Title	Expressive Power of Decision Diagrams and Quantum Computation Models
Author(s)	中西, 正樹
Citation	大阪大学, 2002, 博士論文
Version Type	VoR
URL	https://hdl.handle.net/11094/2704
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

I.S.8394

Expressive Power of Decision Diagrams and Quantum Computation Models

Masaki Nakanishi

November 2001

Expressive Power of Decision Diagrams and Quantum Computation Models

Masaki Nakanishi

November 2001

Abstract

In this dissertation, we discuss expressive power of some decision diagrams, which are representations of functions, and some quantum computation models.

Many kinds of representations of functions have been proposed such as truth tables, CNF's and various kinds of decision diagrams. It is desired that they have much expressive power, that is, the size of representations of functions should be small, or the class of functions that can be represented should be large on the condition that the size of representations is restricted.

When we discuss abilities of computation models such as finite automata and push down automata, we associate computation models with formal languages. Then abilities of computation models are evaluated in terms of to what extent the computation models can recognize languages. This evaluation is also based on the criteria of expressive power.

In this dissertation, we show several results concerning expressive power such as lower bounds on the size of a certain decision diagram and abilities to recognize languages of certain computation models.

In a practical sense, it is inconvenient if finding a small representation takes too much time even though the representation has a small expression. Thus it is also important to investigate how complex finding a small (or minimum) representation is. We also discuss complexity of finding a minimum representation of a certain decision diagram.

In this dissertation, we deal with two sorts of decision diagrams. One is a binary moment diagram, which represents a function from binary vectors to integers. The other is a Kronecker functional decision diagram, which represents a boolean function. Binary moment diagrams can represent several arithmetic functions, including multiplication, efficiently, while binary decision diagrams need an exponential number of nodes to represent (a particular bit of) multiplication. However, some experimental results show that lower bounds on the size of a binary moment diagram representing division may be exponential. We give a theoretical proof to this result. Kronecker functional decision diagrams are generalized decision diagrams, which take any decomposition type for each variable.

The size of a Kronecker functional decision diagram depends on a given decomposition type list. We show that the problem of finding the best decomposition type list is NP-hard.

We deal with "quantum computation" as well as "classical computation". In this dissertation, we propose two quantum computation models, quantum branching programs and non-deterministic quantum finite automata. Branching programs are decision diagrams that are also known as (unordered) binary decision diagrams. We show that under a bounded-width restriction, ordered quantum branching programs can compute some function that ordered probabilistic branching programs cannot compute. We also show that the class of languages recognized by non-deterministic quantum finite automata properly includes the regular languages. This result means that non-deterministic quantum finite automata are strictly more powerful than classical non-deterministic finite automata since finite automata (regardless of deterministic or not) can recognize exactly the regular languages.

Acknowledgement

I am deeply indebted to many people for the advice and supports on this work. I would especially like to thank my supervisor Professor Toshinobu Kashiwabara for his invaluable suggestions, advice, and discussions throughout the work. I am obliged to Professor Toru Fujiwara and Professor Toshimitsu Masuzawa for their helpful comments and suggestions.

I would like to thank Associate Professor Kiyoharu Hamaguchi for his discerning comments and worthy discussions. I would also like to thank Research Assistant Takashi Kizu for his helpful supports. I would also like to express hearty thanks to Mrs. Yukiko Tanobe for her kind supports.

Finally, I would like to thank all the members of Kashiwabara Laboratory, Graduate School of Engineering Science, Osaka University.

List of Publications

Publications Related to the Thesis

• Journal Papers

[1] M. Nakanishi, K. Hamaguchi, and T. Kashiwabara: An Exponential Lower Bound on the Size of a Binary Moment Diagram Representing Integer Division, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol.E82-A, no.5, pp.756–766, May 1999.

[2] M. Nakanishi, M. Sawada, K. Hamaguchi, and T. Kashiwabara: The Complexity of the Decomposition Type Selection Problem of Kronecker Functional Decision Diagrams, IEICE Transactions on Information and Systems, vol.J83-D-I, no.1, pp.115–120, January 2000 (in Japanese). (Letter)

[3] M. Nakanishi, T. Indoh, K. Hamaguchi, and T. Kashiwabara: On the Power of Non-deterministic Quantum Finite Automata: IEICE Transactions on Information and Systems (to appear).

• International Conference

[4] M. Nakanishi, K. Hamaguchi, and T. Kashiwabara: Ordered Quantum Branching Programs Are More Powerful than Ordered Probabilistic Branching Programs under a Bounded-Width Restriction, Proceedings of 6th Annual International Computing and Combinatorics Conference (COCOON 2000), Lecture Notes in Computer Science, vol.1858, pp.467 – 476, July 2000.

• Workshops and Technical Reports

[5] M. Nakanishi, K. Hamaguchi, and T. Kashiwabara: Exponential Lower Bounds of the Sizes of Binary Moment Diagrams Representing Divi-

sion, IEICE Technical Report, COMP97-53, pp.63–70, October 1997 (in Japanese).

[6] M. Nakanishi, K. Hamaguchi, and T. Kashiwabara: An Exponential Lower Bound on the Size of a Binary Moment Diagram Representing Division, Booklet of 7th International Workshop on Post-Binary ULSI Systems, pp.42–43, May 1998.

[7] M. Nakanishi, K. Hamaguchi, and T. Kashiwabara: On Comparison between Quantum Branching Programs and Probabilistic Branching Programs under Read-Once and Bounded-Width Restrictions, IEICE Technical Report, COMP99-95, pp.127–134, March 2000 (in Japanese).

[8] M. Nakanishi, K. Hamaguchi, and T. Kashiwabara: On the Power of Bounded Width Quantum Branching Programs, NAIST Technical Report, NAIST-IS-TR2000010, November 2000 (in Japanese).

[9] M. Nakanishi, T. Indoh, K. Hamaguchi, and T. Kashiwabara: Non--deterministic Quantum Finite Automata, NAIST Technical Report, NAIST-IS-TR2000014, December 2000 (in Japanese).

[10] M. Nakanishi, K. Hamaguchi, and T. Kashiwabara: On the Power of Bounded Width Quantum Branching Programs, LA Symposium, January 2001 (in Japanese).

[11] M. Nakanishi, T. Indoh, K. Hamaguchi, and T. Kashiwabara: On the Power of Quantum Pushdown Automata with a Classical Stack and 1.5-way Quantum Finite Automata, NAIST Technical Report, NAIS-T-IS-TR2001005, March 2001.

Other Publications

• International Conference

[12] K. Nakamura, M. Nakanishi, T. Horiyama, M. Suzuki, S. Kimura, and K. Watanabe: A Real-Time User-Independent Eye Tracking LSI with Environment Adaptability, Proceedings of 10th Workshop on Synthesis and System Integration of Mixed Technologies (SASIMI 2001), pp.357–361, October 2001.

• Workshops

[13] K. Nakamura, M. Nakanishi, T. Horiyama, M. Suzuki, S. Kimura, and K. Watanabe: Real-Time Eye Tracking LSI for Eye-Based Interface, 4th System LSI Biwako Workshop, pp.267–270, November 2000 (in Japanese).

[14] K. Watanabe, S. Kimura, T. Horiyama, M. Nakanishi, Y. Itoh, K. Nakamura, F. Lo, and F. Miyawaki: A Design Methodology of Unconventional Systems – Promotion of Designers of New Information Systems –, Symposium on Information System and Social Environment, IPSJ Symposium Series, vol.2001, no.3, pp.25–32, January 2001 (in Japanese).

Contents

List of Figures

Chapter 1

Introduction

To discuss how powerful particular algorithms and, in addition, computers on which the algorithms run are, we analyze them in various kinds of criterion such as time complexity and space complexity. In this dissertation, we focus on the criteria of *expressive power*.

To manipulate functions is one of basic operations of computation. Many kinds of representations of functions have been proposed such as truth tables, CNF's and various kinds of decision diagrams. It is desired that they have much expressive power, that is, the size of representations of functions should be small, or the class of functions that can be represented should be large on the condition that the size of representations is restricted.

When we discuss abilities of computation models such as finite automata and push down automata, we associate computation models with formal languages. Then abilities of computation models are evaluated in terms of to what extent the computation models can recognize languages. This evaluation is also based on the criteria of expressive power.

In this dissertation, we show several results concerning expressive power such as lower bounds on the size of a certain representation of functions and abilities to recognize languages of certain computation models.

In a practical sense, it is inconvenient if finding a small representation takes too much time even though the representation has a small expression. Thus it is also important to investigate how complex finding a small (or minimum) representation is. We also discuss complexity of finding a minimum representation of a certain data structure.

We deal with several sorts of decision diagrams as representations of functions. Decision diagrams are directed acyclic graphs representing functions. They can represent some functions with relatively small space. The function is decomposed at each node according to the decomposition type such as Shannon expansion and

Davio expansion, which is associated with each of the variables. The children of the node represent the terms of the decomposed function. As for *ordered* decision diagrams, given a variable ordering, the variables appear on any paths from the root to terminal nodes according to the ordering. With a fixed variable ordering and fixed decomposition types, a decision diagram can represent a function uniquely. This property of unique representation is used in the area of verification of circuits[9, 10, 16], comparing the decision diagram that represents the function computed by the circuit with the decision diagram that represents the specification.

The computation time and space of manipulating a decision diagram depend on the size of the decision diagram. Studies on upper/lower bounds on the size of decision diagrams and the algorithms of finding the best ordering have been made[5, 8, 17, 22].

In this dissertation, we deal with two sorts of decision diagrams. One is a binary moment diagram, which represents a function from binary vectors to integers. The other is a Kronecker functional decision diagram, which represents a boolean function. Binary moment diagrams can represent several arithmetic functions, including multiplication, efficiently, while binary decision diagrams need an exponential number of nodes to represent (a particular bit of) multiplication. Thus they are used in the area of verification of arithmetic circuits. However, some experimental results show that lower bounds on the size for division may be exponential. In Chapter 2, we give a theoretical proof to this result, that is, we show that a binary moment diagram needs an exponential number of nodes to represent division. Kronecker functional decision diagrams are generalized decision diagrams, which take any decomposition type for each variable. The size of a Kronecker functional decision diagram depends on a given decomposition type list. In Chapter 3, we show that the problem of finding the best decomposition type list is NP-hard.

The representations mentioned above are based on "classical computation". We deal with "quantum computation" as well as "classical computation", and propose several quantum computation models. Since Shor developed a polynomial time factoring algorithm for quantum computers[21], much attention has focused on quantum computation, and it has been shown that quantum computation models can be more powerful than classical counterparts. Quantum computers can exploit quantum effects, and can be in superpositions of configurations in their computation processes. With this feature, the computation by quantum computers can be regarded as a kind of parallel computation, called "quantum parallelism". This quantum parallelism is the source of the power of quantum computation.

Several kinds of quantum computation models have been proposed such as quantum Turing machines[11], quantum circuits[24] and quantum finite automata [3, 4, 6, 18, 20]. It is believed that quantum Turing machines are more powerful than classical Turing machines. Moreover it has been shown that even restricted (or finite) models of quantum computation can be more powerful than classical counterparts in spite that quantum computation models must be reversible in order to obey quantum theory. For example, the class of languages recognized by 2-way quantum finite automata properly includes the regular languages. However, the constraint of the reversibility makes some quantum computation models less powerful than classical counterparts such that 1-way quantum finite automata can recognize only a proper subset of the regular languages while 1-way (classical) finite automata can recognize exactly the regular languages. Thus it is unclear what kinds of quantum computation models can be more/less powerful than classical counterparts, and it is important to investigate quantum computation models in detail.

In this dissertation, we propose two quantum computation models, quantum branching programs and non-deterministic quantum finite automata. Branching programs are decision diagrams that are also known as (unordered) binary decision diagrams. We use "branching programs" rather than "binary decision diagrams" when we consider them as computation models, not as representations of functions. In Chapter 4, we show that under a bounded-width restriction, ordered quantum branching programs can compute some function that ordered probabilistic branching programs cannot compute. In Chapter 5, we show that the class of languages recognized by non-deterministic quantum finite automata properly includes the regular languages. This result means that non-deterministic quantum finite automata are strictly more powerful than classical non-deterministic finite automata since finite automata (regardless of deterministic or not) can recognize exactly the regular languages.

Chapter 6 concludes this dissertation, and discusses future works.

Chapter 2

On the Size of Binary Moment Diagrams Representing Division

2.1 Introduction

A binary decision diagram (BDD) [2, 7] is a directed acyclic graph representing a boolean function. BDD's can represent many useful boolean functions with relatively small space. The size of a BDD depends on a given variable ordering. The known lower bounds on the size of a BDD representing (a particular bit of) multiplication[8] or division[17] are exponential, therefore BDD's are inconvenient to represent arithmetic functions. Thus binary moment diagrams were introduced to represent arithmetic functions efficiently by Bryant et al [9].

A binary moment diagram (BMD) is a directed acyclic graph representing a function from binary vectors to integers. A multiplicative binary moment diagram (*BMD) is an extension of a BMD with edge weights attached, and can represent some arithmetic functions, including multiplication, with the polynomial number of nodes in terms of the number of inputs [9]. Thus it is used widely in the area of verification of arithmetic circuits [9, 10, 16]. On the other hand, it had been thought that division could not be represented efficiently even by a BMD or a *BMD. Some experimental results show that lower bounds for division may be exponential. However, there is no theoretical proof showing lower bounds for division since the way of constructing BMD's or *BMD's for arithmetic functions is more complex than that for BDD's. Thus it is not trivial to prove exponential lower bounds for BMD's or *BMD's. In this chapter, we show exponential lower bounds on the size of a *BMD representing a quotient function or a remainder function. This also means that BMD's cannot represent these functions in polynomial sizes since *BMD's can represent arbitrary functions more compactly than BMD's.

In Section 2.2, we explain the notion of fooling sets, which are useful to show lower bounds, and show the relation between the cardinality of a fooling set and the number of nodes of a *BMD. In Section 2.3 and 2.4, we show that lower bounds on the size of a *BMD representing a quotient or a remainder function are $\Omega(2^{n/24})$.

2.2 Preliminaries

2.2.1 Multiplicative Binary Moment Diagrams

A multiplicative binary moment diagram (*BMD) is a directed acyclic graph representing a function from binary vectors to integers ($f : \{0,1\}^n \rightarrow \mathbb{Z}$), where $\mathbb{Z}$ is the set of all integers.

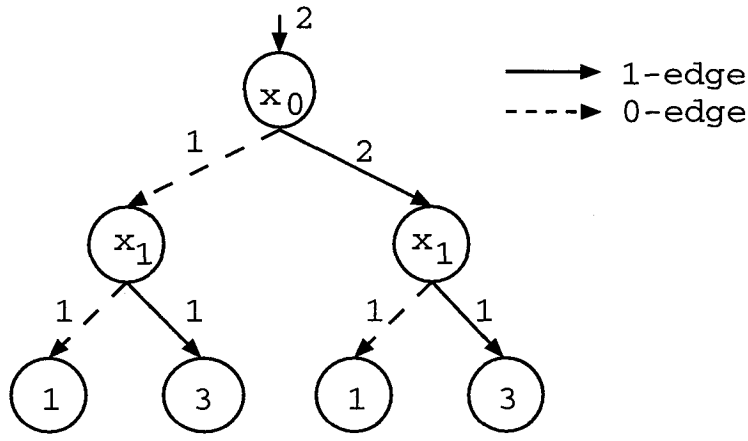
In Bryant and Chen's original definition [9], we can construct a multiplicative binary moment diagram uniquely by sharing isomorphic subgraphs and removing redundant nodes. In this chapter, however, we allow multiplicative binary moment diagrams to contain multiple isomorphic subgraphs and to have redundant nodes, since our purpose is to investigate a lower bound. Our results on lower bounds guarantee that, even if we have all isomorphic subgraphs shared with each other as much as possible, the size of a BMD or a *BMD representing a quotient or a remainder cannot be less than the lower bound. In other words, the lower bound is still valid for the original BMD's and *BMD described by Bryant and Chen.

A variable is attached to each internal node, and a constant integer value is attached to each terminal node. Each internal node has exactly two outgoing edges, called the 0-edge and the 1-edge respectively. Each node represents a function from binary vectors to integers. For an internal node v, let the function represented by v and the variable attached to v be $f[v]$ and $var(v)$ respectively, let the node to which the 0-edge of v points and the node to which the 1-edge of v points be $low(v)$ and $high(v)$ respectively, and let the non-zero integer weight attached to the 0-edge of v and that to the 1-edge of v be 0-$weight(v)$ and 1-$weight(v)$ respectively. We do not allow *BMD's to have zero edge weight.

The relation between functions represented by the nodes of a multiplicative binary moment diagram is defined as follows:

$$f[low(v)] = \frac{f[v]|_{var(v)=0}}{0\text{-}weight(v)}$$

$$f[high(v)] = \frac{f[v]|_{var(v)=1} - f[v]|_{var(v)=0}}{1\text{-}weight(v)},$$



$2 \cdot f[root] = 12x_0x_1 + 4x_0 + 6x_1 + 2, \pi = (x_0 < x_1)$

Figure 2.1: An example of a *BMD.

where $f|_{x=a}$ is a function obtained by substituting a for a variable x of the function f. A terminal node represents the constant function, whose value is attached to the node. Let *root-weight* be the weight of the edge pointing to the root node. The function *root-weight* $\cdot f[root]$ denotes the function represented by the multiplicative binary moment diagram (See Figure 2.1).

We assume that a variable ordering is given, and the appearances of variables along any path from the root node to a terminal node obey the ordering. That is, given $\pi = (x_{k_1} < x_{k_2} < \ldots < x_{k_n})$ for a set of variables $\{x_1, x_2, \ldots, x_n\}$, x_{k_i} precedes x_{k_j} on any path from the root node to a terminal node if $i < j$, where $(k_1, k_2, \ldots, k_n)$ is a permutation of $(1, 2, \ldots, n)$. Each variable can appear at most once on any path from the root node to a terminal node.

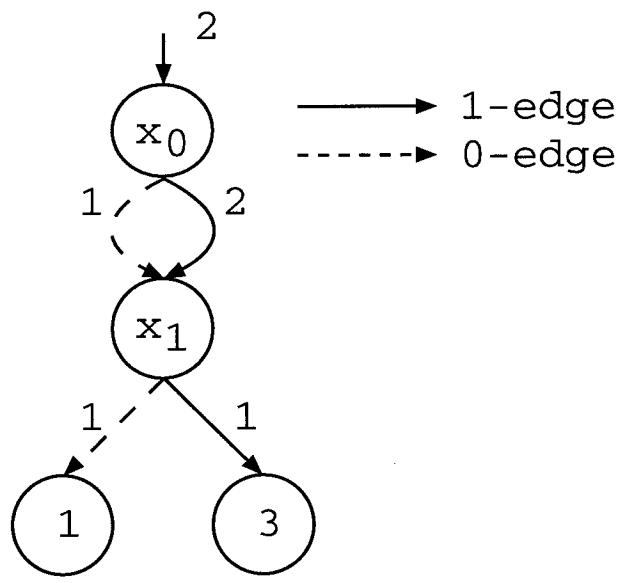
For convenience, we describe 'binary moment diagram' and 'multiplicative binary moment diagram' as 'BMD' and '*BMD' respectively. When constructing a BMD or a *BMD, we can share the nodes that represent the same functions (See Figure 2.2).

We give some more definitions to describe properties of *BMD's.

2.2.2 Input Assignments and Left Terms

Let f be a function from binary vectors to integers. Let $X = \{x_1, x_2, \ldots, x_n\}$ be the set of inputs of f. An input assignment $a : X \rightarrow \{0, 1\}$ is an assignment of boolean values to inputs. Given an input assignment a, $f(a) \in \mathbb{Z}$ denotes the resulting output value, where $\mathbb{Z}$ is the set of all integers.

Given a variable ordering $\pi = (x_{k_1} < x_{k_2} < \ldots < x_{k_n})$, where $(k_1, k_2, \ldots, k_n)$



$2 \cdot f[root] = 12x_0x_1 + 4x_0 + 6x_1 + 2,\ \pi = (x_0 < x_1)$

Figure 2.2: An example of sharing nodes of a *BMD.

is a permutation of $(1, 2, \ldots, n)$, and some i $(1 \leq i \leq n)$, we define the set of variables L (resp., R) as $L = \{x_{k_1}, x_{k_2}, \ldots, x_{k_i}\}$ (resp., $R = \{x_{k_{i+1}}, x_{k_{i+2}}, \ldots, x_{k_n}\}$). We call L (resp., R) a left (resp., a right) partition. For particular partitions L and R, a left (resp., a right) input assignment $l : L \rightarrow \{0, 1\}$ (resp., $r : R \rightarrow \{0, 1\}$) is an assignment of boolean values to the inputs in L (resp., R). We define a left input assignment as $l = \varepsilon$ when $L = \emptyset$. When a value a is assigned to a variable x in l, we describe as $l(x) = a$. The assignment $l \cdot r$ denotes the complete input assignment resulting from a left input assignment l and a right input assignment r. When $x_{k_1} = a_1$, $x_{k_2} = a_2$, ..., $x_{k_i} = a_i$ in a left input assignment l for π given above, l is described as $l = (a_1, a_2, \ldots, a_i)$. Given a left input assignment l, $f(l)$ denotes the resulting function, whose set of inputs is R. For example, for the function $f = x_0x_2 + x_1x_2 + x_3$, the variable ordering $\pi = (x_0 < x_1 < x_2 < x_3)$ and the left input assignment $l = (10)$, we obtain $f(l) = x_2 + x_3$.

For a left input assignment l, we define a left term f_l of a function f recursively as follows:

Case $l = \varepsilon$: $f_\varepsilon = f$

Case $l \neq \varepsilon$:

$$f_l = \begin{cases} f_{l'}|_{x_{k_i}=0} & (a_i = 0) \\ f_{l'}|_{x_{k_i}=1} - f_{l'}|_{x_{k_i}=0} & (a_i = 1), \end{cases}$$

where $l' = (a_1, a_2, \ldots, a_{i-1})$ for $l = (a_1, a_2, \ldots, a_i)$, and $\pi = (x_{k_1} < x_{k_2} < \ldots <$

x_{k_n}).

For example, for the function $f = x_0x_2 + x_1x_2 + x_3$, the variable ordering $\pi = (x_0 < x_1 < x_2 < x_3)$ and the left input assignment $l = (10)$, we obtain $f_l = x_2$. Note that f_l differs from $f(l)$.

As for the relation between a *BMD representing a function f and a left term of f, the following lemma holds.

Lemma 1 *Let f be a function from binary vectors to integers. Given a variable ordering π, let l be a left input assignment. We consider a *BMD that is constructed according to π. We follow edges starting from the root node to a node to which the last variable in l is attached as follows:*

If the assignment to the variable on the current node is 0 in l, then we follow the 0-edge.
If the assignment to the variable on the current node is 1 in l, then we follow the 1-edge.

We suppose that we arrive at a node u. Let p be the product of the weights of the edges on the path from the root node to u. Then the function represented by u is f_l/p.

(Proof) This lemma is obvious by the definitions. □

Let l be a left input assignment. Let $\hat{l}$ be a left input assignment such that if the assignment to a variable x is 1 in $\hat{l}$, then the assignment to x is 1 in l. We describe as $\hat{l} \lesssim l$. We define as $\varepsilon \lesssim \varepsilon$. For example, when $l = (101)$, then $(101) \lesssim l$, $(100) \lesssim l$, $(001) \lesssim l$ and $(000) \lesssim l$. We call the number of 1's in an assignment l the weight of l. We define a function Sgn as

$$Sgn(l) = \begin{cases} 1 & \text{(the weight of } l \text{ is even)} \\ -1 & \text{(the weight of } l \text{ is odd).} \end{cases}$$

The weight of ε is 0 and $Sgn(\varepsilon) = 1$.

