and thus proved that ICE is PSPACE-hard. This result, together with the paper [10], shows the ICE is PSPACE-complete.

There are two techniques used in the proof that have independent interests. The first technique is: we use the truth table to compute the value of a QBF: we started by computing the truth table of the un-quantified DNF formula and then keep track of the truth table as we add quantifiers. Finally we get a truth table that is either contains all truth assignments or is the empty set and thus we can test it. The second technique is to use the Chinese Remainder Theorem to reduce the Vector Integer Circuits to Integer Circuits. In this way we reduce the problem of manipulating vectors (which correspond to truth assignments) to manipulating integers.

An interesting question about this proof is: the All-or-Nothing Rule is actually stronger than needed for our purpose, and what can we do to fully exploit its strength? One thought is that maybe we can slightly weaken the problem. So consider the "size-version" of the ICE: we still have an Integer Circuit C, along with input x and an integer N, but now the problem is not membership but the size — if $|C(x)| = N$ or not.

One can easily modify the Part 3 of the reduction to show the size-version of ICE is also PSPACE-hard, and it is also PSPACE-complete.

But now one can ask the "approximation" problem: given an Integer Circuit C, its input x, and an integer N, and a real number ϵ, we ask if N is an approximation of $|C(x)|$ within a factor of ϵ, or, if $(1-\epsilon)|C(x)| \leq N \leq (1+\epsilon)|C(x)|$. Is the problem still PSPACE-complete? Does the complexity depend on ϵ now?

ACKNOWLEDGMENT

The author would like to thank Steven Rudich for suggesting ideas to simplify the proof and helping refine the paper. The author also like to thank Pierre McKenzie for introducing the problem, for his warm encouragement and careful proof-reading. The author is much indebted to the anonymous referees for pointing out numerous mistakes and giving many valuable suggestions to the previous version of the paper.

REFERENCES

1. Allender, E., Jiao, J., Mahajan, M., and Vinay, V. (1998). Noncommutative arithmetic circuits: depth reduction and size lower bounds, *Theoretical Computer Science*, Vol. 209 (1998), pp. 47 - 86.
2. Babai, L., Fortnow, L. (1991). Arithmetization: A new method in structural complexity theory. *Computational Complexity*, Vol. 1 (1991). pp. 41-66.
3. Beaudry, M., McKenzie, P., Péladeau, P., and Thérien, D. (1997). Finite monoids: from word to circuit evaluation. *SIAM Journal on Computing*. Vol. 26, No.1, (1997), pp.138-152.
4. Hua, L. K.(1982). *Introduction to Number Theory*. Springer-Verlag, Berlin.
5. Cai, J. and Furst, M. L. (1987). PSPACE survives three-bit bottlenecks. In Proceedings, *Structure in Complexity Theory, 2nd Annual Conference*, pages 94-102, Cornell University, Ithaca, NY, 16-19 June 1987. IEEE Computer Society Press.
6. Long, C. T. (1965). *Elementary Introduction to Number Theory*. D.C.Heath and Company, Boston.

7. McKenzie, P., Vollmer, H., and Wagner, K. W. (2000). Arithmetic Circuits and Polynomial Replacement Systems. To appear in the *Proceedings of the 2000 FSTTCS conference*, New Delhi, India, December 2000.

8. Papadimitriou, C.H. (1994). *Computational Complexity*, pp. 455 - 491. Addison Wesley.

9. Stockmeyer, L. J. and Meyer, A. R. (1973). Word problems requiring exponential time. In Proceedings, *The 5th Annual ACM Symposium on Theory of Computing*, pages 1-9, Austin, Texas, 30 April-2 May 1973.

10. Wagner, K. W. (1984). The complexity of problems concerning graphs with regularities. *Technical report N/84/52, Friedrich-Schiler-Universität Jena*, 1984. Extended abstract in *Proceedings of the 11th Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 176, pp. 544 – 552, Springer-Verlag, 1984.