To calculate a left term, we introduce the following lemma.

Lemma 2 *For a function f and a left input assignment l,*

$$f_l = Sgn(l) \cdot \sum_{\hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l}).$$

(Proof) Without loss of generality, let $\pi = (x_1 < x_2 < \ldots < x_n)$ be a given variable ordering. And let $l = (a_1, a_2, \ldots, a_i)$ be a left input assignment for the set of inputs $X = \{x_1, x_2, \ldots, x_n\}$. Let m be the weight of l. We prove this lemma by induction on m.

When $m = 0$ (including the case that $l = \varepsilon$). Since $\{\hat{l}|\hat{l} \lesssim l\} = \{l\}$ ($\{\hat{l}|\hat{l} \lesssim \varepsilon\} = \{\varepsilon\}$) and $f_l = f(l)$, $Sgn(l)\sum_{\hat{l}\lesssim l} Sgn(\hat{l})f(\hat{l}) = Sgn(l)^2 f(l) = f(l) = f_l$. Hence the lemma holds for $m = 0$.

We suppose that the lemma holds for $m = k$. We consider the case that $m = k + 1$.

We suppose that a_i=1. The case that $a_i = 0$ is described later. We consider the left input assignment $l' = (a_1, a_2, \ldots, a_{i-1})$, whose weight is k. Then,

$$
\begin{aligned}
& f_l \\
= \; & f_{l'}|_{x_i=1} - f_{l'}|_{x_i=0} \\
& \quad \text{(by the definition of left terms)} \\
= \; & \left[Sgn(l') \sum_{\hat{l}' \lesssim l'} Sgn(\hat{l}') f(\hat{l}') \right] \Bigg|_{x_i=1} - \left[Sgn(l') \sum_{\hat{l}' \lesssim l'} Sgn(\hat{l}') f(\hat{l}') \right] \Bigg|_{x_i=0} \\
& \quad \text{(by the hypothesis of induction)} \\
= \; & Sgn(l') \left[- \sum_{\hat{l} \in \{\hat{l}|\hat{l}(x_i)=1, \hat{l} \lesssim l\}} Sgn(\hat{l}) f(\hat{l}) \right] - Sgn(l') \left[\sum_{\hat{l} \in \{\hat{l}|\hat{l}(x_i)=0, \hat{l} \lesssim l\}} Sgn(\hat{l}) f(\hat{l}) \right] \\
= \; & Sgn(l) \sum_{\hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l}). \\
& \quad (\text{by } -Sgn(l') = Sgn(l))
\end{aligned}
$$

We now consider the case that $a_i = 0$. Let j_{max} be the maximum of j such that $j < i$ and $a_j = 1$. For the left input assignment $l'' = (a_1, a_2, \ldots, a_{j_{max}})$, we can apply the same method as in the case that $a_i = 1$, identifying l'' with l. Thus the following equation is obtained.

$$f_{l''} = Sgn(l'') \sum_{\hat{l} \lesssim l''} Sgn(\hat{l}) f(\hat{l})$$

By the definition of left terms, the following equation is also obtained.

$$f_l = f_{l''}|_{x_{j_{max}+1}=0, x_{j_{max}+2}=0, \ldots, x_i=0}$$

Thus,

$$
\begin{aligned}
f_l &= f_{l''}|_{x_{j_{max}+1}=0, x_{j_{max}+2}=0, \ldots, x_i=0} \\
&= [Sgn(l'') \sum_{\hat{l} \lesssim l''} Sgn(\hat{l}) f(\hat{l})]|_{x_{j_{max}+1}=0, \ldots, x_i=0} \\
&= Sgn(l) \sum_{\hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l}).
\end{aligned}
$$

Hence, the lemma is proved. □

2.2.3 Fooling Sets

Given a function f and a variable ordering π, a set A of left input assignments with the same length is a fooling set if and only if the following condition holds for any two distinct left input assignments $l, l' \in A$.

$$\begin{aligned} & \exists r\ f_l(r) = 0, f_{l'}(r) \neq 0 \\ \text{or}\ & \exists r\ f_l(r) \neq 0, f_{l'}(r) = 0 \\ \text{or}\ & \exists r, r'\ f_l(r) f_{l'}(r') \neq f_{l'}(r) f_l(r'), \end{aligned}$$

where r and r' are right input assignments. When the cardinality of the set is less than 2, we define the set to be a fooling set.

Note that this definition of a fooling set differs from Bryant's [8].

The following theorem states the relation between the cardinality of a fooling set and the number of nodes of a *BMD representing a function f.

Theorem 1 *Given a function f, if there exists a fooling set for f whose cardinality is more than or equal to c for any variable ordering π, then any *BMD representing f has at least c nodes.*

(Proof) Considering any *BMD representing f, let π be its variable ordering, and let A be its fooling set whose cardinality is more than or equal to c. For any two distinct left input assignments l, $l' \in A$, let P and Q be the nodes defined in Lemma 1 as u for l and l' respectively, and let f_l/p and $f_{l'}/q$ be the functions represented by P and Q respectively, where p and q are the integers defined in Lemma 1 as p.

In the following, we show $P \neq Q$. Since l and l' belong to the fooling set, at least one of the following cases holds.

Case There exists a right input assignment r such that $f_l(r) = 0$ and $f_{l'}(r) \neq 0$: $f_l(r)/p(= 0) \neq f_{l'}(r)/q(\neq 0)$. Hence, $P \neq Q$.

Case There exists a right input assignment r such that $f_l(r) \neq 0$ and $f_{l'}(r) = 0$: $f_l(r)/p(\neq 0) \neq f_{l'}(r)/q(= 0)$. Hence, $P \neq Q$.

Case There exist two distinct right input assignments r and r' such that $f_l(r) \cdot f_{l'}(r') \neq f_{l'}(r) \cdot f_l(r')$:

We suppose that $P = Q$. Then $f_l(r)/p = f_{l'}(r)/q$ and $f_l(r')/p = f_{l'}(r')/q$. Thus, $f_l(r) \cdot f_{l'}(r') = f_{l'}(r) \cdot f_l(r')$, a contradiction. Hence $P \neq Q$.

Thus there is a node corresponding to each element in the fooling set, and they are distinct. Therefore the *BMD has at least c nodes. □

2.2.4 p-Splits

For the set of inputs $X = \{x_{n-1}, x_{n-2}, \ldots, x_0\}$ and $Y = \{y_{n-1}, y_{n-2}, \ldots, y_0\}$, we define binary representation of X and Y as

$$\begin{aligned} ||X|| &= 2^{n-1}x_{n-1} + 2^{n-2}x_{n-2} + \ldots + 2^0 x_0 \text{ and} \\ ||Y|| &= 2^{n-1}y_{n-1} + 2^{n-2}y_{n-2} + \ldots + 2^0 y_0, \end{aligned}$$

respectively.

Let f be a quotient function of $||X||$ divided by $||Y||$, whose set of inputs is $X \cup Y$.

We assume that n is even. Note that our purpose is to show an exponential lower bound. Thus, if n is odd, we can treat n bit division as $n-1$ bit division by regarding both x_{n-1} and y_{n-1} as 0. Then we obtain a lower bound $\Omega(c^n)$ from the result for the case that n is even by ignoring a constant coefficient of $\Omega(c^{n-1})$.

We define X_U and X_D as

$$\begin{aligned} X_U &= \{x_{n-1}, x_{n-2}, \ldots, x_{n/2}\} \text{ and} \\ X_D &= \{x_{n/2-1}, x_{n/2-2}, \ldots, x_0\}. \end{aligned}$$

Given a variable ordering π, a left partition L and a right partition R, we define as follows:

$$\begin{aligned} X_{UL} = X_U \cap L \quad , \quad X_{DL} = X_D \cap L \\ X_{UR} = X_U \cap R \quad , \quad X_{DR} = X_D \cap R. \end{aligned}$$

For an integer p $(1 \leq p \leq n-1)$, we define $Args_p$ as

$$Args_p = \left\{ (x_a, x_b) \;\middle|\; \begin{array}{l} a - b = p, \\ n-1 \geq a \geq n/2, \\ n/2 > b \geq 0 \end{array} \right\}.$$

(See Figure 2.3)

We define a p-split $Split_p$ as

$$Split_p = Args_p \cap [(X_{UL} \times X_{DR}) \cup (X_{UR} \times X_{DL})].$$

(See Figure 2.4)

The following lemma is proved in [8].

Lemma 3 *If $|X \cap L| = |X \cap R|$, then there exists some integer p and the cardinality of $Split_p$ is at least $n/8$.* □

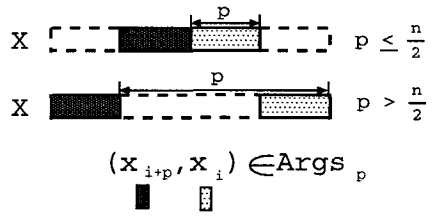


Figure 2.3: $Args_p$

Variable	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
Partition	L	L	R	R	L	L	R	L	L	R	L	R

If $p = 4$, $Args_p = \{(x_9, x_5), (x_8, x_4), (x_7, x_3), (x_6, x_2)\}$ and $Split_p = \{(x_8, x_4), (x_6, x_2)\}$.

Figure 2.4: An example of $Split_p$.

In the following, we assume that $|X \cap L| = |X \cap R|$. For the integer p defined in Lemma 3, we define s and t as

$$\begin{aligned} s = n/2 - p, \quad & t = n/2 \ \ (\text{if } p \leq n/2), \\ s = 0, \quad & t = p \ \ (\text{if } p > n/2). \end{aligned}$$

These s and t are illustrated in Figure 2.5. We define s' and t', if exists, as

$$\begin{aligned} s' &= \min\{s' | n/2 > s' \geq s, y_{s'} \in R\} \text{ and} \\ t' &= \min\{t' | n - 1 \geq t' \geq t, y_{t'} \in R\}. \end{aligned}$$

We define a high split bit s_h and a low split bit s_l as follows:

Case $s' - s \geq t' - t$, or s' does not exist and t' exists:

$s_l = s + (t' - t) = t' - p, s_h = t'$

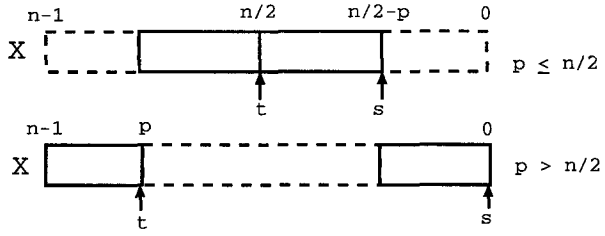


Figure 2.5: s and t

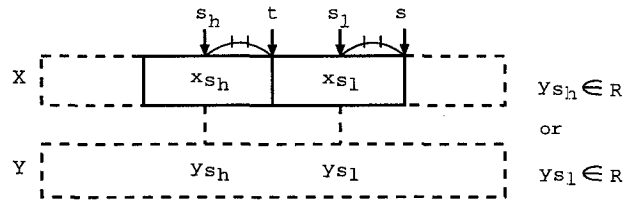


Figure 2.6: s_h and s_l

Case $s' - s < t' - t$, or t' does not exist and s' exists:
$s_l = s', s_h = t + (s' - s) = s' + p$
Case neither s' nor t' exists:
Neither s_l nor s_h is defined.

These s_l and s_h are illustrated in Figure 2.6. Intuitively, the pair (x_{s_h}, x_{s_l}) is the lowest member in $Args_p$ such that $y_{s_h} \in R$ or $y_{s_l} \in R$.

We define $Split'_p$ based on the split bits as follows:

$$Split'_p = \begin{cases} Split_p \setminus \{(x_{s_h-i}, x_{s_l-i}) | 1 \leq i \leq s_h - t\} \\ \quad \text{(when both } s_h \text{ and } s_l \text{ are defined)} \\ \emptyset \\ \quad \text{(when neither } s_h \text{ nor } s_l \text{ is defined)} \end{cases}$$

The following lemma holds.

Lemma 4 *The number of pairs* (x_i, y_i) *such that* $x_i \in R$ *and* $y_i \in L$ *is at least* $|Split_p \setminus Split'_p|$.

(Proof) For $(x_u, x_d) \in Split_p \setminus Split'_p$, both y_u and y_d belong to L by the definition of split bits. Exactly one of x_u and x_d is a member of R by the definition of $Split_p$. Hence, the lemma is proved. □

2.3 A Lower Bound on the Size of a *BMD Representing a Quotient Function

We consider two sets of inputs, $X = \{x_{n-1}, x_{n-2}, \ldots, x_0\}$ and $Y = \{y_{n-1}, y_{n-2}, \ldots, y_0\}$. Let f be a quotient function of $||X||$ divided by $||Y||$, whose set of inputs is $X \cup Y$.

We have the following theorem for a lower bound on the size of a *BMD representing a quotient function.

Theorem 2 *A lower bound on the number of nodes of any *BMD representing a quotient function* f *is* $\Omega(2^{n/24})$. □

To prove Theorem 2, we show the following two lemmas.

Lemma 5 *Given a variable ordering π and a left input assignment l, if the weight of l is more than or equal to one, then*

$$\sum_{\hat{l} \lesssim l} Sgn(\hat{l}) = 0.$$

(Proof) The number of $\hat{l}$'s such that $\hat{l} \lesssim l$ and whose weights are even is equal to that of $\hat{l}$'s such that $\hat{l} \lesssim l$ and whose weights are odd. Hence the lemma is proved. □

The following corollary is easily obtained.

Corollary 1 *Given a set of left input assignments S, if the number of left input assignments that belong to S and whose weights are even is equal to the number of left input assignments that belong to S and whose weights are odd, then*

$$\sum_{\hat{l} \in S} Sgn(\hat{l}) = 0.$$

□

In the following, $X(a)$ and $Y(a)$ denotes $||X||$ and $||Y||$ resulting from a complete assignment a respectively.

Lemma 6 *Given a variable ordering π, a left input assignment l whose weight is more than or equal to two and a right input assignment r, then*

$$\sum_{\hat{l} \lesssim l} X(\hat{l} \cdot r) Sgn(\hat{l}) = 0.$$

(Proof) For convenience, we describe the assigned value to x_i in an assignment $\hat{l} \cdot r$ as $x_i(\hat{l} \cdot r)$ instead of $\hat{l} \cdot r(x_i)$ defined before. Rewrite the given equation as follows:

$$\sum_{\hat{l} \lesssim l} X(\hat{l} \cdot r) Sgn(\hat{l}) = \sum_{\hat{l} \lesssim l} (2^{n-1} x_{n-1}(\hat{l} \cdot r) + \cdots + 2^0 x_0(\hat{l} \cdot r)) Sgn(\hat{l})$$

For each x_i $(0 \leq i \leq n-1)$,

Case $x_i \in R$:

Since $x_i(\hat{l} \cdot r)$ is a constant that does not depend on $\hat{l}$, the following equation holds by Lemma 5.

$$\sum_{\hat{l} \lesssim l} 2^i x_i(\hat{l} \cdot r) Sgn(\hat{l}) = 0$$

Case $x_i \in L$ and $x_i(l \cdot r) = 0$:

Since for all $\hat{l} \lesssim l$, $x_i(\hat{l} \cdot r) = 0$, the following equation holds.

$$\sum_{\hat{l} \lesssim l} 2^i x_i(\hat{l} \cdot r) Sgn(\hat{l}) = 0$$

Case $x_i \in L$ and $x_i(l \cdot r) = 1$:

$$\begin{aligned} & \sum_{\hat{l} \lesssim l} 2^i x_i(\hat{l} \cdot r) Sgn(\hat{l}) \\ = & \sum_{\hat{l} \in \{\hat{l} | \hat{l} \lesssim l, x_i(\hat{l}) = 1\}} 2^i x_i(\hat{l} \cdot r) Sgn(\hat{l}) + \sum_{\hat{l} \in \{\hat{l} | \hat{l} \lesssim l, x_i(\hat{l}) = 0\}} 2^i x_i(\hat{l} \cdot r) Sgn(\hat{l}) \\ = & \sum_{\hat{l} \in \{\hat{l} | \hat{l} \lesssim l, x_i(\hat{l}) = 1\}} 2^i x_i(\hat{l} \cdot r) Sgn(\hat{l}) + 0 \end{aligned}$$

Since the weight of l is more than or equal to two, the number of $\hat{l}$'s ($\hat{l} \lesssim l$) whose weights are even and which assign 1 to x_i is equal to the number of $\hat{l}$'s ($\hat{l} \lesssim l$) whose weights are odd and which assign 1 to x_i. Thus, by Corollary 1,

$$\sum_{\hat{l} \lesssim l} 2^i x_i(\hat{l} \cdot r) Sgn(\hat{l}) = \sum_{\hat{l} \in \{\hat{l} | \hat{l} \lesssim l, x_i(\hat{l}) = 1\}} 2^i x_i(\hat{l} \cdot r) Sgn(\hat{l}) = 0.$$

Hence,

$$\sum_{\hat{l} \lesssim l} X(\hat{l} \cdot r) Sgn(\hat{l}) = 0.$$

□

The following corollary is easily obtained from Lemma 6.

Corollary 2 *Given a variable ordering π, a set of left input assignments S and a right input assignment r for π, if there exist a variable ordering π', a left input assignment l' whose weight is more than or equal to two and a right input assignment r' for π' such that $\{\hat{l} \cdot r | \hat{l} \in S\} = \{\hat{l'} \cdot r' | \hat{l'} \lesssim l'\}$, then*

$$\sum_{\hat{l} \in S} X(\hat{l} \cdot r) Sgn(\hat{l}) = 0.$$

□

We now give the proof of Theorem 2

(Proof of Theorem 2) We assume that n is even by the same reason described in Section 2.2.4.

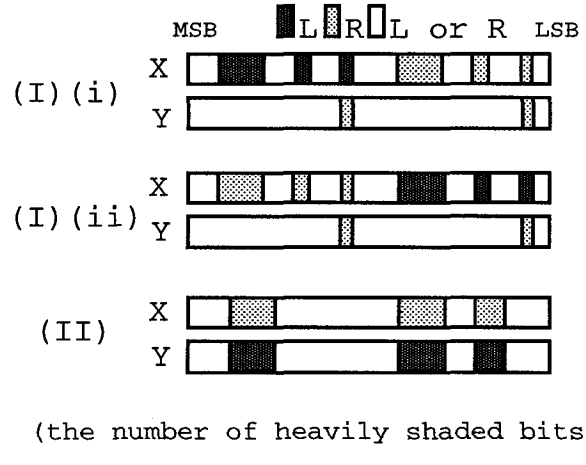


Figure 2.7: An example of the cases.

For any variable ordering π, we show that there exist a left partition and a fooling set whose cardinality is $\Omega(2^{n/24})$. Then we can conclude that the number of nodes of a *BMD representing f is $\Omega(2^{n/24})$ by Theorem 1.

We suppose that an arbitrary variable ordering π is given. Let L and R be a left partition and a right partition respectively such that $|X \cap L| = |X \cap R|$. The partitions L and R exist obviously. There exists some integer p $(1 \leq p \leq n-1)$ such that $|Split_p| \geq n/8$ by Lemma 3. We consider such a fixed p in the following. When an assignment a assigns 0's to all the variables, we describe $a = \bar{0}$.

We show a brief outline of the proof. We consider the following cases.

(I) $|Split'_p| \geq n/12$

- **(i)** $|Split'_p \cap (X_{UL} \times X_{DR})| \geq |Split'_p \cap (X_{UR} \times X_{DL})|$
- **(ii)** $|Split'_p \cap (X_{UL} \times X_{DR})| < |Split'_p \cap (X_{UR} \times X_{DL})|$

(II) $|Split'_p| < n/12$

While detailed explanations are described later, here we illustrate what each case is like (See Figure 2.7). In the case(I)(i), there exist a sufficient number of pairs (x_{i+p}, x_i) such that x_{i+p} is in the upper half of X and in L and x_i is in the lower half of X and in R. L and R are exchanged in the case(I)(ii). In the case(II), there exist a sufficient number of pairs (x_i, y_i) such that x_i is in $X \cap R$ and y_i is in $Y \cap L$. For each case, we show that there exists a set of left input assignments, and any two distinct left input assignments in the set satisfy the condition of the definition of fooling sets. Then we conclude that the set of left input assignments is a fooling set.

We can apply one of the following cases to obtain a fooling set whose cardinality is $\Omega(2^{n/24})$. We explain detail of each case in the following.

(I) $|Split'_p| \geq n/12$

(i) $|Split'_p \cap (X_{UL} \times X_{DR})| \geq |Split'_p \cap (X_{UR} \times X_{DL})|$

We define $Split''_p$ as $Split''_p = Split'_p \cap (X_{UL} \times X_{DR})$. Then $|Split''_p| \geq n/24$ since $|Split'_p| \geq n/12$ and $|Split'_p \cap (X_{UL} \times X_{DR})| \geq |Split'_p \cap (X_{UR} \times X_{DL})|$.

Note that, in this case, the high split bit s_h and the low split bit s_l exist. Let A be a set of all left input assignments satisfying the following conditions.

Conditions: For any variable x such that $x \in \{x_u | (x_u, x_d) \in Split''_p\} \triangleq B$,

[1] If x is the lowest member in B, that is, x has the minimum index in B, then the value 1 is assigned to x. We define this variable x to be x_q.

[2] The value 1 is assigned to at least one member of $B \setminus \{x_q\}$. That is, the value 1 is assigned to each of at least two members of B, including x_q.

[3] If $y_{s_h} \in L$, then the value 1 is assigned to y_{s_h}, where s_h is the high split bit.

[4] If $y_{s_l} \in L$, then the value 1 is assigned to y_{s_l}, where s_l is the low split bit.

[5] The value 0 is assigned to each of the variables that belongs to L other than mentioned above.

It is obvious that $|A| \geq 2^{n/24-1} - 1$. Intuitively these conditions mean that, for the bits belonging L in Figure 2.7(I)(i), the value 1 can be assigned only to the heavily shaded bits. We show that A is a fooling set in the following. If $|A| \leq 1$, it is obvious that A is a fooling set. We cope with the case that $|A| \geq 2$ in the following.

Let l and l' be any two distinct left input assignments that belong to A. We assume that $X(l \cdot \bar{0}) > X(l' \cdot \bar{0})$ without loss of generality. We define a right input assignment r as follows:

[1] If $y_{s_h} \in R$, then r assigns 1 to y_{s_h}.

[2] If $y_{s_l} \in R$, then r assigns 1 to y_{s_l}.

[3] If $i + s_h - s_l \neq q$ and the value 1 is assigned to $x_{i+s_h-s_l}$ in l, that is, $l(x_{i+s_h-s_l}) = 1$, then r assigns 1 to x_i. If $i + s_h - s_l = q$, in which case $l(x_{i+s_h-s_l}) = l(x_q) = 1$ obviously, then r assigns 0 to x_i.

X												
Variable	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
Partition	L	L	R	L	L	L	R	R	R	R	L	R
Assignment	1	1	0	0	0	1	1	1	0	0	0	0
Y												
Variable	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
Assignment	0	0	0	0	0	1	0	0	0	0	0	1

$Split''_p = \{(x_{11}, x_5), (x_{10}, x_4), (x_8, x_2), (x_6, x_0)\}$, $s_h = 6$, $s_l = 0$, $p = 6$, $q = 6$

Figure 2.8: An example of assignments.

[4] r assigns 0 to the variables that belong to R other than mentioned above.

An example of an assignment satisfying [1]–[4] is shown in Figure 2.8. We show that $f_l(r) \neq 0$ and $f_{l'}(r) = 0$ in the following. First, we cope with the case that both y_{s_h} and y_{s_l} belong to R (case(a)), and then, we cope with the case that either y_{s_h} or y_{s_l} belongs to L (case(b)).

(a) Both y_{s_h} and y_{s_l} belong to R.

First, we compute $f_l(r)$. We consider $\hat{l}$'s such that $\hat{l} \lesssim l$.

When $\hat{l} = l$, since $[X(\bar{0} \cdot r) + 2^{q-(s_h-s_l)}]/2^{s_l} = X(l \cdot \bar{0})/2^{s_h}$, the following equation holds.

$$\begin{aligned} & X(l \cdot r) + 2^{q-(s_h-s_l)} \\ = \; & X(l \cdot \bar{0}) + X(\bar{0} \cdot r) + 2^{q-(s_h-s_l)} \\ = \; & (2^{s_h} + 2^{s_l}) X(l \cdot \bar{0})/2^{s_h} \end{aligned}$$

Thus, $X(l \cdot r) = (2^{s_h} + 2^{s_l}) X(l \cdot \bar{0})/2^{s_h} - 2^{q-(s_h-s_l)}$.
Since $2^{s_h} + 2^{s_l} > 2^{q-(s_h-s_l)} > 0$ and $Y(l \cdot r) = 2^{s_h} + 2^{s_l}$, the quotient of $X(l \cdot r)$ divided by $Y(l \cdot r)$ is

$$f(l \cdot r) = X(l \cdot \bar{0})/2^{s_h} - 1. \tag{2.1}$$

For any left input assignment $\hat{l}$ such that $\hat{l} \lesssim l$ and $\hat{l} \neq l$, since x_q is the lowest in B, $X(\hat{l} \cdot \bar{0}) \leq X(l \cdot \bar{0}) - 2^q$.
Since $[X(l \cdot \bar{0}) - 2^q]/2^{s_h} = X(\bar{0} \cdot r)/2^{s_l}$, $0 \leq X(\hat{l} \cdot \bar{0})/2^{s_h} \leq X(\bar{0} \cdot r)/2^{s_l}$. Thus,

$$\begin{aligned} & X(\hat{l} \cdot r) - (2^{s_h} + 2^{s_l}) X(\hat{l} \cdot \bar{0})/2^{s_h} \\ = \; & [X(\hat{l} \cdot \bar{0}) + X(\bar{0} \cdot r)] - [X(\hat{l} \cdot \bar{0}) + \frac{2^{s_l}}{2^{s_h}} X(\hat{l} \cdot \bar{0})] \\ = \; & X(\bar{0} \cdot r) - \frac{2^{s_l}}{2^{s_h}} X(\hat{l} \cdot \bar{0}) \\ \begin{cases} \geq \\ \leq \end{cases} \; & \begin{matrix} X(\bar{0} \cdot r) - X(\bar{0} \cdot r) = 0 \\ X(\bar{0} \cdot r) < 2^{s_h} + 2^{s_l} \end{matrix} \end{aligned}$$

Thus,

$$2^{s_h} + 2^{s_l} > X(\hat{l} \cdot r) - (2^{s_h} + 2^{s_l})X(\hat{l} \cdot \bar{0})/2^{s_h} \geq 0.$$

Since $Y(\hat{l} \cdot r) = 2^{s_h} + 2^{s_l}$, the quotient of $X(\hat{l} \cdot r)$ divided by $Y(\hat{l} \cdot r)$ is

$$f(\hat{l} \cdot r) = X(\hat{l} \cdot \bar{0})/2^{s_h}. \tag{2.2}$$

Thus, by Lemma 2,

$$\begin{aligned} & f_l(r) \\ = \; & Sgn(l) \sum_{\hat{i} \lesssim l} Sgn(\hat{i}) f(\hat{i} \cdot r) \\ = \; & Sgn(l)[\sum_{\hat{i} \lesssim l, \hat{i} \neq l} Sgn(\hat{i}) X(\hat{i} \cdot \bar{0})/2^{s_h} + Sgn(l)(X(l \cdot \bar{0})/2^{s_h} - 1)] \\ = \; & Sgn(l)[\{\sum_{\hat{i} \lesssim l} Sgn(\hat{i}) X(\hat{i} \cdot \bar{0})/2^{s_h}\} - Sgn(l) \cdot 1] \end{aligned}$$

By Lemma 6, $f_l(r) = -Sgn(l)^2 = -1$.

Next, we compute $f_{l'}(r)$.

For any $\hat{l}' \lesssim l'$, since $X(l \cdot \bar{0}) > X(l' \cdot \bar{0})$,

$$X(\hat{l}' \cdot \bar{0}) \leq X(l \cdot \bar{0}) - 2^q.$$

Since $[X(l \cdot \bar{0}) - 2^q]/2^{s_h} = X(\bar{0} \cdot r)/2^{s_l}$,

$$0 \leq X(\hat{l}' \cdot \bar{0})/2^{s_h} \leq X(\bar{0} \cdot r)/2^{s_l}.$$

Note that this inequality holds even for $\hat{l}' = l'$. By the same reason used to obtain the equation (2.2), the quotient of $X(\hat{l}' \cdot r)$ divided by $Y(\hat{l}' \cdot r)$ is

$$f(\hat{l}' \cdot r) = X(\hat{l}' \cdot \bar{0})/2^{s_h}. \tag{2.3}$$

Thus by Lemma 2,

$$\begin{aligned} f_{l'}(r) &= Sgn(l') \sum_{\hat{i}' \lesssim l'} Sgn(\hat{i}') f(\hat{i}' \cdot r) \\ &= Sgn(l') \sum_{\hat{i}' \lesssim l'} Sgn(\hat{i}') X(\hat{i}' \cdot \bar{0})/2^{s_h}. \end{aligned}$$

By Lemma 6, $f_{l'}(r) = 0$.

Hence, $f_l(r) \neq 0$ and $f_{l'}(r) = 0$. Therefore A is a fooling set.

(b) Either y_{s_h} or y_{s_l} belongs to L.

Note that, in this case, either y_{s_h} or y_{s_l} belongs to R by the definition of split bits (s_h and s_l). When $y_{s_h} \in L$, we define sets of left input assignments S_l^1 and S_l^0 as

$$S_l^1 = \{\hat{l} | \hat{l} \lesssim l, \hat{l}(y_{s_h}) = 1\} \text{ and } S_l^0 = \{\hat{l} | \hat{l} \lesssim l, \hat{l}(y_{s_h}) = 0$$

When $y_{s_l} \in L$, we define S_l^1 and S_l^0 similarly, replacing y_{s_h} with y_{s_l}.

First, we compute $f_l(r)$.

By Lemma 2,

$$\begin{aligned} f_l(r) &= Sgn(l) \sum_{\hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l} \cdot r) \\ &= Sgn(l)[\sum_{\hat{l} \in S_l^1} Sgn(\hat{l}) f(\hat{l} \cdot r) + \sum_{\hat{l} \in S_l^0} Sgn(\hat{l}) f(\hat{l} \cdot r)]. \end{aligned}$$

In order to obtain $f(\hat{l} \cdot r)$ for $\hat{l} \in S_l^1$, we can apply the same method that is used to obtain the equations (2.1) and (2.2), treating y_{s_h} or y_{s_l} as fixed, in other words, as if it were in R. Then $f(l \cdot r) = X(l \cdot \bar{0})/2^{s_h} - 1$ for l, and $f(\hat{l} \cdot r) = X(\hat{l} \cdot \bar{0})/2^{s_h}$ for $\hat{l} \in S_l^1$ and $\hat{l} \neq l$. Thus,

$$\begin{aligned} f_l(r) &= Sgn(l)[\{\sum_{\hat{l} \in S_l^1} Sgn(\hat{l}) X(\hat{l} \cdot \bar{0})/2^{s_h}\} - Sgn(l) \cdot 1 + \sum_{\hat{l} \in S_l^0} Sgn(\hat{l}) f(\hat{l} \cdot r)] \\ &= Sgn(l)[\{\sum_{\hat{l} \in S_l^1} Sgn(\hat{l}) X(\hat{l} \cdot \bar{0})/2^{s_h}\} - Sgn(l) \cdot 1 \\ &\quad + \begin{cases} \sum_{\hat{l} \in S_l^0} Sgn(\hat{l}) X(\hat{l} \cdot \bar{0})/2^{s_h}] \\ (y_{s_h} \in R, \text{ that is, } Y(\hat{l} \cdot r) = 2^{s_h}) \\ \sum_{\hat{l} \in S_l^0} Sgn(\hat{l}) X(\hat{l} \cdot r)/2^{s_l}] \\ (y_{s_l} \in R, \text{ that is, } Y(\hat{l} \cdot r) = 2^{s_l}). \end{cases} \end{aligned}$$

Thus, by Corollary 2, $f_l(r) = -Sgn(l)^2 = -1$.

Similarly, we can compute $f_{l'}(r)$.

$$\begin{aligned} f_{l'}(r) &= Sgn(l') \sum_{\hat{l'} \lesssim l'} Sgn(\hat{l'}) f(\hat{l'} \cdot r) \\ &= Sgn(l')[\sum_{\hat{l'} \in S_{l'}^1} Sgn(\hat{l'}) f(\hat{l'} \cdot r) + \sum_{\hat{l'} \in S_{l'}^0} Sgn(\hat{l'}) f(\hat{l'} \cdot r)]. \end{aligned}$$

In order to obtain $f(\hat{l'} \cdot r)$ for $\hat{l'} \in S_{l'}^1$, we can apply the same method that is used to obtain the equation (2.3), treating y_{s_h} or y_{s_l} as fixed, in other words, as if it were in R. Thus, $f(\hat{l'} \cdot r) = X(\hat{l'} \cdot \bar{0})/2^{s_h}$ for $\hat{l'} \in S_{l'}^1$. $\sum_{\hat{l'} \in S_{l'}^0} Sgn(\hat{l'}) f(\hat{l'} \cdot r) = 0$ as shown in the above. By Corollary 2, $f_{l'}(r) = 0$. Hence, $f_l(r) \neq 0$ and $f_{l'}(r) = 0$.

Therefore, A is a fooling set.

(ii) $|Split'_p \cap (X_{UL} \times X_{DR})| < |Split'_p \cap (X_{UR} \times X_{DL})|$

We define $Split''_p$ as $Split''_p = Split'_p \cap (X_{UR} \times X_{DL})$. Then $|Split''_p| \geq n/24$ since $|Split'_p| \geq n/12$ and $|Split'_p \cap (X_{UL} \times X_{DR})| < |Split'_p \cap (X_{UR} \times X_{DL})|$.

Note that, in this case, the high split bit s_h and the low split bit s_l exist. Let A be a set of all left input assignments satisfying the following conditions.

Conditions: For any variable x such that $x \in \{x_d | (x_u, x_d) \in Split''_p\} \triangleq B$,

[1] If x is the lowest member in B, that is, x has the minimum index in B, then the value 1 is assigned to x. We define this variable x to be x_q.

[2] The value 1 is assigned to at least one member of $B \setminus \{x_q\}$. That is, the value 1 is assigned to each of at least two members of B, including x_q.

[3] If $y_{s_h} \in L$, then the value 1 is assigned to y_{s_h}, where s_h is the high split bit.

[4] If $y_{s_l} \in L$, then the value 1 is assigned to y_{s_l}, where s_l is the low split bit.

[5] The value 0 is assigned to each of the variables that belongs to L other than mentioned above.

It is obvious that $|A| \geq 2^{n/24-1} - 1$. Intuitively these conditions mean that, for the bits belonging to L in Figure 2.7(I)(ii), the value 1 can be assigned only to the heavily shaded bits. We show that A is a fooling set in the following. If $|A| \leq 1$, it is obvious that A is a fooling set. We cope with the case that $|A| \geq 2$ in the following.

Let l and l' be any two distinct left input assignments that belong to A. We assume that $X(l \cdot \bar{0}) > X(l' \cdot \bar{0})$ without loss of generality. We define a right input assignment r as follows:

[1] If $y_{s_h} \in R$, then r assigns 1 to y_{s_h}.

[2] If $y_{s_l} \in R$, then r assigns 1 to y_{s_l}.

[3] If the value 1 is assigned to x_i in l, that is, $l(x_i) = 1$, then r assigns 1 to $x_{i+s_h-s_l}$.

[4] r assigns 0 to the variables that belong to R other than mentioned above.

An example of an assignment satisfying [1]–[4] is shown in Figure 2.9. We show that $f_l(r) \neq 0$ and $f_{l'}(r) = 0$ in the following. First, we cope with the case that both y_{s_h} and y_{s_l} belong to R (case(a)), and then, we cope with the case that either y_{s_h} or y_{s_l} belongs to L (case(b)).

X												
Variable	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
Partition	R	R	L	R	R	R	L	L	R	L	R	L
Assignment	1	1	0	0	0	1	1	1	0	0	0	1
Y												
Variable	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
Assignment	0	0	0	0	0	1	0	0	0	0	0	1

$Split''_p = \{(x_{11}, x_5), (x_{10}, x_4), (x_8, x_2), (x_6, x_0)\}$, $s_h = 6$, $s_l = 0$, $p = 6$, $q = 0$

Figure 2.9: An example of an assignment.

(a) Both y_{s_h} and y_{s_l} belong to R.

First, we compute $f_l(r)$. We consider $\hat{l}$'s such that $\hat{l} \lesssim l$.

When $\hat{l} = l$, $X(l \cdot \bar{0})/2^{s_l} = X(\bar{0} \cdot r)/2^{s_h}$.

Thus, the following equation holds.

$$\begin{aligned} X(l \cdot r) &= X(l \cdot \bar{0}) + X(\bar{0} \cdot r) \\ &= \frac{2^{s_l}}{2^{s_h}} X(\bar{0} \cdot r) + X(\bar{0} \cdot r) \\ &= (2^{s_h} + 2^{s_l}) X(\bar{0} \cdot r)/2^{s_h} \end{aligned}$$

Since $Y(l \cdot r) = 2^{s_h} + 2^{s_l}$, the quotient of $X(l \cdot r)$ divided by $Y(l \cdot r)$ is

$$f(l \cdot r) = X(\bar{0} \cdot r)/2^{s_h}. \tag{2.4}$$

For any left input assignment $\hat{l}$ such that $\hat{l} \lesssim l$ and $\hat{l} \neq l$, $X(\hat{l} \cdot \bar{0}) < X(l \cdot \bar{0})$. Thus, $X(\hat{l} \cdot \bar{0})/2^{s_l} < X(\bar{0} \cdot r)/2^{s_h} < (2^{s_h} + 2^{s_l})/2^{s_l}$.

Thus,

$$\begin{aligned} & X(\hat{l} \cdot r) - (2^{s_h} + 2^{s_l}) X(\bar{0} \cdot r)/2^{s_h} \\ = \; & [X(\hat{l} \cdot \bar{0}) + X(\bar{0} \cdot r)] - [X(\bar{0} \cdot r) + \frac{2^{s_l}}{2^{s_h}} X(\bar{0} \cdot r)] \\ = \; & X(\hat{l} \cdot \bar{0}) - \frac{2^{s_l}}{2^{s_h}} X(\bar{0} \cdot r) \\ \begin{cases} < \\ > \end{cases} \; & \begin{matrix} X(\hat{l} \cdot \bar{0}) - X(\hat{l} \cdot \bar{0}) = 0 \\ X(\hat{l} \cdot \bar{0}) - (2^{s_h} + 2^{s_l}) \geq -(2^{s_h} + 2^{s_l}). \end{matrix} \end{aligned}$$

Thus,

$$-(2^{s_h} + 2^{s_l}) \leq X(\hat{l} \cdot r) - (2^{s_h} + 2^{s_l}) X(\bar{0} \cdot r)/2^{s_h} < 0.$$

Since $Y(\hat{l} \cdot r) = 2^{s_h} + 2^{s_l}$, the quotient of $X(\hat{l} \cdot r)$ divided by $Y(\hat{l} \cdot r)$ is

$$f(\hat{l} \cdot r) = X(\bar{0} \cdot r)/2^{s_h} - 1. \tag{2.5}$$

Thus, by Lemma 2,

$$\begin{aligned} f_l(r) &= Sgn(l)\sum_{\hat{l}\lesssim l} Sgn(\hat{l})f(\hat{l}\cdot r) \\ &= Sgn(l)[\{\sum_{\hat{l}\lesssim l,\hat{l}\neq l} Sgn(\hat{l})(\frac{X(\bar{0}\cdot r)}{2^{s_h}}-1)\}+Sgn(l)X(\bar{0}\cdot r)/2^{s_h}] \\ &= Sgn(l)[\{\sum_{\hat{l}\lesssim l} Sgn(\hat{l})(X(\bar{0}\cdot r)/2^{s_h}-1)\}+Sgn(l)\cdot 1]. \end{aligned}$$

Since $X(\bar{0}\cdot r)/2^{s_h}$ is a constant, $f_l(r) = Sgn(l)^2 = 1$ by Lemma 5.

Next, we compute $f_{l'}(r)$.

For any $\hat{l}' \overset{<}{\sim} l'$, since $X(l'\cdot\bar{0}) < X(l\cdot\bar{0})$,

$$X(\hat{l}'\cdot\bar{0}) < X(l\cdot\bar{0}).$$

Thus,

$$X(\hat{l}'\cdot\bar{0})/2^{s_l} < X(\bar{0}\cdot r)/2^{s_h} < (2^{s_h}+2^{s_l})/2^{s_l}.$$

Note that this inequality holds even for $\hat{l}' = l'$. By the same reason used to obtain the equation (2.5), the quotient of $X(\hat{l}'\cdot r)$ divided by $Y(\hat{l}'\cdot r)$ is

$$f(\hat{l}'\cdot r) = X(\bar{0}\cdot r)/2^{s_h} - 1 \tag{2.6}$$

Thus by Lemma 2,

$$\begin{aligned} f_{l'}(r) &= Sgn(l')\sum_{\hat{l}'\lesssim l'} Sgn(\hat{l}')f(\hat{l}'\cdot r) \\ &= Sgn(l')\sum_{\hat{l}'\lesssim l'} Sgn(\hat{l}')(X(\bar{0}\cdot r)/2^{s_h}-1). \end{aligned}$$

Since $X(\bar{0}\cdot r)/2^{s_h}$ is a constant, $f_{l'}(r) = 0$ by Lemma 5.

Hence, $f_l(r) \neq 0$ and $f_{l'}(r) = 0$. Therefore A is a fooling set.

(b) Either y_{s_h} or y_{s_l} belongs to L.

Note that, in this case, either y_{s_h} or y_{s_l} belongs to R by the definition of split bits (s_h and s_l). We define S_l^1 and S_l^0 in the same way as in (i). Then, by Lemma 2,

$$\begin{aligned} f_l(r) &= Sgn(l)\sum_{\hat{l}\lesssim l} Sgn(\hat{l})f(\hat{l}\cdot r) \\ &= Sgn(l)[\sum_{\hat{l}\in S_l^1} Sgn(\hat{l})f(\hat{l}\cdot r)+\sum_{\hat{l}\in S_l^0} Sgn(\hat{l})f(\hat{l}\cdot r)]. \end{aligned}$$

In order to obtain $f(\hat{l}\cdot r)$ for $\hat{l} \in S_l^1$, we can apply the same method that is used to obtain the equations (2.4) and (2.5), treating y_{s_h} or y_{s_l} as fixed, in other words, as if it were in R. Then $f(l\cdot r) = X(\bar{0}\cdot r)/2^{s_h}$ for l, and $f(\hat{l}\cdot r) = X(\bar{0}\cdot r)/2^{s_h} - 1$ for $\hat{l} \in S_l^1$ and $\hat{l} \neq l$. Thus,

$$
\begin{aligned}
& f_l(r) \\
= \; & Sgn(l)[\{\sum_{\hat{l}\in S_l^1} Sgn(\hat{l})(X(\bar{0}\cdot r)/2^{s_h} - 1)\} + Sgn(l)\cdot 1 + \sum_{\hat{l}\in S_l^0} Sgn(\hat{l}) f(\hat{l}\cdot r)] \\
= \; & Sgn(l)[\{\sum_{\hat{l}\in S_l^1} Sgn(\hat{l})(X(\bar{0}\cdot r)/2^{s_h} - 1)\} + Sgn(l)\cdot 1 \\
& + \begin{cases} \sum_{\hat{l}\in S_l^0} Sgn(\hat{l}) X(\bar{0}\cdot r)/2^{s_h}] \\ (y_{s_h} \in R, \text{ that is, } Y(\hat{l}\cdot r) = 2^{s_h}) \\ \sum_{\hat{l}\in S_l^0} Sgn(\hat{l}) X(\hat{l}\cdot r)/2^{s_l}] \\ (y_{s_l} \in R, \text{ that is, } Y(\hat{l}\cdot r) = 2^{s_l}) \end{cases}
\end{aligned}
$$

Thus, by Corollary 1 and Corollary 2, $f_l(r) = Sgn(l)^2 = 1$.

Similarly, we can compute $f_{l'}(r)$.

$$
\begin{aligned}
f_{l'}(r) &= Sgn(l') \sum_{\hat{l'}\lesssim_{l'}} Sgn(\hat{l'}) f(\hat{l'}\cdot r) \\
&= Sgn(l')[\sum_{\hat{l'}\in S_{l'}^1} Sgn(\hat{l'}) f(\hat{l'}\cdot r) + \sum_{\hat{l'}\in S_{l'}^0} Sgn(\hat{l'}) f(\hat{l'}\cdot r)].
\end{aligned}
$$

In order to obtain $f(\hat{l'}\cdot r)$ for $\hat{l'} \in S_{l'}^1$, we can apply the same method that is used to obtain the equation (2.6), treating y_{s_h} or y_{s_l} as fixed, in other words, as if it were in R. Thus, for $\hat{l'} \in S_{l'}^1$, $f(\hat{l'}\cdot r) = X(\bar{0}\cdot r)/2^{s_h} - 1$. $\sum_{\hat{l'}\in S_{l'}^0} Sgn(\hat{l'}) f(\hat{l'}\cdot r) = 0$ as shown in the above. Thus $f_{l'}(r) = 0$ by Corollary 1.

Hence, $f_l(r) \neq 0$ and $f_{l'}(r) = 0$. Therefore A is a fooling set.

(II) $|Split'_p| < n/12$

There exist more than $n/8 - n/12 = n/24$ pairs of (x_i, y_i) such that $x_i \in R$ and $y_i \in L$ by Lemma 4.

If there exists no pair of $(x_i \in R, y_i \in R)$, we can reconstruct a left partition L' and a right partition R' such that there exists exactly one pair of $(x_i \in R', y_i \in R')$, and there exist more than $n/24 - 1$ pairs $(x_i \in R', y_i \in L')$, by decreasing the cardinality of L and increasing the cardinality of R, that is, $|L' \cap X| < |R' \cap X|$. In this case, we can use L' and R' as L and R respectively.

Let (x_w, y_w) be a pair such that $x_w \in R$ and $y_w \in R$. Let A be a set of all left input assignments satisfying the following conditions.

Variable	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
Partition	L	R	R	R	R	R	L	R
Assignment	0	1	0	0	1	0	0	0

Variable	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
Partition	L	R	L	L	L	L	R	L
lr	0	1	0	1	0	1	0	1
$l'r$	0	1	1	0	1	1	0	1
$l''r$	0	1	0	0	0	1	0	1

$w = 6$, $q = 3$

Figure 2.10: An example of assignments.

Conditions:

[1] If $x_i \in R$ and $y_i \in L$, then any value is assigned to y_i.

[2] The value 0 is assigned to each of the other variables that belong to L.

It is obvious that $|A| > 2^{n/24-1}$. Intuitively these conditions mean that, for the bits belonging to L in Figure 2.7(II), the value 1 can be assigned only to the heavily shaded bits. We show that A is a fooling set in the following. If $|A| \leq 1$, it is obvious that A is a fooling set. We cope with the case that $|A| \geq 2$ in the following.

Let l and l' be any two distinct left input assignments that belong to A. We show that there exist right input assignments r and r' such that $f_l(r)f_{l'}(r') \neq f_{l'}(r)f_l(r')$. We assume that q is the minimum index of $y_i \in Y$ such that $y_i \in L$ and $l(y_i) \neq l'(y_i)$. We assume that $l(y_q) = 0$ and $l'(y_q) = 1$ without loss of generality. We define a left input assignment l'' as follows:

[1] For $y_i \in Y \cap L$ $(i < q)$, l'' assigns the same value as l. It is also the same value as l' as a result.

[2] l'' assigns 0 to the other variables that belong to L.

We define a right input assignment r as follows:

[1] r assigns 1 to each variable x_w, y_w and x_q.

[2] r assigns 0 to the other variables that belong to R.

An example of an assignment satisfying [1] and [2] is shown in Figure 2.10. We define SL_l and SS_l as

$$\begin{aligned} SL_l &= \{\hat{l} \lesssim l | \exists y_i \in L,\ i \geq q,\ \hat{l}(y_i) = 1\} \\ \text{and} & \\ SS_l &= \{\hat{l} \lesssim l | \forall y_i \in L,\ i \geq q \Rightarrow \hat{l}(y_i) = 0\}. \end{aligned}$$

Similarly, we define $SL_{l'}$ and $SS_{l'}$. Note that $\{\hat{l} | \hat{l} \lesssim l\} = SL_l \cup SS_l$ and $\{\hat{l'} | \hat{l'} \lesssim l'\} = SL_{l'} \cup SS_{l'}$. Then we obtain the following.

$$\begin{aligned} &\forall \hat{l} \in SL_l \quad f(\hat{l} \cdot r) = 0 \\ &\forall \hat{l'} \in SL_{l'} \quad f(\hat{l'} \cdot r) = \begin{cases} 1 & (Y(\hat{l'} \cdot r) = 2^w + 2^q) \\ 0 & (\text{otherwise}) \end{cases} \end{aligned}$$

Note that $SS_l = SS_{l'} = \{\hat{l} | \hat{l} \lesssim l''\}$. By Lemma 2,

$$\begin{aligned} & f_l(r) \\ = \ & Sgn(l) \sum_{\hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l} \cdot r) \\ = \ & Sgn(l)[\sum_{\hat{l} \in SL_l} Sgn(\hat{l}) f(\hat{l} \cdot r) + \sum_{\hat{l} \in SS_l} Sgn(\hat{l}) f(\hat{l} \cdot r)] \\ = \ & Sgn(l)[\sum_{\hat{l} \in SL_l} 0 + \sum_{\hat{l} \lesssim l''} Sgn(\hat{l}) f(\hat{l} \cdot r)] \\ = \ & Sgn(l) \sum_{\hat{l} \lesssim l''} Sgn(\hat{l}) f(\hat{l} \cdot r) \end{aligned}$$

Note that, when $Y(\hat{l'} \cdot r) = 2^w + 2^q$, the weight of $\hat{l'}$ is 1. Thus,

$$\begin{aligned} f_{l'}(r) &= Sgn(l') \sum_{\hat{l'} \lesssim l'} Sgn(\hat{l'}) f(\hat{l'} \cdot r) \\ &= Sgn(l')[\sum_{\hat{l'} \in SL_{l'}} Sgn(\hat{l'}) f(\hat{l'} \cdot r) + \sum_{\hat{l'} \in SS_{l'}} Sgn(\hat{l'}) f(\hat{l'} \cdot r)] \\ &= Sgn(l')[-1 + \sum_{\hat{l} \lesssim l''} Sgn(\hat{l}) f(\hat{l} \cdot r)]. \end{aligned}$$

Next, we define a right input assignment r' as

[1] r' assigns 1 to each variable x_w and y_w.

[2] r' assigns 0 to the other variables that belong to R.

Variable	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
Partition	L	R	R	R	R	R	L	R
Assignment	0	1	0	0	0	0	0	0

Variable	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
Partition	L	R	L	L	L	L	R	L
lr'	0	1	0	1	0	1	0	1
$l'r'$	0	1	1	0	1	1	0	1
$l''r'$	0	1	0	0	0	1	0	1

$w = 6$, $q = 3$

Figure 2.11: An example of assignments.

An example of an assignment satisfying [1] and [2] is shown in Figure 2.11. Then, for any $\hat{l}$ such that $\hat{l} \lesssim l$ or $\hat{l} \lesssim l'$,

Case $Y(\hat{l} \cdot r') = 2^w$, that is, $\hat{l} = \bar{0}$:

$f(\hat{l} \cdot r') = 1$

Case $Y(\hat{l} \cdot r') \neq 2^w$:

$f(\hat{l} \cdot r') = 0$

Thus by Lemma 2,

$$f_l(r') = Sgn(l) \sum_{\hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l} \cdot r') = Sgn(l) \cdot 1.$$

Similarly, $f_{l'}(r') = Sgn(l') \cdot 1$.

Hence, $f_l(r) f_{l'}(r') \neq f_{l'}(r) f_l(r')$. Therefore, A is a fooling set.

We conclude that a lower bound on the number of nodes of a *BMD representing a quotient function f is $\Omega(2^{n/24})$ from (I), (II) and Theorem 1. □

2.4 A Lower Bound on the Size of a *BMD Representing a Remainder Function

An exponential lower bound for a remainder function can also be proved by the similar method used in Section 2.3. We show that a lower bound on the size of a *BMD representing a remainder function is $\Omega(2^{n/24})$.

Let f be a remainder function of $||X||$ divided by $||Y||$, whose set of inputs is $X \cup Y$. We have the following theorem for a lower bound on the size of a *BMD representing f.

Theorem 3 *A lower bound on the number of nodes of any *BMD representing a remainder function f is $\Omega(2^{\frac{n}{24}})$.* □

To prove Theorem 3, we show the following lemmas.

Lemma 7 *The remainder of* 2^{i+p} *divided by* 2^i+1 *is* $(2^i+1)-2^p$, *where* $0 \leq p \leq i$.

(Proof) We prove Lemma 7 by induction.

When $p = 0$, the lemma holds obviously.

We suppose that the lemma holds for $p = k$. We consider the case that $p = k + 1$.

The following equation holds by the hypothesis of induction.

$$2^{i+k} = (2^i + 1)Q_k + (2^i + 1) - 2^k,$$

where Q_k is the quotient of 2^{i+k} divided by $2^i + 1$.

Let Q_{k+1} and R_{k+1} be the quotient and the remainder of 2^{i+k+1} divided by $2^i + 1$ respectively. Then,

$$\begin{aligned}
2^{i+k+1} &= (2^i + 1)Q_{k+1} + R_{k+1} \\
2(2^i + 1)Q_k + 2(2^i + 1) - 2^{k+1} &= (2^i + 1)Q_{k+1} + R_{k+1} \\
(2^i + 1)(2Q_k - Q_{k+1} + 2) - 2^{k+1} &= R_{k+1} \\
R_{k+1} &= (2^i + 1)Q - 2^{k+1} \\
&\quad (\text{where } Q = (2Q_k - Q_{k+1} + 2)) \\
\text{by } 0 \leq R_{k+1} < 2^i + 1 & \\
R_{k+1} &= (2^i + 1) - 2^{k+1}.
\end{aligned}$$

□

The following corollary is easily obtained from Lemma 7.

Corollary 3 *The remainder of* $2^{i+p_m} + 2^{i+p_{m-1}} + \cdots + 2^{i+p_1}$ *divided by* $2^i + 1$ *is* $(2^i + 1) - (2^{p_m} + 2^{p_{m-1}} + \cdots + 2^{p_1})$, *where* $p_k \neq p_l$ *for* $k \neq l$ *and* $0 \leq p_k < i$ *for* $1 \leq k \leq m$. □

The following corollary is easily obtained from Corollary 3.

Corollary 4 *The remainder of* $2^{i+p_m} + 2^{i+p_{m-1}} + \cdots + 2^{i+p_1}$ *divided by* $2^i + 2^j$ *is*

$$\begin{aligned}
& 2^j\{(2^{i-j} + 1) - (2^{p_m} + 2^{p_{m-1}} + \cdots + 2^{p_1})\} \\
= \; & (2^i + 2^j) - 2^j(2^p_m + 2^p_{m-1} + \cdots + 2^{i+p_1}),
\end{aligned}$$

where $p_k \neq p_l$ *for* $k \neq l$ *and* $0 \leq p_k < i - j$ *for* $1 \leq k \leq m$. □

We introduce two notations for left input assignments. A notation $l|_{x=a}$ denotes the left input assignment that is obtained from the left input assignment l restricting the assignment to x to be a. For two distinct left input assignments l, l', and a right input assignment r, we define a critical left input assignment l_c to be the left input assignment satisfying the following conditions.

Conditions:

[1] $l_c \lesssim l$ and $l_c \not\sim l'$ or $l_c \lesssim l'$ and $l_c \not\sim l$. We assume that $l_c \lesssim l$ and $l_c \not\sim l'$ for the rest of this definition. For the case of $l_c \lesssim l'$ and $l_c \not\sim l$, the following conditions are similarly defined.

[2] $f(l_c \cdot r) \neq f(\hat{l'} \cdot r)$ for $\hat{l'} \lesssim l'$.

[3] If $f(\hat{l} \cdot r) > f(l_c \cdot r)$ (resp. $f(\hat{l'} \cdot r) > f(l_c \cdot r)$) for $\hat{l} \lesssim l$ (resp. $\hat{l'} \lesssim l'$), then there exists $\hat{l'} \lesssim l'$ (resp. $\hat{l} \lesssim l$) and $f(\hat{l} \cdot r) = f(\hat{l'} \cdot r)$ (resp. $f(\hat{l'} \cdot r) = f(\hat{l} \cdot r)$).

Intuitively, $f(l_c \cdot r)$ for $l_c \lesssim l$ is the maximum of $f(\hat{l} \cdot r)$ where $\hat{l} \lesssim l$ and $f(\hat{l} \cdot r) \neq f(\hat{l'} \cdot r)$ for all $\hat{l'} \lesssim l'$. Note that if $\{a | a = f(\hat{l} \cdot r), \hat{l} \lesssim l\} \neq \{a | a = f(\hat{l'} \cdot r), \hat{l'} \lesssim l'\}$, a critical left input assignment exists.

We have the following lemma for fooling sets.

Lemma 8 *If the set of left input assignments A satisfies the following conditions, then A is a fooling set.*

Conditions:

[1] $\exists r', \forall l, l' \in A, \exists r, f(l_c \cdot r) = 0$ and $r' \lesssim r$, where l_c is defined for l, l' and r'.

[2] $\forall l \in A, \sum_{\hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l} \cdot r') = 0$, where r' is the right input assignment in condition [1].

[3] $\exists q, \forall l \in A$, the lowest bit to which the value 1 is assigned is x_q.

[4] $\forall l, l' \in A$, if $\hat{l} \neq l_c|_{x_q=0}$ and $\hat{l} \neq l_c|_{x_q=1}$ for $\hat{l} \lesssim l$ or $\hat{l} \lesssim l'$, then $f(\hat{l}|_{x_q=0} \cdot r') > f(l_c \cdot r')$ if and only if $f(\hat{l}|_{x_q=1} \cdot r') > f(l_c \cdot r')$, where r' is the right input assignment in condition [1].

[5] $\exists t, s, \forall l \in A, Y(l \cdot r) = Y(l \cdot r') = 2^t + 2^s$, where r and r' are the right input assignments in condition [1].

[6] $\forall l, l' \in A$, if $f(\hat{l} \cdot r') = f(\hat{l'} \cdot r')$ for $\hat{l} \lesssim l$ and $\hat{l'} \lesssim l'$, then $f(\hat{l}|_{x_q=0} \cdot r') = f(\hat{l'}|_{x_q=0} \cdot r')$ and $f(\hat{l}|_{x_q=1} \cdot r') = f(\hat{l'}|_{x_q=1} \cdot r')$, where r' is the right input assignment in condition [1], and q is the integer in condition [3].

□

To prove Lemma 8, we introduce the following lemma.

Lemma 9 *We suppose that the set of left input assignments A satisfies the conditions in Lemma 8. For left input assignments $l, l' \in A$, $f(l_c|_{x_q=0} \cdot r') \leq f(l_c \cdot r')$ and $f(l_c|_{x_q=1} \cdot r') \leq f(l_c \cdot r')$, where r' and x_q are defined in the conditions in Lemma 8.*

(Proof) We assume that $l_c \lesssim l$ and $l_c \not\lesssim l'$ without loss of generality, and also assume that either $f(l_c|_{x_q=0} \cdot r') > f(l_c \cdot r')$ or $f(l_c|_{x_q=1} \cdot r') > f(l_c \cdot r')$. Then there exists $\hat{l}' \lesssim l'$ such that $f(\hat{l}' \cdot r') = f(l_c|_{x_q=0} \cdot r')$ or $f(\hat{l}' \cdot r') = f(l_c|_{x_q=1} \cdot r')$ by the definition of a critical left input assignment. Thus $f(\hat{l}'|_{x_q=0} \cdot r') = f(l_c \cdot r')$ or $f(\hat{l}'|_{x_q=1} \cdot r') = f(l_c \cdot r')$ by condition [6] in Lemma 8. Note that $\hat{l}'|_{x_q=0} \lesssim l'$ and $\hat{l}'|_{x_q=1} \lesssim l'$ by $\hat{l}' \lesssim l'$ and condition [3] in Lemma 8. Thus there exists $\hat{l}'' \lesssim l'$, $f(\hat{l}'' \cdot r') = f(l_c \cdot r')$, contradicting the definition of a critical left input assignment. □

We now give the proof of Lemma 8.

(Proof) For $l, l' \in A$, we assume that $l_c \lesssim l$ and $l_c \not\lesssim l'$ without loss of generality, and let r and r' be the right input assignments in condition [1] in Lemma 8, and let t and s be the integers in condition [5] in Lemma 8. We show that $f_l(r) \neq 0$ and $f_{l'}(r) = 0$.

We define $S^L_{l'}$ and $S^S_{l'}$ as

$$\begin{aligned} S^L_{l'} &= \{\hat{l}' | \hat{l}' \lesssim l',\ f(\hat{l}' \cdot r') > f(l_c \cdot r')\} \\ S^S_{l'} &= \{\hat{l}' | \hat{l}' \lesssim l',\ f(\hat{l}' \cdot r') < f(l_c \cdot r')\}. \end{aligned}$$

Note that the number of left input assignments that belong to $S^L_{l'}$ (resp. $S^S_{l'}$) and whose weights are even is equal to the number of left input assignments that belong to $S^L_{l'}$ (resp. $S^S_{l'}$) and whose weights are odd by condition [4] in Lemma 8, and also note that for $W = (2^t + 2^s) - f(l_c \cdot r')$, $f(\hat{l} \cdot r) = f(\hat{l} \cdot r') + W - (2^t + 2^s)$ for $\hat{l} \in S^L_{l'}$ and $f(\hat{l} \cdot r) = f(\hat{l} \cdot r') + W$ for $\hat{l} \in S^S_{l'}$ since $f(l_c \cdot r') + W = (2^t + 2^s)$ and by condition [5] in Lemma 8. To compute $f_{l'}(r)$ by Lemma 2, we expand $f_{l'}(r)$ into two terms as follows:

$$\begin{aligned} f_{l'}(r) &= Sgn(l') \sum_{\hat{l}' \lesssim l'} Sgn(\hat{l}') f(\hat{l}' \cdot r) \\ &= Sgn(l') \left[\sum_{\hat{l}' \in S^L_{l'}} Sgn(\hat{l}') f(\hat{l}' \cdot r) + \sum_{\hat{l}' \in S^S_{l'}} Sgn(\hat{l}') f(\hat{l}' \cdot r) \right] \end{aligned}$$

For the first term,

$$\begin{aligned}\sum_{\hat{l'}\in S_{l'}^L} Sgn(\hat{l'})f(\hat{l'}\cdot r) &= \sum_{\hat{l'}\in S_{l'}^L} Sgn(\hat{l'})\Big(f(\hat{l'}\cdot r') + W - (2^t+2^s)\Big)\\ &= \sum_{\hat{l'}\in S_{l'}^L} Sgn(\hat{l'})f(\hat{l'}\cdot r').\end{aligned}$$

(since $W-(2^t+2^s)$ is a constant, and by Corollary 1)

For the last term,

$$\begin{aligned}\sum_{\hat{l'}\in S_{l'}^S} Sgn(\hat{l'})f(\hat{l'}\cdot r) &= \sum_{\hat{l'}\in S_{l'}^S} Sgn(\hat{l'})\Big(f(\hat{l'}\cdot r') + W\Big)\\ &= \sum_{\hat{l'}\in S_{l'}^S} Sgn(\hat{l'})f(\hat{l'}\cdot r').\end{aligned}$$

(since W is a constant, and by Corollary 1)

Thus,

$$\begin{aligned}f_{l'}(r) &= Sgn(l')\left[\sum_{\hat{l'}\in S_{l'}^L} Sgn(\hat{l'})f(\hat{l'}\cdot r') + \sum_{\hat{l'}\in S_{l'}^S} Sgn(\hat{l'})f(\hat{l'}\cdot r')\right]\\ &= Sgn(l')\sum_{\hat{l'}\lesssim l'} Sgn(\hat{l'})f(\hat{l'}\cdot r')\\ &= f_{l'}(r')\\ &= 0 \text{ (by condition [2] in Lemma 8).}\end{aligned}$$

Next, we compute $f_l(r)$.
We define S_l^L and S_l^S as

$$\begin{aligned}S_l^L &= \{\hat{l}|\hat{l}\lesssim l,\ f(\hat{l}\cdot r') > f(l_c\cdot r')\}\\ S_l^S &= \{\hat{l}|\hat{l}\lesssim l,\ f(\hat{l}\cdot r') < f(l_c\cdot r')\}\setminus\{l_c|_{x_q=0}, l_c|_{x_q=1}\}.\end{aligned}$$

Note that the number of left input assignments that belong to S_l^L (resp. S_l^S) and whose weights are even is equal to the number of left input assignments that belong to S_l^L (resp. S_l^S) and whose weights are odd by condition [4] in Lemma 8. To compute $f_l(r)$, we expand $f_l(r)$ into three terms as follows:

$$\begin{aligned}f_l(r) &= Sgn(l)\sum_{\hat{l}\lesssim l} Sgn(\hat{l})f(\hat{l}\cdot r)\\ &= Sgn(l)\Bigg[\sum_{\hat{l}\in S_l^L} Sgn(\hat{l})f(\hat{l}\cdot r) + \sum_{\hat{l}\in S_l^S} Sgn(\hat{l})f(\hat{l}\cdot r)\\ &\quad + \big(Sgn(l_c|_{x_q=0})f(l_c|_{x_q=0}\cdot r) + Sgn(l_c|_{x_q=1})f(l_c|_{x_q=1}\cdot r)\big)\Bigg]\end{aligned}$$

For the first term,

$$\begin{aligned}\sum_{\hat{l}\in S_l^L} Sgn(\hat{l})f(\hat{l}\cdot r) &= \sum_{\hat{l}\in S_l^L} Sgn(\hat{l})\left(f(\hat{l}\cdot r')+W-(2^t+2^s)\right)\\ &= \sum_{\hat{l}\in S_l^L} Sgn(\hat{l})f(\hat{l}\cdot r').\end{aligned}$$

(since $W-(2^t+2^s)$ is a constant, and by Corollary 1)

For the second term,

$$\begin{aligned}\sum_{\hat{l}\in S_l^S} Sgn(\hat{l})f(\hat{l}\cdot r) &= \sum_{\hat{l}\in S_l^S} Sgn(\hat{l})(f(\hat{l}\cdot r')+W)\\ &= \sum_{\hat{l}\in S_l^S} Sgn(\hat{l})f(\hat{l}\cdot r').\end{aligned}$$

(since W is a constant, and by Corollary 1)

Note that, by Lemma 9, one of $f(l_c|_{x_q=0}\cdot r')$ and $f(l_c|_{x_q=1}\cdot r')$ is $f(l_c\cdot r')$, and the other is less than $f(l_c\cdot r')$. That is, one of $f(l_c|_{x_q=0}\cdot r')+W$ and $f(l_c|_{x_q=1}\cdot r')+W$ is equal to 2^t+2^s, and the other is less than 2^t+2^s. Thus, for the last term,

$$\begin{aligned}& Sgn(l_c|_{x_q=0})f(l_c|_{x_q=0}\cdot r)+Sgn(l_c|_{x_q=1})f(l_c|_{x_q=1}\cdot r)\\ =\ & Sgn(l_c|_{x_q=0})\left(f(l_c|_{x_q=0}\cdot r')+W\right)+Sgn(l_c|_{x_q=1})\left(f(l_c|_{x_q=1}\cdot r')+W\right)\\ & +\left(-Sgn(l_c)(2^t+2^s)\right)\\ =\ & Sgn(l_c|_{x_q=0})f(l_c|_{x_q=0}\cdot r')+Sgn(l_c|_{x_q=1})f(l_c|_{x_q=1}\cdot r')\\ & +\left(-Sgn(l_c)(2^t+2^s)\right).\end{aligned}$$

Thus,

$$\begin{aligned}f_l(r) &= Sgn(l)\left[\sum_{\hat{l}\in S_l^L} Sgn(\hat{l})f(\hat{l}\cdot r')+\sum_{\hat{l}\in S_l^S} Sgn(\hat{l})f(\hat{l}\cdot r')\right.\\ &\quad \left.+\left(Sgn(l_c|_{x_q=0})f(l_c|_{x_q=0}\cdot r)+Sgn(l_c|_{x_q=1})f(l_c|_{x_q=1}\cdot r)\right)\right]\\ &= Sgn(l)\left[\sum_{\hat{l}\lesssim l} Sgn(\hat{l})f(\hat{l}\cdot r')+\left(-Sgn(l_c)(2^t+2^s)\right)\right]\\ &= f_l(r')+Sgn(l)\left(-Sgn(l_c)(2^t+2^s)\right)\\ &= Sgn(l)\left(-Sgn(l_c)(2^t+2^s)\right).\end{aligned}$$

Thus, $f_l(r)\neq 0$ and $f_{l'}(r)=0$. Hence A is a fooling set. □

We now give the proof of Theorem 3

(Proof) We assume the same situation as in the proof of Theorem 2.

(I) $|Split'_p| \geq n/12$

(i) $|Split'_p \cap (X_{UL} \times X_{DR})| \geq |Split'_p \cap (X_{UR} \times X_{DL})|$

We define $Split''_p$ as $Split''_p = Split'_p \cap (X_{UL} \times X_{DR})$. Then $|Split''_p| \geq \frac{n}{24}$, since $|Split'_p| \geq \frac{n}{12}$ and $|Split'_p \cap (X_{UL} \times X_{DR})| \geq |Split'_p \cap (X_{UR} \times X_{DL})|$.

If there exists no pair of (x_w, x_v) ($x_w, x_v \in X \cap R$ and $w = v + s_h - s_l$), we can reconstruct a left partition L' and a right partition R' such that there exists exactly one pair of (x_w, x_v) ($x_w, x_v \in X \cap R'$ and $w = v + s_h - s_l$), and there exist more than $n/24 - 1$ pairs $(x_{i+s_h-s_l} \in L', x_i \in R') \in Split''_p$, by decreasing the cardinality of L and increasing the cardinality of R, where s_h and s_l are the high split bit and the low split bit respectively. In this case, we can use L' and R' as L and R respectively.

Let (x_w, x_v) be a pair such that $x_w, x_v \in X \cap R'$ and $w = v + s_h - s_l$.

Similarly to the previous section, let A be a set of all left input assignments satisfying the following conditions.

Conditions: For any variable x such that $x \in \{x_u | (x_u, x_d) \in Split''_p\} \stackrel{\text{def}}{=} B$,

[1] If x is the lowest member in B, that is, x has the minimum index in B, then the value 1 is assigned to x. We define this variable x to be x_q.

[2] The value 1 is assigned to at least one member of $B \setminus \{x_q\}$. That is, the value 1 is assigned to each of at least two members of B, including x_q.

[3] If $y_{s_h} \in L$, then the value 1 is assigned to y_{s_h}, where s_h is the high split bit.

[4] If $y_{s_l} \in L$, then the value 1 is assigned to y_{s_l}, where s_l is the low split bit.

[5] The value 0 is assigned to each of the variables that belong to L other than mentioned above.

It is obvious that $|A| \geq 2^{\frac{n}{24}-2} - 1$. We show that A is a fooling set in the following. If $|A| \leq 1$, it is obvious that A is a fooling set. We cope with the case that $|A| \geq 2$ in the following.

We define a right input assignment r' as follows:

[1] If $y_{s_h} \in R$, then r' assigns 1 to y_{s_h}.

[2] If $y_{s_l} \in R$, then r' assigns 1 to y_{s_l}.

[3] r' assigns 1 to x_w.

[4] r' assigns 0 to the variables that belong to R other than mentioned above.

X												
Variable	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
Partition	L	L	R	L	L	L	R	R	R	R	L	R
Assignment	1	1	1	0	0	1	0	0	0	0	0	0
Y												
Variable	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
Assignment	0	0	0	0	0	1	0	0	0	0	0	1

$Split''_p = \{(x_{11}, x_5), (x_{10}, x_4), (x_8, x_2), (x_6, x_0)\}$, $s_h = 6$, $s_l = 0$, $q = 6$

Figure 2.12: An example of an assignment

X												
Variable	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
Partition	L	L	R	L	L	L	R	R	R	R	L	R
Assignment	1	1	1	0	0	1	1	1	1	0	0	1
Y												
Variable	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
Assignment	0	0	0	0	0	1	0	0	0	0	0	1

$Split''_p = \{(x_{11}, x_5), (x_{10}, x_4), (x_8, x_2), (x_6, x_0)\}$, $s_h = 6$, $s_l = 0$, $q = 6$

Figure 2.13: An example of an assignment

An example of an assignment satisfying [1]–[4] is shown in Fig.2.12.

Let l and l' be any two distinct left input assignments that belong to A. We define l_c for l, l' and r'. We assume that $l_c \lesssim l$ and $l_c \not\lesssim l'$ without loss of generality. We define a right input assignment r as follows:

[1] If $y_{s_h} \in R$, then r assigns 1 to y_{s_h}.

[2] If $y_{s_l} \in R$, then r assigns 1 to y_{s_l}.

[3] r assigns 1 to x_w and x_v.

[4] If the value 1 is assigned to $x_{i+s_h-s_l}$ in l_c, that is, $l_c(x_{i+s_h-s_l}) = 1$, then r assigns 1 to x_i.

[5] r assigns 0 to the other variables that belong to R other than mentioned above.

An example of an assignment satisfying [1]–[5] is shown in Fig.2.13.

The remainder of 2^i $(i \geq s_h)$ divided by $2^{s_h} + 2^{s_l}$ is $(2^{s_h} + 2^{s_l}) - 2^{i-(s_h-s_l)}$ by Corollary 4, and the remainder of $2^{i-(s_h-s_l)}$ $(2s_h - s_l > i \geq s_h)$ divided by

$2^{s_h} + 2^{s_l}$ is $2^{i-(s_h-s_l)}$. Thus the remainder of $2^i + 2^{i-(s_h-s_l)}$ $(2s_h - s_l > i \geq s_h)$ divided by $2^{s_h} + 2^{s_l}$ is 0. Thus,

$$f(l_c \cdot r) = 0.$$

We show that A satisfies the conditions in Lemma 8 in the following. First, we cope with the case that both y_{s_h} and y_{s_l} belong to R (case(a)), and then, we cope with the case that either y_{s_h} or y_{s_l} belongs to L (case(b)).

(a) Both y_{s_h} and y_{s_l} belong to R.

Since the variables in X with the assignment of 1 in $\hat{l} \cdot r'$ for $\hat{l} \lesssim l \in A$ are higher than the variable x_{s_h}, $f(\hat{l} \cdot r') = (2^{s_h} + 2^{s_l}) - \frac{X(\hat{l} \cdot r')}{2^{s_h - s_l}}$ for $\hat{l} \lesssim l \in A$ by Corollary 4. Thus by Lemma 2, Lemma 5 and Lemma 6,

$$\begin{aligned} f_l(r') &= Sgn(l) \sum_{\hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l} \cdot r') \\ &= Sgn(l) \sum_{\hat{l} \lesssim l} Sgn(\hat{l}) \left((2^{s_h} + 2^{s_l}) - \frac{X(\hat{l} \cdot r')}{2^{s_h - s_l}} \right) \\ &= 0. \end{aligned}$$

Similarly,

$$f_{l'}(r') = 0.$$

(b) Either y_{s_h} or y_{s_l} belongs to L.

Note that, in this case, either y_{s_h} or y_{s_l} belongs to R by the definition of split bits (s_h and s_l). For $\hat{l} \lesssim l$ or $\hat{l} \lesssim l'$, $f(\hat{l} \cdot r') = 0$ if the value 0 is assigned to y_{s_h} or y_{s_l} that belongs to L in $\hat{l}$, and $f(\hat{l} \cdot r') = (2^{s_h} + 2^{s_l}) - \frac{X(\hat{l} \cdot r')}{2^{s_h - s_l}}$ by Corollary 4 if the value 1 is assigned to y_{s_h} or y_{s_l} that belongs to L in $\hat{l}$.

Let S be the set of left input assignments such that $\hat{l} \lesssim l$ and $f(\hat{l} \cdot r') = (2^{s_h} + 2^{s_l}) - \frac{X(\hat{l} \cdot r')}{2^{s_h - s_l}}$. Thus, by Lemma 2, Corollary 1 and Corollary 2,

$$\begin{aligned} f_l(r') &= Sgn(l) \sum_{\hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l} \cdot r') \\ &= Sgn(l) \left[\sum_{\hat{l} \notin S, \hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l} \cdot r') + \sum_{\hat{l} \in S} Sgn(\hat{l}) f(\hat{l} \cdot r') \right] \\ &= Sgn(l) \left[0 + \sum_{\hat{l} \in S} Sgn(\hat{l}) \left((2^{s_h} + 2^{s_l}) - \frac{X(\hat{l} \cdot r')}{2^{s_h - s_l}} \right) \right] \\ &= 0. \end{aligned}$$

Similarly,

$$f_{l'}(r') = Sgn(l') \sum_{\hat{l} \lesssim l'} Sgn(\hat{l}) f(\hat{l} \cdot r') = 0.$$

Hence the set A satisfies condition [2] in Lemma 8.

It is obvious that $f(\hat{l}|_{x_q=0} \cdot r') > f(\hat{l}|_{x_q=1} \cdot r')$ for $\hat{l} \lesssim l$ or $\hat{l} \lesssim l'$ by Corollary 4. We assume that $\hat{l} \lesssim l$ or $\hat{l} \lesssim l'$, $\hat{l}|_{x_q=0} \neq l_c$, $\hat{l}|_{x_q=1} \neq l_c$, and $f(\hat{l}|_{x_q=0} \cdot r') > f(l_c \cdot r') > f(\hat{l}|_{x_q=1} \cdot r')$. There exists no left input assignment $\hat{l}' \lesssim l$ or $\hat{l}' \lesssim l'$ such that $f(\hat{l}|_{x_q=0} \cdot r') > f(\hat{l}' \cdot r') > f(\hat{l}|_{x_q=1} \cdot r')$ since x_q is the lowest bit to which the value 1 is assigned and by Corollary 4. This is a contradiction. Hence the set A satisfies condition [4] in Lemma 8.

It is obvious that the set A satisfies conditions [1], [3], [5] and [6] in Lemma 8. Hence A is a fooling set.

(ii) $|Split'_p \cap (X_{UL} \times X_{DR})| < |Split'_p \cap (X_{UR} \times X_{DL})|$

We define $Split''_p$ as $Split''_p = Split'_p \cap (X_{UR} \times X_{DL})$. Then $|Split''_p| \geq \frac{n}{24}$ since $|Split'_p| \geq \frac{n}{12}$ and $|Split'_p \cap (X_{UL} \times X_{DR})| < |Split'_p \cap (X_{UR} \times X_{DL})|$.

Similarly to the previous section, let A be a set of all left input assignments satisfying the following conditions.

Conditions: For any variable x such that $x \in \{x_d | (x_u, x_d) \in Split''_p\} \stackrel{\text{def}}{=} B$,

[1] If x is the lowest member in B, that is, x has the minimum index in B, then the value 1 is assigned to x. We define this variable x to be x_q.

[2] The value 1 is assigned to at least one member of $B \setminus \{x_q\}$. That is, the value 1 is assigned to each of at least two members of B, including x_q.

[3] If $y_{s_h} \in L$, then the value 1 is assigned to y_{s_h}, where s_h is the high split bit.

[4] If $y_{s_l} \in L$, then the value 1 is assigned to y_{s_l}, where s_l is the low split bit.

[5] The value 0 is assigned to each of the variables that belong to L other than mentioned above.

It is obvious that $|A| \geq 2^{\frac{n}{24}-1} - 1$. We show that A is a fooling set in the following. If $|A| \leq 1$, it is obvious that A is a fooling set. We cope with the case that $|A| \geq 2$ in the following.

We define a right input assignment r' as follows:

[1] If $y_{s_h} \in R$, then r' assigns 1 to y_{s_h}.

X												
Variable	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
Partition	R	R	L	R	R	R	L	L	R	L	R	L
Assignment	0	0	0	0	0	0	1	1	0	0	0	1
Y												
Variable	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
Assignment	0	0	0	0	0	1	0	0	0	0	0	1

$Split''_p = \{(x_{11}, x_5), (x_{10}, x_4), (x_8, x_2), (x_6, x_0)\}$, $s_h = 6$, $s_l = 0$, $q = 0$

Figure 2.14: An example of an assignment

X												
Variable	x_{11}	x_{10}	x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
Partition	R	R	L	R	R	R	L	L	R	L	R	L
Assignment	1	1	0	0	0	1	1	1	0	0	0	1
Y												
Variable	y_{11}	y_{10}	y_9	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
Assignment	0	0	0	0	0	1	0	0	0	0	0	1

$Split''_p = \{(x_{11}, x_5), (x_{10}, x_4), (x_8, x_2), (x_6, x_0)\}$, $s_h = 6$, $s_l = 0$, $q = 0$

Figure 2.15: An example of an assignment

[2] If $y_{s_l} \in R$, then r' assigns 1 to y_{s_l}.

[3] r' assigns 0 to the variables that belong to R other than mentioned above.

An example of an assignment satisfying [1]–[3] is shown in Fig.2.14.

Let l and l' be any two distinct left input assignments that belong to A. We define l_c for l, l' and r'. We assume that $l_c \lesssim l$ and $l_c \not\lesssim l'$ without loss of generality. We define a right input assignment r as follows:

[1] If $y_{s_h} \in R$, then r assigns 1 to y_{s_h}.

[2] If $y_{s_l} \in R$, then r assigns 1 to y_{s_l}.

[3] If the value 1 is assigned to x_i in l_c, that is, $l_c(x_i) = 1$, then r assigns 1 to $x_{i+s_h-s_l}$.

[4] r assigns 0 to the variables that belong to R other than mentioned above.

An example of an assignment satisfying [1]–[4] is shown in Fig.2.15.

We show that A satisfies the conditions in Lemma 8 in the following. First, we cope with the case that $y_{s_h} \in R$ and $y_{s_l} \in R$, or $y_{s_h} \in R$ and $y_{s_l} \in L$ (case(a)), and then, we cope with the case that $y_{s_h} \in L$ and $y_{s_l} \in R$ (case(b)).

(a) $y_{s_h} \in R$ and $y_{s_l} \in R$, or $y_{s_h} \in R$ and $y_{s_l} \in L$.

For $\hat{l} \lesssim l \in A$, since $X(\hat{l} \cdot r') < 2^{s_h}$, $f(\hat{l} \cdot r') = X(\hat{l} \cdot r')$. Thus by Lemma 2 and Lemma 6,

$$f_l(r') = Sgn(l) \sum_{\hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l} \cdot r') = 0.$$

Similarly,

$$f_{l'}(r') = Sgn(l') \sum_{\hat{l} \lesssim l'} Sgn(\hat{l}) f(\hat{l} \cdot r') = 0.$$

(b) $y_{s_h} \in L$ and $y_{s_l} \in R$.

For $\hat{l} \lesssim l$, $f(\hat{l} \cdot r') = 0$ if the value 0 is assigned to y_{s_h} in $\hat{l}$, and $f(\hat{l} \cdot r') = X(\hat{l} \cdot r')$ if the value 1 is assigned to y_{s_h} in $\hat{l}$.

Let S be the set of left input assignments such that $\hat{l} \lesssim l$ and $f(\hat{l} \cdot r') = X(\hat{l} \cdot r')$. Thus, by Lemma 2 and Corollary 2,

$$\begin{aligned} f_l(r') &= Sgn(l) \sum_{\hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l} \cdot r') \\ &= Sgn(l) \left[\sum_{\hat{l} \notin S, \hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l} \cdot r') + \sum_{\hat{l} \in S} Sgn(\hat{l}) f(\hat{l} \cdot r') \right] \\ &= Sgn(l) \left[0 + \sum_{\hat{l} \in S} Sgn(\hat{l}) X(\hat{l} \cdot r') \right] \\ &= 0. \end{aligned}$$

Similarly,

$$f_{l'}(r') = Sgn(l') \sum_{\hat{l} \lesssim l'} Sgn(\hat{l}) f(\hat{l} \cdot r') = 0.$$

Hence the set A satisfies condition [2] in Lemma 8.

By the same reason as shown in (i), the set A satisfies condition [4] in Lemma 8. It is obvious that the set A satisfies conditions [1], [3], [5] and [6] in Lemma 8. Hence A is a fooling set.

(II) $|Split'_p| < n/12$

There exist more than $\frac{n}{8} - \frac{n}{12} = \frac{n}{24}$ pairs of (x_i, y_i) such that $x_i \in R$ and $y_i \in L$ by Lemma 4.

Variable	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
Partition	L	R	R	R	R	R	L	R
Assignment	0	1	0	0	1	0	0	0

Variable	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
Partition	L	R	L	L	L	L	R	L
lr	0	1	0	1	0	1	0	1
$l'r$	0	1	1	0	1	1	0	1
$l''r$	0	1	0	0	0	1	0	1

$w = 6$, $q = 3$

Figure 2.16: An example of an assignment

Similarly to (II) in the previous section, let (x_w, y_w) be a pair such that $x_w \in R$ and $y_w \in R$.

Similarly to the previous section, let A be a set of all left input assignments satisfying the following conditions.

Conditions:

[1] If p is the minimum index such that $x_p \in R$ and $y_p \in L$, then the value 1 is assigned to y_p.

[2] For $i \neq p$, $x_i \in R$ and $y_i \in L$, the value 1 is assigned to at least one variable.

[3] The value 0 is assigned to each of the variables that belong to L other than mentioned above.

It is obvious that $|A| \geq 2^{\frac{n}{24}-1} - 1$. We show that A is a fooling set in the following. If $|A| \leq 1$, it is obvious that A is a fooling set. We cope with the case that $|A| \geq 2$ in the following.

Let l and l' be any two distinct left input assignments that belong to A. We assume that q is the minimum index of $y_i \in Y$ such that $y_i \in L$ and $l(y_i) \neq l'(y_i)$. We assume that $l(y_q) = 0$ and $l'(y_q) = 1$ without loss of generality. We define a left input assignment l'' in the same way as in (II) in the previous section

We define a right input assignment r as follows:

[1] r assigns 1 to each variable x_w, y_w and x_q.

[2] r assigns 0 to the other variables that belong to R.

An example of an assignment satisfying [1] and [2] is shown in Fig.2.16.

We define SL_l, SS_l, $SL_{l'}$, and $SS_{l'}$ in the same way as in (II) in the previous section. Then we obtain the following.

$$\forall \hat{l} \in SL_l \qquad f(\hat{l} \cdot r) = 2^w + 2^q$$
$$\forall \hat{l'} \in SL_{l'} \qquad f(\hat{l'} \cdot r) = \begin{cases} 0 & (Y(\hat{l'} \cdot r) = 2^w + 2^q) \\ 2^w + 2^q & (\text{otherwise}) \end{cases}$$

Note that $SS_l = SS_{l'} = \{\hat{l} | \hat{l} \lesssim l''\}$. By Lemma 2,

$$\begin{aligned} f_l(r) &= Sgn(l) \sum_{\hat{l} \lesssim l} Sgn(\hat{l}) f(\hat{l} \cdot r) \\ &= Sgn(l) \left[\sum_{\hat{l} \in SL_l} Sgn(\hat{l}) f(\hat{l} \cdot r) + \sum_{\hat{l} \in SS_l} Sgn(\hat{l}) f(\hat{l} \cdot r) \right] \\ &= Sgn(l) \left[\sum_{\hat{l} \in SL_l} Sgn(\hat{l}) f(\hat{l} \cdot r) + \sum_{\hat{l} \lesssim l''} Sgn(\hat{l}) f(\hat{l} \cdot r) \right]. \end{aligned}$$

Similarly,

$$f_{l'}(r) = Sgn(l') \left[\sum_{\hat{l'} \in SL_{l'}} Sgn(\hat{l'}) f(\hat{l'} \cdot r) + \sum_{\hat{l} \lesssim l''} Sgn(\hat{l}) f(\hat{l} \cdot r) \right].$$

The number of left input assignments that belong to SL_l and whose weights are even is equal to the number of left input assignments that belong to SL_l and whose weights are odd. Thus by Corollary 1,

$$f_l(r) = Sgn(l) \left[0 + \sum_{\hat{l} \lesssim l''} Sgn(\hat{l}) f(\hat{l} \cdot r) \right].$$

Note that when $Y(\hat{l'} \cdot r) = 2^w + 2^q$, the weight of $\hat{l'}$ is 1. By Corollary 1,

$$f_{l'}(r) = Sgn(l') \left[-\{-(2^w + 2^q)\} + \sum_{\hat{l'} \lesssim l''} Sgn(\hat{l'}) f(\hat{l'} \cdot r) \right].$$

Next, we define a right input assignment r' as

[1] r' assigns 1 to each variable x_w and y_w.

[2] r' assigns 0 to the other variables that belong to R.

Variable	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0
Partition	L	R	R	R	R	R	L	R
Assignment	0	1	0	0	0	0	0	0

Variable	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
Partition	L	R	L	L	L	L	R	L
lr'	0	1	0	1	0	1	0	1
$l'r'$	0	1	1	0	1	1	0	1
$l''r'$	0	1	0	0	0	1	0	1

$w = 6,\ q = 3$

Figure 2.17: An example of an assignment

An example of an assignment satisfying [1], [2] is shown in Fig.2.17.
For $\hat{l} \lesssim l$ or $\hat{l} \lesssim l'$,

$$f(\hat{l} \cdot r') = \begin{cases} 0 & (Y(\hat{l} \cdot r') = 2^w) \\ 2^w & (\text{otherwise}). \end{cases}$$

Note that when $Y(\hat{l} \cdot r') = 2^w$, the weight of $\hat{l}$ is 0. Thus, by Lemma 2 and Lemma 5,

$$\begin{aligned} f_l(r') &= Sgn(l) \sum_{\hat{l} \lesssim l} Sgn(\hat{l} \cdot r') f(\hat{l} \cdot r') \\ &= -Sgn(l) \cdot 2^w. \end{aligned}$$

Similarly,

$$f_{l'}(r') = -Sgn(l') \cdot 2^w.$$

Therefore

$$f_l(r) f_{l'}(r') \neq f_{l'}(r) f_l(r').$$

Hence, A is a fooling set.

We conclude that a lower bound on the number of nodes of a *BMD representing a remainder function f is $\Omega(2^{\frac{n}{24}})$ from (I), (II) and Theorem 1. □

2.5 Conclusion

In this chapter, we show that a lower bound on the size of a *BMD representing a quotient function or a remainder function is $\Omega(2^{n/24})$. We are expecting

that the techniques in this chapter can be applied to analysis of other graph representations of arithmetic functions.

Researchers have observed that a BMD or a *BMD representing a quotient or a remainder becomes intractable in terms of size even for a small number of inputs. For example, in [10], the authors have given up construction of a BMD or a *BMD for a quotient or a remainder. Our result on the lower bound is consistent with the observation by other researchers. Besides, the time complexity of performing various operations over BMD's or *BMD's is also very high. For some operations, it can be exponential in the number of nodes. Our result on the lower bound in size, the experimental observation and the time complexity seem to suggest that it is hard to handle a BMD or a *BMD for dividers of practical size. We believe that it is significant to search for better graph representations for dividers or other arithmetic functions.

Chapter 3

Kronecker Functional Decision Diagrams and the Complexity of Finding the Best Decomposition Type List

3.1 Introduction

Binary decision diagrams[2, 7], which are data structures representing boolean functions and can be manipulated efficiently in terms of both time and space, are widely used in various areas of computer science. Kronecker functional decision diagrams are extensions of binary decision diagrams such that any decomposition type, such as Shannon expansion and Davio expansion, is allowed for each variable.

Time and space needed to manipulate decision diagrams deeply depend on the number of nodes. Thus it is important to represent boolean functions using as few nodes as possible. Researches has been made on finding the optimal variable ordering for binary decision diagrams and Kronecker functional decision diagrams[5, 22].

As for Kronecker functional decision diagrams, any decomposition type may be selected for each variable. Thus it is possible to reduce the number of nodes by selecting good decomposition types. In this chapter, we show that the following problem is NP-hard: Determine whether we can assign a decomposition type to each variable for a fixed variable ordering so that the number of nodes of shared Kronecker functional decision diagrams representing the function given as a shared binary decision diagram can be less than or equal to K.

The remainder of this chapter has the following organization. In Section 3.2,

we define Kronecker functional decision diagrams. In Section 3.3, we show that the problem of finding the best decomposition types is NP-hard. We conclude with future work in Section 3.4.

3.2 Preliminaries

3.2.1 Ordered Kronecker Functional Decision Diagrams

An ordered Kronecker functional decision diagram (OKFDD) is a directed acyclic graph representing a boolean function. Let the input variables of a given function be $X = \{x_1, x_2, \ldots, x_n\}$. A variable in X is attached to each internal node of an OKFDD. A boolean value is attached to each terminal node. Each internal node has two outgoing edges, called the 0-edge and the 1-edge respectively. Let $low(v)$ and $high(v)$ be the nodes to which the 0-edge and the 1-edge of a node v point respectively. Let $var(v)$ and $value(v)$ be the variable and the boolean value attached to an internal node v and a terminal node v respectively.

Each node of an OKFDD represents a boolean function, and one of the following two conditions holds between adjacent two nodes:

[1] $f[v] = \overline{var(v)} \cdot f[low(v)] \vee var(v) \cdot f[high(v)]$. That is, $f[low(v)] = f[v]|_{var(v)=0}$, $f[high(v)] = f[v]|_{var(v)=1}$. In this case, we say that the decomposition type of the node v is Shannon expansion.

[2] $f[v] = f[low(v)] \oplus var(v) \cdot f[high(v)]$. That is, $f[low(v)] = f[v]|_{var(v)=0}$, $f[high(v)] = f[v]|_{var(v)=0} \oplus f[v]|_{var(v)=1}$. In this case, we say that the decomposition type of the node v is positive Davio expansion.

In the above conditions, $f[v]$ is the function represented by a node v, and $f[v]|_{var(v)=0}$ (resp. $f[v]|_{var(v)=1}$) is the function obtained by substituting 0 (resp. 1) for the variable $var(v)$. We assume that the priority of boolean product $(\cdot)$ is higher than that of exclusive OR $(\oplus)$. We denote 'Shannon expansion' and 'positive Davio expansion' as 'S' and 'pD' respectively.

For any two nodes, if the attached variables are the same, then the attached decomposition types are also the same. A terminal node represents a constant function, whose value is attached to the node. The function represented by the root node is the function represented by the OKFDD. An OKFDD has a variable ordering $\pi = (x_{k_1} < x_{k_2} < \ldots < x_{k_n})$, and if x_{k_i} precedes x_{k_j}, then $i < j$, where $(k_1, \ldots, k_n)$ is a permutation of $(1, \ldots, n)$. We illustrate examples of OKFDD's in Figure 3.1. If decomposition types are restricted only to 'S', we call the decision diagram an ordered binary decision diagram (OBDD).

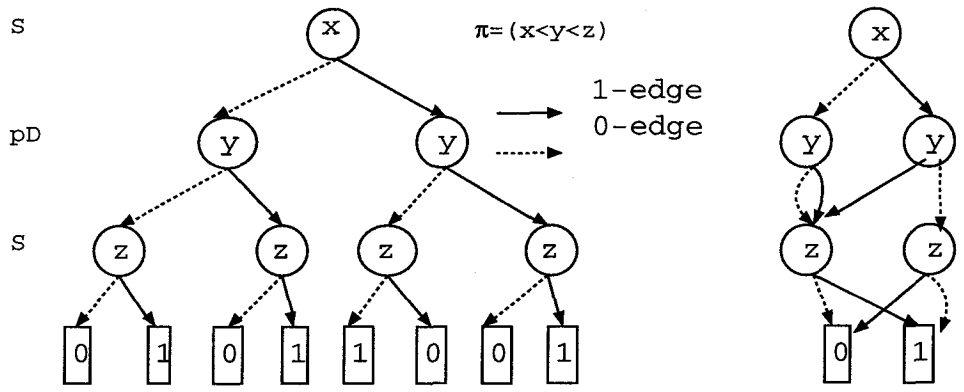


Figure 3.1: Examples of an OKFDD and a reduced OKFDD representing $f = x \oplus \overline{y}z$.

In general, negative Davio expansion ($f[low(v)] = f[v]|_{var(v)=1}$, $f[high(v)] = f[v]|_{var(v)=0} \oplus f[v]|_{var(v)=1}$) is also allowed. However we restrict OKFDD's to take only Shannon expansion and positive Davio expansion.

3.2.2 Reduced OKFDD's

We say that two nodes u and v are equivalent if $low(u) = low(v)$, $high(u) = high(v)$ and $var(u) = var(v)$. Since equivalent nodes represent the same function, one of the two equivalent nodes can be deleted by redirecting the edges pointing to one node to the other. If the decomposition type of a node v is 'S' and $low(v) = high(v)$, then v is called a redundant node. Also if the decomposition type of a node v is 'pD' and $high(v)$ is the constant node of 0, then v is called a redundant node. In both cases, the node v can be deleted by redirecting the edges pointing to v to $low(v)$. An OKFDD is a reduced OKFDD if its equivalent nodes and redundant nodes are maximally deleted. We illustrate an example of a reduced OKFDD in Figure 3.1.

A reduced OKFDD represents a function uniquely with a fixed variable ordering and fixed decomposition types. In this chapter, we use the term "OKFDD" to refer to "reduced OKFDD".

3.2.3 Shared OKFDD's

Given OKFDD's, they can share nodes representing the same functions if we use the same variable ordering for all OKFDD's. A shared OKFDD (SOKFDD) is a set of OKFDD's that share the nodes representing the same functions and maximally reduce the redundant nodes. SOKFDD($\{f_1, f_2, \ldots, f_n\}$), or simply

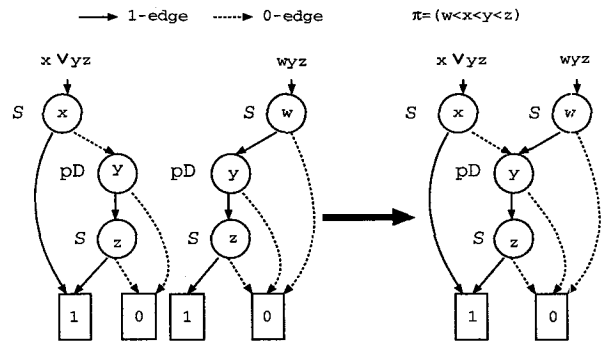


Figure 3.2: An example of an SOKFDD.

SOKFDD($f_1, f_2, \ldots, f_n$), denotes an SOKFDD that represents a set of functions $\{f_1, f_2, \ldots, f_n\}$. If decomposition types are restricted only to 'S', we call the OKFDD a shared OBDD (SOBDD). We illustrate an example of an SOKFDD in Figure 3.2.

3.2.4 Complement Edge

A complement edge is an edge such that the function represented by the node to which it points is negated[12]. If an internal node v has a complement edge, the relation between v and its children is modified as follows:

Case: The 0-edge of v is a complement edge.

$$f[low(v)] = \overline{f[v]|_{var(v)=0}}$$

Case: The 1-edge of v is a complement edge.

- If the decomposition type of v is 'S',
  $f[high(v)] = \overline{f[v]|_{var(v)=1}}$.
- If the decomposition type of v is 'pD',
  $f[high(v)] = \overline{f[v]|_{var(v)=0} \oplus f[v]|_{var(v)=1}}$.

It is possible to share the nodes representing functions complementing each other by introducing complement edges. We illustrate an example of complement edges in Figure 3.3. To preserve the property of unique representation, we restrict the use of complement edges as follows[12]:

- There is only one terminal node, which represents the constant function of 0. The terminal node of 1 is replaced with the terminal node of 0 with a complement edge.

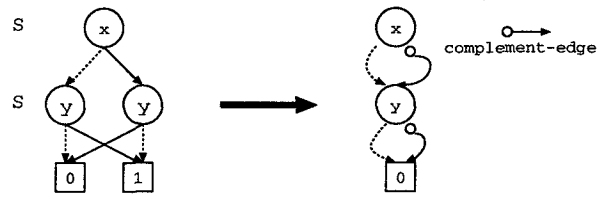


Figure 3.3: An example of an OKFDD representing $f = x \oplus y$ with a complement edge.

- Only 1-edges can be complement edges. We do not use complement edges for 0-edges (if necessary, we use complement edges at lower levels).

We use complement edges for every OKFDD in the following.

3.3 NP-hardness of Decomposition Type Selection Problem of Kronecker Functional Decision Diagrams

In this chapter, we show that the following SOKFDD-MIN problem is NP-hard.

Definition 1 [SOKFDD-MIN]

INSTANCE *A variable ordering* $\pi = (x_{k_1} < x_{k_2} < \ldots < x_{k_n})$, *an SOBDD representing functions* $f_i (1 \leq i \leq n)$, *and a constant* K.

PROBLEM *Are there decomposition types according to which we can construct an SOKFDD*$(f_1, \ldots, f_n)$ *with at most* K *nodes for the variable ordering* π*?*

□

We reduce 3-SAT problem, which is known to be NP-complete, to SOKFDD-MIN problem in the following.

Definition 2 [3-SAT]

INSTANCE *A CNF expression of a boolean function* $F = C_1 C_2 \ldots C_n$, *where each clause has exactly three literals (i.e. variables or their complements).*

PROBLEM *Is there an assignment that satisfies* F*?*

□

Definition 3 [Transforming 3-SAT to SOKFDD-MIN]

Let the boolean function of a given instance of 3-SAT be $F = C_1C_2 \ldots C_n$. Let the set of the variables on which F depends be $X = \{x_1, x_2, \ldots, x_m\}$. We consider the variable ordering $\pi' = (x_1 < \ldots < x_m)$. For each clause $C_i = l_{i_1} \vee l_{i_2} \vee l_{i_3}$ ($l_{i_j} = x_{i_j}$ or $l_{i_j} = \overline{x_{i_j}}$), a boolean function G_i is generated as follows. We assume that $x_{i_1} < x_{i_2} < x_{i_3}$ in π' without loss of generality. Let the set of the variables on which G_i depends be $\{x_{i_1}, x_{i_2}, x_{i_3}, a_i, b_i, \ldots, h_i\}$, where $x_{i_1}, x_{i_2}, x_{i_3} \in X$, and $a_i, b_i, \ldots, h_i \notin X$.

[1]

Case: $l_{i_1} = x_{i_1}$

$$G_i = \overline{x_{i_1}} H^i_{ABCD} \vee x_{i_1} H^i_{EFGH}$$

Case: $l_{i_1} = \overline{x_{i_1}}$

$$G_i = \overline{x_{i_1}} H^i_{ABCD} \vee x_{i_1} (H^i_{ABCD} \oplus H^i_{EFGH})$$

[2]

Case: $l_{i_2} = x_{i_2}$

$$\begin{aligned} H^i_{ABCD} &= \overline{x_{i_2}} H^i_{AB} \vee x_{i_2} H^i_{CD} \\ H^i_{EFGH} &= \overline{x_{i_2}} H^i_{EF} \vee x_{i_2} H^i_{GH} \end{aligned}$$

Case: $l_{i_2} = \overline{x_{i_2}}$

$$\begin{aligned} H^i_{ABCD} &= \overline{x_{i_2}} H^i_{AB} \vee x_{i_2} (H^i_{AB} \oplus H^i_{CD}) \\ H^i_{EFGH} &= \overline{x_{i_2}} H^i_{EF} \vee x_{i_2} (H^i_{EF} \oplus H^i_{GH}) \end{aligned}$$

[3]

Case: $l_{i_3} = x_{i_3}$

$$\begin{aligned} H^i_{AB} &= \overline{x_{i_3}} a_i \vee x_{i_3} b_i \quad , \quad H^i_{CD} = \overline{x_{i_3}} c_i \vee x_{i_3} d_i \\ H^i_{EF} &= \overline{x_{i_3}} e_i \vee x_{i_3} f_i \quad , \quad H^i_{GH} = \overline{x_{i_3}} g_i \vee x_{i_3} h_i \end{aligned}$$

Case: $l_{i_3} = \overline{x_{i_3}}$

$$\begin{aligned} H^i_{AB} &= \overline{x_{i_3}} a_i \vee x_{i_3} (a_i \oplus b_i) \\ H^i_{CD} &= \overline{x_{i_3}} c_i \vee x_{i_3} (c_i \oplus d_i) \\ H^i_{EF} &= \overline{x_{i_3}} e_i \vee x_{i_3} (e_i \oplus f_i) \\ H^i_{GH} &= \overline{x_{i_3}} g_i \vee x_{i_3} (g_i \oplus h_i) \end{aligned}$$

For each $i(1 \leq i \leq n)$, we define boolean functions $g_{i\_000}, g_{i\_001}, g_{i\_002}, g_{i\_010}, \ldots, g_{i\_221}$ as follows.

$$
\begin{array}{ll}
g_{i\_000} = a_i, & g_{i\_001} = b_i, \\
g_{i\_002} = a_i \oplus b_i, & g_{i\_010} = c_i, \\
g_{i\_011} = d_i, & g_{i\_012} = c_i \oplus d_i, \\
g_{i\_020} = a_i \oplus c_i, & g_{i\_021} = b_i \oplus d_i, \\
g_{i\_022} = a_i \oplus b_i \oplus c_i \oplus d_i, & g_{i\_100} = e_i, \\
g_{i\_101} = f_i, & g_{i\_102} = e_i \oplus f_i, \\
g_{i\_110} = g_i, & g_{i\_111} = h_i, \\
g_{i\_112} = g_i \oplus h_i, & g_{i\_120} = e_i \oplus g_i, \\
g_{i\_121} = f_i \oplus h_i, & g_{i\_122} = e_i \oplus f_i \oplus g_i \oplus h_i, \\
g_{i\_200} = a_i \oplus e_i, & g_{i\_201} = b_i \oplus f_i, \\
g_{i\_202} = a_i \oplus b_i \oplus e_i \oplus f_i, & g_{i\_210} = c_i \oplus g_i, \\
g_{i\_211} = d_i \oplus h_i, & g_{i\_212} = c_i \oplus d_i \oplus g_i \oplus h_i, \\
g_{i\_220} = a_i \oplus c_i \oplus e_i \oplus g_i, & g_{i\_221} = b_i \oplus d_i \oplus f_i \oplus h_i.
\end{array}
$$

Then the resulting instance of SOKFDD-MIN is as follows.

INSTANCE The variable ordering $\pi = (x_1 < x_2 < \ldots < x_m < a_1 < b_1 < \ldots < h_1 < \ldots < a_i < b_i < \ldots < h_n)$, the SOBDD representing the set of the functions $T = \{G_i, g_{i\_000}, \ldots, g_{i\_221} | 1 \leq i \leq n\}$, and the constant $K = 39n + 1$. □

We consider the time complexity of the above transformation. G_i is generated based only on the combination of the positive and negative literals in C_i. Thus there are only the eight candidates for G_i. If we prepare the eight OBDD's that represent the candidates in advance, the OBDD that represents G_i can be constructed by copying the appropriate candidate and attaching the variables to its nodes. Similarly, the OBDD's that represent $g_{i\_000}, \ldots, g_{i\_221}$ can be constructed by copying the prepared candidates. Then we reduce the constructed OBDD's and obtain an SOBDD that represents T. This reducing procedure can be done in polynomial time[7]. It is obvious that the variable ordering π and the constant K can be determined in polynomial time. Therefore the whole procedure can be done in polynomial time. □

In the following, we show that 3-SAT is reduced to SOKFDD-MIN by the above transformation.

Theorem 4 *For the original instance of 3-SAT and the resulting instance of*

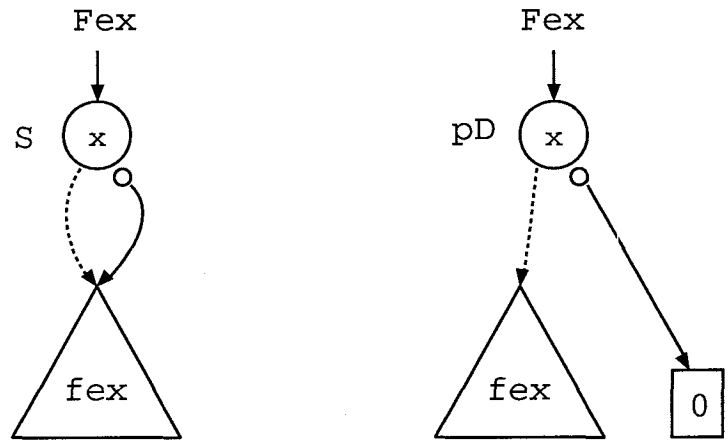


Figure 3.4: Examples of OKFDD's representing Fex with complement edges.

SOKFDD-MIN described above, the following holds.

The boolean function F of the instance of 3-SAT is satisfiable.

$\Leftrightarrow$

There are decomposition types according to which the number of nodes of SOKFDD(T) is less than or equal to K.

□

We introduce the following lemmas and definition.

Lemma 10 *The number of nodes of SOKFDD($g_{i\_000}, \ldots, g_{i\_221}$) without terminal nodes does not depend on the decomposition types, and it is 32.* □

(Proof)

We consider the following boolean function Fex.

$$Fex = x \oplus fex,$$

where x is a boolean variable and fex is a boolean function. Let v be the node that represents Fex in an OKFDD, that is, $f[v] = Fex$. We assume that $var(v) = x$. Then Fex is decomposed in terms of Shannon expansion as $f[low(v)] = fex$ and $f[high(v)] = \overline{fex}$. Also Fex is decomposed in terms of positive Davio expansion as $f[low(v)] = fex$ and $f[high(v)] = fex \oplus \overline{fex} = 1$. We illustrate examples of OKFDD's representing Fex in Figure 3.4. By Figure 3.4, it is obvious that the number of nodes of an OKFDD representing Fex does not depend

on the decomposition type on x. Since $g_{i\_000}, \ldots, g_{i\_221}$ are literals or boolean functions that can be expressed by having only exclusive OR's as operators, any node of SOKFDD($g_{i\_000}, \ldots, g_{i\_221}$) represents a boolean function of the form Fex or a constant. Therefore the number of nodes of SOKFDD($g_{i\_000}, \ldots, g_{i\_221}$) does not depend on the given decomposition types. If a child u of a node v in SOKFDD($g_{i\_000}, \ldots, g_{i\_221}$) is an internal node, u represents the boolean function $f[u] = f[v]|_{var(v)=0}$ by Figure 3.4. Thus each node of SOKFDD($g_{i\_000}, \ldots, g_{i\_221}$) represents a boolean function that is obtained by substituting 0's for the former k variables (in the given variable ordering for some $1 \leq k \leq 8$) of some g_{i_xxx}. It is straightforward to see that the number of the distinct boolean functions thus obtained is 32. □

We consider the following one-to-one correspondence. Let F be the boolean function of the instance of 3-SAT, and let X be the set of the variables on which F depends. Let A be the set of possible assignments of boolean values to X. Note that the set of the variables on which the boolean functions in $T = \{G_i, g_{i\_000}, \ldots, g_{i\_221} | 1 \leq i \leq n\}$ depends includes X. Then let R be the set of possible assignments of decomposition types to X, and we define the following one-to-one correspondence M between A and R.

[one-to-one correspondence M]

For an assignment $a \in A$, we assign a decomposition type to each variable in X as follows:

- If the value of x in a is 1 (resp. 0), then we assign S (resp. pD) to x.

Let the above assignment of decomposition types be r. We define the mapping M to be $M(a) = r$. It is obvious that the mapping M is a bijection. □

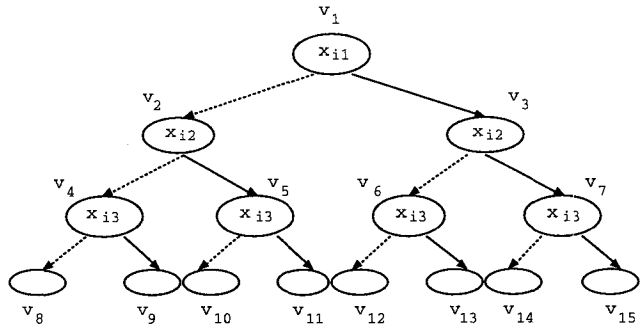
Lemma 11 *For the boolean function F of the original instance of 3-SAT and the transformed instance of SOKFDD-MIN, the following holds.*

Let X be the set of the variables on which F depends, and let $a \in A$ and $r \in R$ be an assignment of boolean values and an assignment of decomposition types to X, respectively, that satisfies $M(a) = r$. Then,

$a \in A$ satisfies a clause C_i.

$\Leftrightarrow$

If we construct an SOKFDD(T_i) determining decomposition types for X according to r and those for the rest arbitrarily, then the number of nodes except for terminal nodes is less than or equal to 39,



$$
\begin{aligned}
f[v_1] &= G_i \\
f[v_2] &= H^i_{ABCD} \\
f[v_3] &= H^i_{EFGH}, \text{ or } H^i_{ABCD} \oplus H^i_{EFGH} \\
f[v_4] &= H^i_{AB} \\
f[v_5] &= H^i_{CD}, \text{ or } H^i_{AB} \oplus H^i_{CD} \\
f[v_6] &= H^i_{EF}, \text{ or } H^i_{AB} \oplus H^i_{EF} \\
f[v_7] &= H^i_{GH}, \text{ or } H^i_{EF} \oplus H^i_{GH}, \text{ or } H^i_{CD} \oplus H^i_{GH}, \text{ or } H^i_{AB} \oplus H^i_{CD} \oplus H^i_{EF} \oplus H^i_{GH} \\
f[v_8] &= g_{i\_000} \\
f[v_9] &= g_{i\_001}, \text{ or } g_{i\_002} \\
f[v_{10}] &= g_{i\_010}, \text{ or } g_{i\_020} \\
f[v_{11}] &= g_{i\_011}, \text{ or } g_{i\_012}, \text{ or } g_{i\_021}, \text{ or } g_{i\_022} \\
f[v_{12}] &= g_{i\_100}, \text{ or } g_{i\_200} \\
f[v_{13}] &= g_{i\_101}, \text{ or } g_{i\_102}, \text{ or } g_{i\_201}, \text{ or } g_{i\_202} \\
f[v_{14}] &= g_{i\_110}, \text{ or } g_{i\_120}, \text{ or } g_{i\_210}, \text{ or } g_{i\_220} \\
f[v_{15}] &= g_{i\_111}, \text{ or } g_{i\_112}, \text{ or } g_{i\_121}, \text{ or } g_{i\_122}, \text{ or } g_{i\_211}, \text{ or } g_{i\_212}, \text{ or } g_{i\_221}, \text{ or } g_{i\_222}
\end{aligned}
$$

Figure 3.5: The functions that are represented by the nodes of SOKFDD(T_i).

where $T_i = \{G_i, g_{i\_000}, \ldots, g_{i\_221}\}$. □

(Proof)

We illustrate the functions that are represented by the nodes of SOKFDD(T_i) in Figure 3.5, where $g_{i\_222} = a_i \oplus b_i \oplus c_i \oplus d_i \oplus e_i \oplus f_i \oplus g_i \oplus h_i$. It is obvious that the functions represented by $v_8, \ldots, v_{15}$ are different from each other by Figure 3.5. Thus there is one node with x_{i_1}, two nodes with x_{i_2} and four nodes with x_{i_3} in SOKFDD(T_i). Note that an SOKFDD($g_{i\_000}, \ldots, g_{i\_221}$) has exactly 32 nodes by Lemma 10. Therefore an SOKFDD(T_i) has at least 39 nodes. In the following, we show that the function represented by v_{15} in Figure 3.5 is g_{i_222}, that is, the number of nodes except for terminal nodes is more than 39, if and only if a does

not satisfy C_i.

(a) a does not satisfy C_i

For the literal l_{i_1} in C_i, if $l_{i_1} = x_{i_1}$ (resp. $l_{i_1} = \overline{x_{i_1}}$), then $x_{i_1} = 0$ (resp. $x_{i_1} = 1$) since a does not satisfy C_i. Thus the decomposition type of pD (resp. S) is assigned to x_{i_1} by the definition of $r = M(a)$. Therefore the function represented by v_3 is always $H^i_{ABCD} \oplus H^i_{EFGH}$, and does not depend on whether l_{i_1} is a positive literal or a negative literal. Similarly, the function represented by v_7 is always $H^i_{AB} \oplus H^i_{CD} \oplus H^i_{EF} \oplus H^i_{GH}$, and the function represented by v_{15} is always g_{i_222}, and each of them does not depend on whether l_{i_2} or l_{i_3} is a positive literals or not.

(b) The function represented by v_{15} is g_{i_222} for the assignment of decomposition types $r(= M(a))$.

The function represented by v_7 must be $H^i_{AB} \oplus H^i_{CD} \oplus H^i_{EF} \oplus H^i_{GH}$ since the function represented by v_{15} is g_{i_222}. In addition, for the literal l_{i_3} in C_i, when $l_{i_3} = x_{i_3}$ (resp. $l_{i_3} = \overline{x_{i_3}}$), the decomposition type of v_7 must be pD (resp. S). Thus the value of x_{i_3} in the assignment a is 0 (resp. 1) by the definition of $r = M(a)$. Therefore $l_{i_3} = 0$ and this does not depend on whether l_{i_3} is a positive literal or not. Similarly, $l_{i_2} = 0$ and $l_{i_3} = 0$, and a does not satisfy C_i. □

We obtain the following corollary by the proof of Lemma 11.

Corollary 5 *The following holds on the same condition as in Lemma 11.*

$a \in A$ satisfies a clause C_i.

$\Leftrightarrow$

If we construct an SOKFDD(T_i) determining decomposition types for X according to r and those for the rest arbitrarily, then the number of nodes except for terminal nodes is exactly 39.

where $T_i = \{G_i, g_{i\_000}, \ldots, g_{i\_221}\}$. □

We show the proof of Theorem 4 in the following.

(Proof of Theorem 4)

(a) Sufficient Condition

We assume that F is satisfiable. Then there is an assignment a of boolean values to X that satisfies F. We define an assignment r of decomposition types to X to be $r = M(a)$. We consider the SOKFDD(T) whose decomposition types are defined as follows:

- For the variables in X, we assign decomposition types according to r.
- For the remaining variables, we assign decomposition types arbitrarily.

We show that the number of nodes of the resulting SOKFDD(T) is less than or equal to K in the following.

The number of nodes of SOKFDD(T_i) is exactly 39 by Corollary 5 since a satisfies C_i. Note that, for any distinct i and j, T_i and T_j have no common variable. Thus only the terminal node can be shared by SOKFDD(T_i) and SOKFDD(T_j). Therefore the number of nodes of the SOKFDD(T) is the sum of the numbers of nodes of SOKFDD(T_1), ... , SOKFDD(T_n), plus 1, to which the terminal node of 0 contributes, and this is $39n + 1 = K$.

(b) Necessary Condition

We assume that F is unsatisfiable. Let r' be an assignment of decomposition types to the variables on which the functions in T depend. Let r be the assignment of decomposition types obtained by restricting r' to X. We define an assignment a of boolean values to X to be $a = M^{-1}(r)$. Then there exists an index i such that a does not satisfy C_i. Note that the number of nodes of SOKFDD(T_i) that is constructed according to r' is more than or equal to 40 by Lemma 11, and that, for any index j, the number of nodes of SOKFDD(T_j) that is constructed according to r' is at least 39 by Lemma 11 and Corollary 5. Similarly to the case in the proof of the sufficient condition, only the terminal node can be shared by SOKFDD(T_1), ..., SOKFDD(T_n). Thus the number of nodes of SOKFDD(T) that is constructed according to r' is at least $39(n-1) + 40 + 1 = K + 1$. □

3.4 Conclusion

In this chapter, we showed that the problem of determining whether there are decomposition types according to which the number of nodes of SOKFDD(T) can be less than or equal to K is NP-hard, where T is a set of boolean functions given as an SOBDD, K is a given constant, and the variable ordering is fixed. It is straightforward to see that it is still NP-hard if a set of boolean functions T is given as an SOKFDD with fixed decomposition types.

It remains to investigate NP-completeness of SOKFDD-MIN and complexity of SOKFDD-MIN with positive Davio expansion.

Chapter 4

On the power of Quantum Branching Programs

4.1 Introduction

Since Shor developed a polynomial time factoring algorithm for quantum computers[21], much attention has focused on quantum computation. There are many results that quantum computers might be more powerful than classical computers [15, 21], it is unclear whether there is a computational gap between the model that may use quantum effects and the model that may not. It has been shown that some quantum automaton models are more powerful than classical ones [4, 18]. It would give hints on the power of quantum computation to study about other computation models to see whether quantum computation models can be more powerful than classical ones.

As one of classical computation models, branching programs have been studied intensively as well as automaton models, and several types of branching programs are introduced including read-once branching programs and bounded-width branching programs [19].

In this chapter, we introduce a new quantum computation model, a quantum branching program, as an extension of a classical probabilistic branching program, and make comparison of the power of these two models. We show that, under a bounded-width restriction, ordered quantum branching programs can compute some function that ordered probabilistic branching programs cannot compute.

The remainder of this chapter has the following organization. In Section 4.2, we define several types of quantum branching programs and probabilistic branching programs. In Section 4.3, we show that, under a bounded-width restriction, ordered quantum branching programs can compute some function that ordered probabilistic branching programs cannot compute. We conclude with future work

in Section 4.4.

4.2 Preliminaries

We define technical terms.

Definition 4 *Probabilistic Branching Programs*

A probabilistic branching program (PBP) is a directed acyclic graph that has two terminal nodes, to which boolean values 0 and 1 are attached, and internal nodes, to which boolean variables taken from a set $X = \{x_1, \ldots, x_n\}$ *are attached. There is a distinguished node, called source, which has in-degree 0. Each internal node has two types of outgoing edges, called the 0-edges and the 1-edges respectively. Each edge e has a weight* $w(e)$ $(0 \leq w(e) \leq 1)$*. Let* $E_0(v)$ *and* $E_1(v)$ *be the set of the 0-edges and the set of the 1-edges of a node v respectively. The sum of the weights of the edges in* $E_0(v)$ *and* $E_1(v)$ *is 1. That is,*

$$\sum_{e \in E_0(v)} w(e) = 1, \sum_{e \in E_1(v)} w(e) = 1 \ .$$

A PBP reads n inputs and returns a boolean value as follows: Starting at the source, the value of the variable attached to the node is tested. If this is 0 (1), an edge in $E_0(v)$ $(E_1(v))$ *is chosen according to the probability distribution given as the weights of the edges. The next node that will be tested is the node pointed by the chosen edge. Arriving at the terminal node, the attached boolean value is returned.*

We say that a PBP P computes a function f (with error rate $1/2 - \delta$*) if P returns the correct value of f for any inputs with probability at least* $1/2 + \delta$ $(\delta > 0)$*.* □

We show examples of PBP's in Fig. 4.1.

Definition 5 *Quantum Branching Programs*

A quantum branching program (QBP) is an extension of a probabilistic branching program, and its form is same as a probabilistic branching program except for edge weights. In a QBP, the weight of each edge is a complex number $w(e)$ $(0 \leq ||w(e)|| \leq 1)$*. The sum of the squared magnitude of the weights of the edges in* $E_0(v)$ *and* $E_1(v)$ *is 1. That is,*

$$\sum_{e \in E_0(v)} ||w(e)||^2 = 1, \sum_{e \in E_1(v)} ||w(e)||^2 = 1 \ .$$

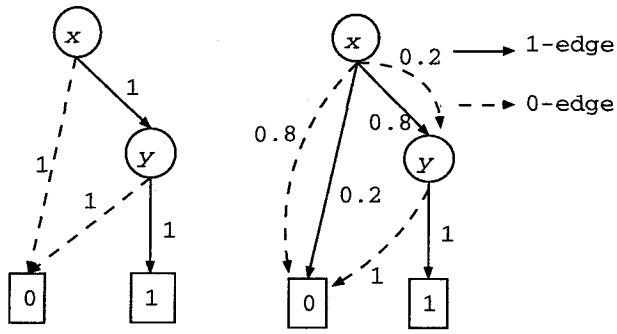


Figure 4.1: Probabilistic branching programs that compute $f = xy$ with error rate 0 and 0.2 respectively.

The edge weight $w(e)$ represents the amplitude with which, currently in the node v, the edge will be followed in the next step.

Nodes are divided into the three sets of the accepting set (Q_{acc}), the rejecting set (Q_{rej}) and the non-halting set (Q_{non}). The configurations of P are identified with the nodes in $Q = (Q_{\mathrm{acc}} \cup Q_{\mathrm{rej}} \cup Q_{\mathrm{non}})$. A superposition of a QBP P is any element of $l_2(Q)$ (the space of mappings from Q to $\mathbb{C}$ with l_2 norm). For each $q \in Q$, $|q\rangle$ denotes the unit vector that takes value 1 at q and 0 elsewhere.

Let $\mathbb{C}$ be the set of all complex numbers. We define a transition function $\delta : (Q \times \{0,1\} \times Q) \longrightarrow \mathbb{C}$ as follows:

$$\delta(v, a, v') = w(e) \ ,$$

where $w(e)$ is the weight of the a-edge ($a = 0$ or 1) from a node v to v'. If the a-edge from v to v' does not exist, then $\delta(v, a, v') = 0$. We define a time evolution operator as follows:

$$U_\delta^{\boldsymbol{x}} |v\rangle = \sum_{v' \in Q} \delta(v, x(v), v') |v'\rangle \ ,$$

where $\boldsymbol{x}$ denotes the input of a QBP, and $x(v)$ denotes the assigned value in $\boldsymbol{x}$ to the variable attached to the node v. If the time evolution operator is unitary, we say that the corresponding QBP is well-formed, that is, the QBP is valid in terms of the quantum theory.

It is required to have edges from terminal nodes in order to be well-formed. For convenience, we allow QBP's to have edges from terminal nodes and to be cyclic on the condition that it is still acyclic without the edges from terminal nodes.

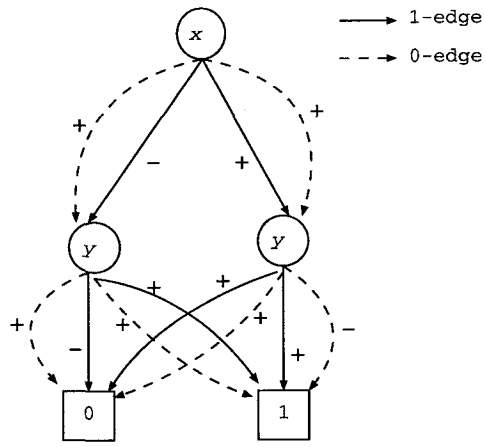


Figure 4.2: A quantum branching program that computes $f = x \oplus y$ with no error.

We define the observable $\mathcal{O}$ to be $E_{\mathrm{acc}} \oplus E_{\mathrm{rej}} \oplus E_{\mathrm{non}}$, where

$$\begin{aligned} E_{\mathrm{acc}} &= \mathrm{span}\{|v\rangle \,|v \in Q_{\mathrm{acc}}\}, \\ E_{\mathrm{rej}} &= \mathrm{span}\{|v\rangle \,|v \in Q_{\mathrm{rej}}\}, \\ E_{\mathrm{non}} &= \mathrm{span}\{|v\rangle \,|v \in Q_{\mathrm{non}}\}. \end{aligned}$$

A QBP reads n inputs and returns a boolean value as follows: The initial state $|\psi_0\rangle$ is the source $|v_{\mathrm{s}}\rangle$. At each step, the time evolution operator is applied to the state $|\psi_i\rangle$, that is, $|\psi_{i+1}\rangle = U_\delta^{\boldsymbol{x}} |\psi_i\rangle$. Next, $|\psi_{i+1}\rangle$ is observed with respect to $E_{\mathrm{acc}} \oplus E_{\mathrm{rej}} \oplus E_{\mathrm{non}}$. Note that this observation causes the quantum state $|\psi_{i+1}\rangle$ to be projected onto the subspace compatible with the observation. Let the outcomes of an observation be "accept", "reject" and "non-halting" corresponding to E_{acc}, E_{rej} and E_{non} respectively. Until "accept" or "reject" is observed, applying the time evolution operator and observation is repeated. If "accept" ("reject") is observed, boolean value 1 (0) is returned.

We say that a QBP P computes a function f (with error rate $1/2 - \delta$) if P returns the correct value of f for any inputs with probability at least $1/2 + \delta$ ($\delta > 0$). □

We show an example of a QBP in Fig. 4.2, where the weight of each edge is $\frac{1}{\sqrt{2}}$ or $-\frac{1}{\sqrt{2}}$, and only signs are put on the figure. To check well-formedness, we introduce the following theorem.

Theorem 5

A QBP P is well-formed.

$\Leftrightarrow$

For any input $\boldsymbol{x}$, the transition function δ satisfies the following condition.

$$\sum_{q'} \overline{\delta(q_1, x(q_1), q')}\delta(q_2, x(q_2), q') = \begin{cases} 1 & q_1 = q_2 \\ 0 & q_1 \neq q_2 \end{cases},$$

where $\overline{\delta(q, a, q')}$ denotes the conjugate of $\delta(q, a, q')$.

(Proof)

It is obvious since $U_\delta^{\boldsymbol{x}}$ is unitary if and only if the vectors $U_\delta^{\boldsymbol{x}} |v\rangle$ are orthonormal. □

Definition 6 *The Language Recognized by a Branching Program*

In this chapter, we define a language L to be a subset of $\{0,1\}^*$. Let the n-th restriction L^n of a language L be $L \cap \{0,1\}^n$. A sequence of branching programs $\{P_n\}$ recognizes a language L if and only if, there exists $\delta(> 0)$, and the n-input branching program P_n computes the characteristic function $f_{L^n}(\boldsymbol{x})$ of L^n with error rate at most $1/2 - \delta$ for all $n \in \mathbf{N}$, where

$$f_{L^n}(\boldsymbol{x}) = \begin{cases} 1 & (\boldsymbol{x} \in L_n) \\ 0 & (\boldsymbol{x} \notin L_n) \end{cases}.$$

□

Definition 7 *Bounded-Width Branching Programs*

For a branching program P, we can make any path from the source to a node v have the same length by inserting dummy nodes. Let the resulting branching program be P'. We say that P' is leveled. Note that P' does not need to compute the same function as P. The length of the path from the source to a node v is called the level of v. We define width(i) *for P' as follows:*

$$\text{width}(i) = |\{v | \textit{the level of } v \textit{ is } i.\}| \ .$$

We define Width(P') *as follows:*

$$\text{Width}(P') = \max_i \{\text{width}(i)\} \ .$$

We say that the width of P is bounded by Width(P').

A sequence of branching programs $\{P_n\}$ is a bounded-width branching program if, for some constant w, $\{P_n\}$ satisfy the following condition.

$\forall P \in \{P_n\}$, *The width of P is bounded by w .*

□

We also call a sequence of branching programs "a branching program" when it is not confused. We denote a bounded-width QBP and a bounded-width PBP as a bw-QBP and a bw-PBP respectively.

Definition 8 *Ordered Branching Programs*

Given a bounded-width branching program, we can make it leveled as shown in the above. For a given variable ordering $\pi = (x_{k_1} < x_{k_2} < \ldots < x_{k_n})$*, if the appearances of the variables obey the ordering* π*, that is,* x_{k_i} *precedes* x_{k_j} $(i < j)$ *on any path from the source to a terminal node, and the attached variables to all the nodes at the same level are the same, we say that the branching program is ordered.* □

4.3 Comparison of the Computational Power of Ordered bw-QBP's and Ordered bw-PBP's

In this section, we show that ordered bw-QBP's can compute some function that ordered bw-PBP's cannot compute. We define the function HALF_n and the language L_{HALF}.

Definition 9 *The Function* HALF_n *and the Language* L_{HALF}

We define $\mathrm{HALF}_n : \mathbf{B}^n \longrightarrow \mathbf{B}$ *as follows:*

$$\mathrm{HALF}_n(x_1, \ldots, x_n) = \begin{cases} 1 & |\{x_i | x_i = 1\}| = \frac{n}{2} \\ 0 & \textit{otherwise} \end{cases} .$$

In the following, we denote the variables on which HALF_n *depends as* $X = \{x_1, x_2, \ldots, x_n\}$*. We define* L_{HALF} *as follows:*

$$L_{\mathrm{HALF}} = \left\{ x \,\middle|\, x \in \{0,1\}^k, \mathrm{HALF}_k(x) = 1 \right\} .$$

□

4.3.1 Ordered bw-QBP's that Recognize L_{HALF}

In quantum computing, different computational paths interfere with each other when they reach the same configuration at the same time. In [18], a quantum finite automaton is constructed so that, only for inputs that the quantum finite automaton should accept, the computational paths interfere with each other.

In this chapter, we modify this technique for quantum branching programs, and construct a quantum branching program that recognizes the language L_{HALF}.

Theorem 6 *Ordered bw-QBP's can recognize* L_{HALF}.

(Proof)

To show that ordered bw-QBP's can recognize L_{HALF}, we construct an ordered bw-QBP that computes HALF_n for any n. Figure 4.3 illustrates the QBP.

We define the set of nodes Q as follows:

$$\begin{aligned} Q &= \{v_{\mathrm{s}}, v_1, v_2, v_3, v_{\mathrm{acc}}, v_{\mathrm{rej1}}, v_{\mathrm{rej2}}\} \\ &\quad \cup \{v_{(i,x_k)} | x_k \in X, 1 \le i \le 3\} \\ &\quad \cup \{v_{(i,x_k,j,\mathrm{T})} | x_k \in X, 1 \le i \le 3, 1 \le j \le i\} \\ &\quad \cup \{v_{(i,x_k,j,\mathrm{F})} | x_k \in X, 1 \le i \le 3, 1 \le j \le 3-i+1\} \ . \end{aligned}$$

The variable attached to the node $v_{(i,x_k)}$, $v_{(i,x_k,j,\mathrm{T})}$, and $v_{(i,x_k,j,\mathrm{F})}$ is x_k. The variable attached to the node v_{s} is x_1. The variable attached to the node v_1, v_2, and v_3 is x_n.

We define the accepting set (Q_{acc}), the rejecting set (Q_{rej}), the set of 0-edges (E_0), the set of 1-edges (E_1) and the weights of edges ($w(e)$) as follows:

$$Q_{\mathrm{acc}} = \{v_{\mathrm{acc}}\}, \quad Q_{\mathrm{rej}} = \{v_{\mathrm{rej1}}, v_{\mathrm{rej2}}\} \ .$$

$$\begin{aligned} E_0 &= \{(v_{\mathrm{s}}, v_{(i,x_1)}) | 1 \le i \le 3\} \\ &\quad \cup \{(v_{(i,x_k)}, v_{(i,x_k,1,\mathrm{F})}) | 1 \le i \le 3, 1 \le k \le n\} \\ &\quad \cup \{(v_{(i,x_k,j,\mathrm{F})}, v_{(i,x_k,j+1,\mathrm{F})}) | 1 \le i \le 3, 1 \le j \le 3-i, 1 \le k \le n\} \\ &\quad \cup \{(v_{(i,x_k,3-i+1,\mathrm{F})}, v_{(i,x_{k+1})}) | 1 \le i \le 3, 1 \le k \le n-1\} \\ &\quad \cup \{(v_{(i,x_n,3-i+1,\mathrm{F})}, v_i) | 1 \le i \le 3\} \\ &\quad \cup \{(v_i, v_{\mathrm{acc}}), (v_i, v_{\mathrm{rej1}}), (v_i, v_{\mathrm{rej2}}) | 1 \le i \le 3\} \ . \end{aligned}$$

$$\begin{aligned} E_1 &= \{(v_{\mathrm{s}}, v_{(i,x_1)}) | 1 \le i \le 3\} \\ &\quad \cup \{(v_{(i,x_k)}, v_{(i,x_k,1,\mathrm{T})}) | 1 \le i \le 3, 1 \le k \le n\} \\ &\quad \cup \{(v_{(i,x_k,j,\mathrm{T})}, v_{(i,x_k,j+1,\mathrm{T})}) | 1 \le i \le 3, 1 \le j \le i-1, 1 \le k \le n\} \\ &\quad \cup \{(v_{(i,x_k,i,\mathrm{T})}, v_{(i,x_{k+1})}) | 1 \le i \le 3, 1 \le k \le n-1\} \\ &\quad \cup \{(v_{(i,x_n,i,\mathrm{T})}, v_i) | 1 \le i \le 3\} \\ &\quad \cup \{(v_i, v_{\mathrm{acc}}), (v_i, v_{\mathrm{rej1}}), (v_i, v_{\mathrm{rej2}}) | 1 \le i \le 3\} \ . \end{aligned}$$

$$w((v_{\mathrm{s}}, v_{(1,x_1)})) = w((v_{\mathrm{s}}, v_{(2,x_1)})) = w((v_{\mathrm{s}}, v_{(3,x_1)})) = \frac{1}{\sqrt{3}} \ ,$$

$$w((v_1, v_{\mathrm{acc}})) = w((v_1, v_{\mathrm{rej1}})) = w((v_1, v_{\mathrm{rej2}})) = \frac{1}{\sqrt{3}} \ ,$$

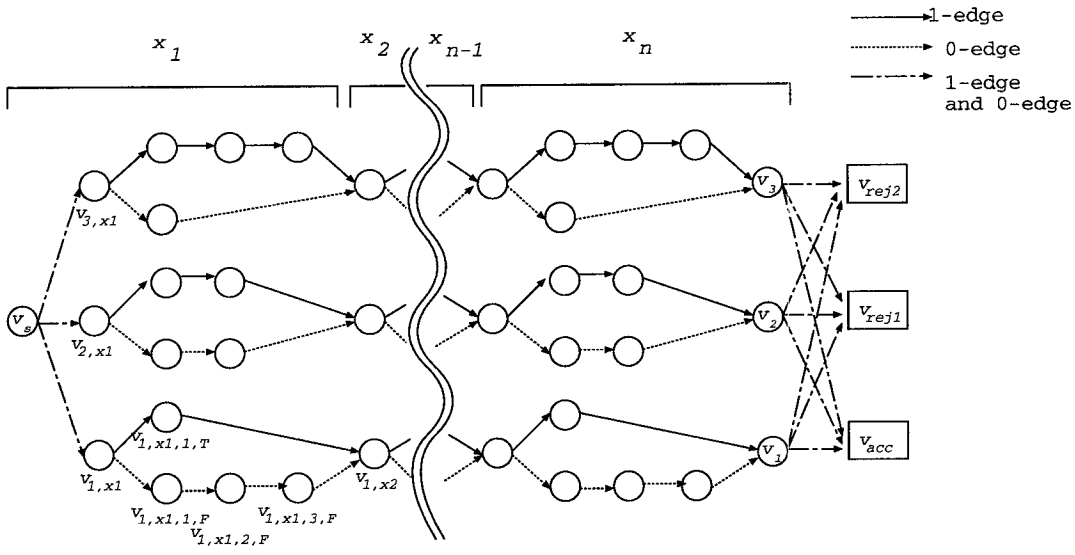


Figure 4.3: A QBP that computes HALF_n.

$$w((v_2, v_{\mathrm{acc}})) = \frac{1}{\sqrt{3}} \ , \ w((v_2, v_{\mathrm{rej1}})) = \frac{1}{\sqrt{3}} exp\left(\frac{2\pi i}{3}\right) \ ,$$
$$w((v_2, v_{\mathrm{rej2}})) = \frac{1}{\sqrt{3}} exp\left(\frac{4\pi i}{3}\right) \ , \ w((v_3, v_{\mathrm{acc}})) = \frac{1}{\sqrt{3}} \ ,$$
$$w((v_3, v_{\mathrm{rej1}})) = \frac{1}{\sqrt{3}} exp\left(\frac{4\pi i}{3}\right) \ , \ w((v_3, v_{\mathrm{rej2}})) = \frac{1}{\sqrt{3}} exp\left(\frac{8\pi i}{3}\right)$$

The weights of the other edges are all 1.

Adding some more nodes and edges, each node of the QBP can be made to have

[1] one incoming 0-edge and one incoming 1-edge with the weight of 1,

or,

[2] three incoming 0-edges and three incoming 1-edges with the same weights as between (v_1, v_2, v_3) and $(v_{\mathrm{acc}}, v_{\mathrm{rej1}}, v_{\mathrm{rej2}})$.

and also have

[1] one outgoing 0-edge and one outgoing 1-edge with the weight of 1,

or,

[2] three outgoing 0-edges and three outgoing 1-edges with the same weights as between (v_1, v_2, v_3) and $(v_{\mathrm{acc}}, v_{\mathrm{rej1}}, v_{\mathrm{rej2}})$.

In addition, incoming edges of each node can be made to be originated from the nodes to which the same variable attaches. Then it is straightforward to see that this QBP can be well-formed by Theorem 5.

Given an input $\boldsymbol{x}$, let the number of the variables in $X = \{x_1, \ldots, x_n\}$ to which the value 1 is assigned be k. Then the number of steps from $v_{(i,x_1)}$ to v_i is $ik + (3 - i + 1)(n - k) + n$. Thus for any two distinct i and j $(1 \leq i, j \leq 3)$, the number of steps from the source to v_i (v_j) is the same if and only if $k = n/2$. Therefore the superposition of this QBP becomes $\frac{1}{\sqrt{3}}|v_1\rangle + \frac{1}{\sqrt{3}}|v_2\rangle + \frac{1}{\sqrt{3}}|v_3\rangle$ after $3n + 1$ steps if $\mathrm{HALF}_n(\boldsymbol{x}) = 1$. Since $U_\delta^{\boldsymbol{x}}(\frac{1}{\sqrt{3}}|v_1\rangle + \frac{1}{\sqrt{3}}|v_2\rangle + \frac{1}{\sqrt{3}}|v_3\rangle) = |v_{\mathrm{acc}}\rangle$, this ordered bw-QBP returns 1 with probability 1 if $\mathrm{HALF}_n(\boldsymbol{x}) = 1$. On the other hand, since $U_\delta^{\boldsymbol{x}}|v_i\rangle = \frac{1}{\sqrt{3}}|v_{\mathrm{acc}}\rangle + \frac{e^{i\alpha}}{\sqrt{3}}|v_{\mathrm{rej1}}\rangle + \frac{e^{i\beta}}{\sqrt{3}}|v_{\mathrm{rej2}}\rangle$, this ordered bw-QBP returns 0 with probability 2/3 if $\mathrm{HALF}_n(\boldsymbol{x}) = 0$. Therefore this ordered bw-QBP computes HALF_n with one-sided error. □

4.3.2 Ordered bw-PBP's cannot Recognize L_{HALF}

Theorem 7 *Ordered bw-PBP's cannot recognize* L_{HALF}. □

To prove Theorem 7, we introduce the following definition and lemma.

Definition 10 *Total Variation Distance*

The total variation distance of two probability distributions P_1 *and* P_2 *over the same sample space* I *is defined as follows:*

$$\frac{1}{2}\sum_{i \in I} |P_1(i) - P_2(i)| \ .$$

Similarly, we define the total variation distance of two vectors $\boldsymbol{x}_1 = (a_1, \ldots, a_n)$ *and* $\boldsymbol{x}_2 = (b_1, \ldots, b_n)$ *$(a_1, \ldots, a_n, b_1, \ldots, b_n$: real numbers) as follows:*

$$\frac{1}{2}\sum_{1 \leq i \leq n} |a_i - b_i| \ .$$

□

Lemma 12 *Let* Γ^m *be the set that consists of all probability distributions of* m *events, that is,*

$$\Gamma^m = \{(a_1, \ldots, a_m) \,|\, a_1 \geq 0, \ldots, a_m \geq 0, a_1 + \ldots + a_m = 1\} \ .$$

For any constant δ *$(\delta > 0)$ there exists a natural number* N, *and for any finite set* $S \subseteq \Gamma^m$, *if the cardinality of* S *is greater than* N, *the following holds.*

$\exists D_1 \in S, D_2 \in S (D_1 \neq D_2)$
(the total variation distance of D_1 *and* $D_2) < \delta$

(Proof)

Γ^m is a bounded subset of a m-dimensional metric space whose distance between points is defined as the total variation distance. Thus Γ^m can be contained by some m-dimensional regular polyhedron whose volume is larger than Γ^m. We decompose the polyhedron into a finite number of smaller m-dimensional regular polyhedra whose lengths of edges are smaller than $\frac{2\delta}{m}$. Let the number of such smaller polyhedra be N. For any two points in such a smaller polyhedron, the total variation distance of them is less than δ. Thus if the cardinality of S is greater than N, there exist two distinct elements in S, say u and v, and the total variation distance of u and v is less than δ. □

We show the proof of Theorem 7 in the following.

(Proof of Theorem 7)

We assume that there is an ordered bw-PBP $\{P_n\}$ that recognizes L_{HALF}, that is, $P \in \{P_n\}$ computes HALF_n with error rate $1/2 - \delta$. We say that a PBP is in normal form when all the variables appear on any path from the source to a terminal node. We assume that P is in normal form with the ordering $\pi = (x_1 < \ldots < x_n)$ without loss of generality. Let $S_{\frac{n}{2}}$ be the set of the variables of the former half of the variable ordering, that is, $S_{\frac{n}{2}} = \{x_j | 1 \leq j \leq \frac{n}{2}\}$. When n is sufficiently large, there are sufficient number of assignments to the variables in $S_{\frac{n}{2}}$ such that, for any two distinct assignments, the weights of the assignments differ. Let D_a denotes the probability distribution for the nodes at which we arrive when we compute according to a. That is, $D_a(v)$ is the probability such that we arrive at the node v after we compute according to a. For sufficient number of assignments, there are sufficient number of corresponding probability distributions. Thus, since P is a bounded-width PBP, when n is sufficiently large, there are two distinct assignment, say a_1 and a_2, to the variables in $S_{\frac{n}{2}}$ satisfying the following conditions by Lemma 12.

- The total variation distance of D_{a_1} and D_{a_2} is less than δ.
- The weight of a_1 (the number of 1 in a_1) differs from that of a_2.

Let a_{rest} be the assignment to the variables of the latter half of the variable ordering such that, for the complete assignment $a_1 \cdot a_{rest}$, $\mathrm{HALF}_n(a_1 \cdot a_{rest}) = 1$. Then if we compute according to $a_1 \cdot a_{rest}$, we arrive at the terminal node of 1 with probability at least $1/2 + \delta$. On the other hand, if we compute according to $a_2 \cdot a_{rest}$, we arrive at the terminal node of 1 with probability at least $1/2 + \delta - \delta = 1/2$. We show the reason in the following.

Let I be

$$I = \{i | D_{a_1}(i) > D_{a_2}(i)\} .$$

Since the total variation distance of D_{a_1} and D_{a_2} is less than δ,

$$\sum_{i \in I} (D_{a_1}(i) - D_{a_2}(i)) \left(= \sum_{i \notin I} (D_{a_2}(i) - D_{a_1}(i)) \right) < \delta \ .$$

Thus comparing the probabilities with which we arrive at the terminal node of 1 computing according to $a_1 \cdot a_{rest}$ and $a_2 \cdot a_{rest}$, the difference of the probabilities is at most δ. Therefore we arrive at the terminal node of 1 with probability at least $1/2 + \delta - \delta = 1/2$ if we compute according to $a_2 \cdot a_{rest}$. However this probability must be less than $1/2 - \delta$. This is a contradiction. □

4.4 Conclusion

In this chapter, we show that there is a function that can be computed by ordered bw-QBP's but cannot be computed by ordered bw-PBP's. This is an evidence that introducing quantum effects to a computational model increases its power.

It is still the future work to study what results we obtain if we remove the restriction of bounded-width and variable ordering. Since quantum computational models must be reversible, introducing classical "reversible branching programs" and comparing them with quantum branching programs can also be future work.

Chapter 5

On the Power of Non-deterministic Quantum Finite Automata

5.1 Introduction

Recently, the power of quantum computation models has been investigated intensively, and many results such as Shor's polynomial-time factoring algorithm [21] and Grover's searching algorithm [15] have been proposed, which suggest that quantum computers might be more powerful than classical ones. However, it is still unclear how the computational power arises.

As quantum computation models, various kinds of quantum finite automata have been proposed, including 1-way and 2-way quantum finite automata. The power of those automata has been studied in [3, 4, 6, 18]. Quantum computers must be reversible since their state transition operator must be unitary. Because of this constraint, it is not always the case that quantum computation models are more powerful than classical counterparts. In fact, it has been shown that the class of languages recognized by 1-way quantum finite automata is a proper subset of the class of all regular languages since reversibility becomes critical for 1-way models [18].

On the other hand, the class NQP was proposed as the class of problems that are solvable by non-deterministic quantum Turing machines in polynomial time [1], and the relation to the class co-$C_{=}P$ has been shown in [13, 14, 23].

As for classical finite automata, the capabilities of deterministic finite automata and non-deterministic finite automata are the same in terms of accepting languages. In this chapter, we investigate whether non-determinism makes (1-way) quantum finite automata more powerful or not. We introduce (1-way)

non-deterministic quantum finite automata in which the same non-determinism as in non-deterministic quantum Turing machines is applied. That is, if for an input word the probability of outputting 1 is 0, the word is rejected, otherwise, it is accepted.

We prove that a non-regular language L_{EQ}, which is described later, can be recognized by non-deterministic quantum finite automata, and also show that any regular language can be recognized by non-deterministic quantum finite automata. In other words, non-deterministic quantum finite automata are strictly more powerful than classical deterministic/non-deterministic finite automata, and also strictly more powerful than 1-way quantum finite automata.

These results mean that the non-determinism introduced to quantum finite automata certainly increases their capabilities in terms of accepting languages. The results also imply that, as for quantum finite automata, non-deterministic 1-way models can be more powerful than classical counterparts in spite of the restriction of reversible state transitions.

As a similar model, unbounded-error measure-many quantum finite automata have been studied in [6]. Our model can be regarded as a subclass of the model, in which its 'cut-point' is restricted to zero. In [6], inclusion of regular languages has not been shown. The above result of ours relating to inclusion of regular languages is shown for models which are considered to be less powerful than unbounded-error measure-many quantum finite automata. Thus, it is implied that unbounded-error measure-many quantum finite automata are powerful enough to accept any regular language.

This chapter is organized as follows: Section 5.2 defines the non-deterministic quantum finite automata. Section 5.3 describes the main results. Section 5.4 concludes this chapter.

5.2 Non-Deterministic Quantum Finite Automata

We define several types of finite automata in the following.

Definition 11 (DFA) *A deterministic Finite Automaton (DFA) is defined by the following 5-tuple:*

$$M = (Q, \Sigma, \delta, q_0, Q_f),$$

where Q is the set of states, Σ is the set of input symbols, δ is the state transition function ($\delta : (Q \times \Sigma \times Q) \longrightarrow \{0, 1\}$), q_0 is the initial state, and Q_f is the set of accepting states. □

$\delta(q, a, q') = 1$ (0) means that the state changes (does not change) from q to q' when reading an input symbol a. For any q and a, there is exactly one q' that satisfies $\delta(q, a, q') = 1$. For an input word w, M reads the input symbol one by one, and states change as follows:

- Let the current state be q and the input symbol be a. The state changes from q to q', where $\delta(q, a, q') = 1$.

If the final state (i.e., the state after reading all the input symbols) is in the set of the accepting states Q_f, we say that M accepts the word w, otherwise, we say that M rejects the word w.

Definition 12 (NQFA) *A non-deterministic Quantum Finite Automaton (NQFA) is defined by the following 8-tuple:*

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Q_{acc}, Q_{rej}, Q_{non}),$$

where Q is the set of states, Σ is the set of input symbols, $\Gamma = \Sigma \cup \{¢, \$\}$ ($¢$ and $\$$ are the left and the right end-marker, respectively) is the set of tape symbols, δ is the state transition function ($\delta : (Q \times \Sigma \times Q) \longrightarrow \mathbb{C}$), q_0 is the initial state, Q_{acc} is the set of accepting states, Q_{rej} is the set of rejecting states, $Q_{non} = Q \setminus (Q_{acc} \cup Q_{rej})$ is the set of non-halting states, and $Q_{acc} \cap Q_{rej} = \emptyset$, where $\mathbb{C}$ is the set of all complex numbers. □

$\delta(q, a, q') = \alpha$ means that the amplitude of the transition from q to q' when reading a is α. Since configurations of an NQFA are described by only its states, we identify a configuration of an NQFA with its state. A superposition of configurations in NQFA M is any element of $l_2(Q)$ of unit length. For each configuration $q \in Q$, we define a column vector $|q\rangle$ as follows:

- $|q\rangle$ is a $|Q| \times 1$ column vector.
- The row corresponding to q is 1, and the other rows are 0.

For an input symbol a, we define a time evolution operator U_a as follows:

$$U_a(|q\rangle) = \sum_{q' \in Q} \delta(q, a, q') \, |q'\rangle \, .$$

If U_a is unitary for any $a \in \Sigma$, that is, $U_a^\dagger U_a = I$, then we say that the corresponding NQFA is well-formed. This means that the NQFA is considered to be valid in terms of the quantum theory. We consider only well-formed NQFA's in the following.

We define the observable $\mathcal{O} = E_{\text{non}} \oplus E_{\text{acc}} \oplus E_{\text{rej}}$ as follows:

$$\begin{aligned} E_{\text{non}} &= span\{|q\rangle \,|q \in Q_{\text{non}}\}, \\ E_{\text{acc}} &= span\{|q\rangle \,|q \in Q_{\text{acc}}\}, \\ E_{\text{rej}} &= span\{|q\rangle \,|q \in Q_{\text{rej}}\}. \end{aligned}$$

We define the outcomes of an observation to be "non", "acc" and "rej" corresponding to E_{non}, E_{acc} and E_{rej} respectively.

We define the notion of "words accepted by an NQFA M" as follows.

Let the initial state be q_0. We define $|\psi_0\rangle = |q_0\rangle$. We operate as follows:

(a) We define $|\psi_{i+1}\rangle$ to be $|\psi_{i+1}\rangle = U_a |\psi_i\rangle$ for the i-th input a.

(b) We observe $|\psi_{i+1}\rangle$ with respect to the observable $\mathcal{O}$. Note that this observation causes $|\psi_{i+1}\rangle$ to be projected onto the subspace compatible with the observation. If the outcome is "acc", then the output of the NQFA is 1. If the outcome is "rej", then the output is 0. If the outcome is "non", then repeat (a).

We call the above (a) and (b) 'one step' collectively. For a word w, if the probability of outputting 1 is not 0, we say that the NQFA accepts the word w, otherwise, we say that it rejects the word w.

The set of words accepted by a finite automata (deterministic or not) is the *language* recognized by the finite automata.

As an unbounded error quantum computation model, unbounded error measure-many quantum finite automata (MM-QFA's) are introduced in [6]. An unbounded error MM-QFA is said to accept a language L with cut-point λ if for all $x \in L$ the probability of M outputting 1 is greater than λ and for all $x \notin L$ the probability of M outputting 1 is at most λ. An NQFA is considered to be a special case of an unbounded-error MM-QFA such that $\lambda = 0$. Thus the languages recognized by NQFA's are also recognized by unbounded-error MM-QFA's, but the converse is not certain.

To check well-formedness, we introduce the following lemma.

Lemma 13 *NQFA M is well-formed. $\Leftrightarrow$ The state transition function satisfies the following condition:*

$$\sum_{q'} \overline{\delta(q', a, q_1)} \delta(q', a, q_2) = \begin{cases} 1 & (q_1 = q_2) \\ 0 & (q_1 \neq q_2) \end{cases},$$

where $\overline{\delta(\cdot, \cdot, \cdot)}$ is the conjugate of $\delta(\cdot, \cdot, \cdot)$.

(Proof)

A matrix U_a is unitary if and only if $U_a U_a^\dagger = I$, where $U_a^\dagger$ is a transpose conjugate of U_a. (i,j)-element of $U_a U_a^\dagger$ is $\sum_{q'} \delta(q', a, q_i)\overline{\delta(q', a, q_j)}$. Thus, it is obvious that the lemma holds. □

5.3 NQFA's and Regular Languages

In this section, we show that the class of languages recognized by NQFA's properly contains the class of all regular languages.

5.3.1 An NQFA that Recognizes the Language L_{EQ}.

We define the language L_{EQ} as follows:

$$L_{\mathrm{EQ}} = \{a,b\}^* \setminus \left\{ w \;\middle|\; \begin{array}{l} w \in \{a,b\}^*, \\ \text{The number of } a \text{ in } w \\ \text{is equal to the number} \\ \text{of } b \text{ in } w. \end{array} \right\}$$

It is obvious that language L_{EQ} is not a regular language. We show that NQFA's can recognize L_{EQ}.

In [3, 6], it has been shown that languages similar to L_{EQ} can be recognized by variants of quantum finite automata. Based on those models, we can also make out quantum finite automata which accepts L_{EQ}. Furthermore, by modifying the definition of "acceptance" appropriately, they can be regarded as NQFA's. In this sense, the following theorem is straightforward. To keep the chapter self-contained, we show the proof of the theorem.

Theorem 8 *NQFA's can recognize language L_{EQ}.*

(Proof)

The NQFA with the following set of states and state transition function recognizes the language L_{EQ}.

$$\begin{aligned} Q &= \{q_0, q_1, q_{\mathrm{acc}}, q_{\mathrm{rej}}\} \\ \delta(q_0, a, q_0) &= \cos\sqrt{2}\pi \\ \delta(q_0, a, q_1) &= \sin\sqrt{2}\pi \end{aligned}$$

$$
\begin{aligned}
\delta(q_1, a, q_0) &= -\sin\sqrt{2}\pi \\
\delta(q_1, a, q_1) &= \cos\sqrt{2}\pi \\
\delta(q_0, b, q_0) &= \cos(-\sqrt{2}\pi) \\
\delta(q_0, b, q_1) &= \sin(-\sqrt{2}\pi) \\
\delta(q_1, b, q_0) &= -\sin(-\sqrt{2}\pi) \\
\delta(q_1, b, q_1) &= \cos(-\sqrt{2}\pi) \\
\delta(q_0, \text{¢}, q_0) &= 1 \\
\delta(q_1, \text{¢}, q_1) &= 1 \\
\delta(q_0, \$, q_{\text{rej}}) &= 1 \\
\delta(q_1, \$, q_{\text{acc}}) &= 1 \\
\delta(q_{\text{rej}}, a, q_{\text{rej}}) &= 1 \\
\delta(q_{\text{acc}}, a, q_{\text{acc}}) &= 1 \\
\delta(q_{\text{rej}}, b, q_{\text{rej}}) &= 1 \\
\delta(q_{\text{acc}}, b, q_{\text{acc}}) &= 1 \\
\delta(q_{\text{rej}}, \$, q_0) &= 1 \\
\delta(q_{\text{acc}}, \$, q_1) &= 1 \\
\delta(q_{\text{rej}}, \text{¢}, q_{\text{rej}}) &= 1 \\
\delta(q_{\text{acc}}, \text{¢}, q_{\text{acc}}) &= 1
\end{aligned}
$$

It is obvious that the NQFA is well-formed by Lemma 13.

Let the initial state be $|q_0\rangle$. The state is rotated by $\sqrt{2}\pi$ in the two dimensional space that is spanned by $|q_0\rangle$ and $|q_1\rangle$ if the input symbol is a, and it is rotated by $-\sqrt{2}\pi$ if the input symbol is b. Thus, the superposition contains exactly one configuration $|q_0\rangle$ after reading the word w if and only if the number of a in w is equal to the number of b in w. If the number of a in w differs from the number of b in w, the superposition becomes $\alpha|q_0\rangle + \beta|q_1\rangle$ ($\beta \neq 0$). Therefore, this NQFA recognizes the language L_{EQ}. □

We discuss the accuracy of the amplitudes of the NQFA in the following. We obtain the following lemma from Lemma 6 in [3].

Lemma 14 *The NQFA in Theorem 8 outputs 1 for $w \in L_{EQ}$ with probability at least $\frac{1}{2|w|^2}$.* □

By Lemma 14, we can say that the accuracy (i.e., the number of bits) needed to represent the amplitudes is at most polynomial in terms of input lengths.

5.3.2 Recognition of Regular Languages by NQFA's

We show that, for any regular language, there is an NQFA that recognizes the language.

Theorem 9 *For any DFA, there is an NQFA that recognizes the language recognized by the DFA.*

(Proof)

Let $M = (Q, \Sigma, \delta, q_0, Q_f)$ be an arbitrary DFA. We are to make an NQFA

$$M^\star = (Q^\star, \Sigma, \Gamma, \delta^\star, q_0, Q^\star_{\rm acc}, Q^\star_{\rm rej}, Q^\star_{\rm non})$$

that recognizes the language recognized by M.

First, we will define the set of states $Q^\star$. Let $Q_m \triangleq Q \setminus Q_f$.

For each $q \in Q$ and $a \in \Sigma$, let $S(q,a) \triangleq \{q' | \delta(q', a, q) = 1\}$, that is, the set of origins of incoming transitions to q. And for each $a \in \Sigma$, let $D_a \triangleq \{q \in Q \mid |S(q,a)| \geq 2\}$, and let $N_a \triangleq \{q \in Q | S(q,a) = \emptyset\}$.

Using these sets, we define $Q^\star$ as follows.

$Q^\star \triangleq (Q_f \cup Q_m \cup \cup_{a\in\Sigma} \cup_{q\in D_a} R_{aq}) \cup (\tilde{Q_f} \cup \tilde{Q_m} \cup \cup_{a\in\Sigma} \cup_{q\in D_a} \tilde{R_{aq}})$, where R_{aq}'s, $\tilde{R_{aq}}$'s, $\tilde{Q_f}$ and $\tilde{Q_m}$ are mutually disjoint sets of new states (they are all disjoint from Q) such that $|\tilde{Q_f}| = |Q_f|$, $|\tilde{Q_m}| = |Q_m|$, and $|R_{aq}| = |\tilde{R_{aq}}| = |S(q,a)| - 1$ for $a \in \Sigma, q \in D_a$.

As the cardinalities are the same, we can arbitrarily fix one to one correspondence between Q_f and $\tilde{Q_f}$, also between Q_m and $\tilde{Q_m}$, and between R_{aq} and $\tilde{R_{aq}}$ for $a \in \Sigma, q \in D_a$. For $q \in Q_f$, we will denote the corresponding state in $\tilde{Q_f}$ by $\tilde{q}$. Similar convention will be used for Q_m and R_{aq}'s.

For each R_{aq}, we arbitrarily fix an ordering of states in R_{aq}, which will be used in defining a transition function later. For $a \in \Sigma$, we define $T_a \triangleq \cup_{q\in D_a} R_{aq}$.

As each state in M has exactly one destination state in the transition defined by $\delta(\cdot, a, \cdot)$, it is easy to see that $|T_a| = |N_a|$. For each $a \in \Sigma$, we arbitrarily define a bijection $mate_a(\cdot)$ from T_a to N_a.

Now we will define $\delta^\star$ as follows. In the definition, as is stated before, $\tilde{q}$ denotes the state corresponding to q.

$$\delta^\star(q, a, mate_a(q)) = 1 \ (q \in T_a, a \in \Sigma),$$

$$\delta^\star(q, b, q) = 1 \ (q \in T_a, a \in \Sigma, b \in \Sigma \setminus \{a\}),$$

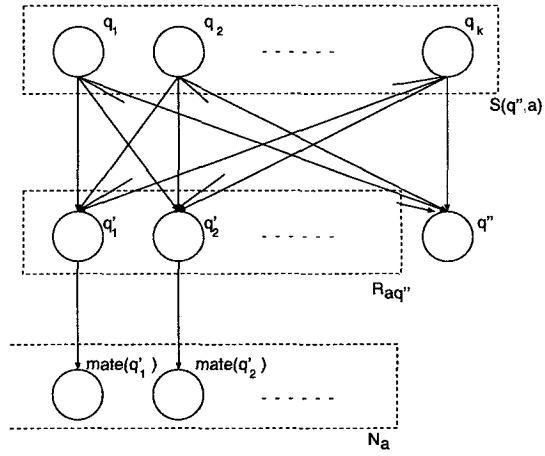


Figure 5.1: Definition of $\delta^\star(\cdot, a, \cdot)$ around R_{aq}.

$$\delta^\star(q, \$, \tilde{q}) = 1 \ (q \in Q_f \cup Q_m \cup \cup_a T_a),$$

$$\delta^\star(\tilde{q}, \$, q) = 1 \ (q \in Q_f \cup Q_m \cup \cup_a T_a),$$

$$\delta^\star(q, a, q) = 1 \ (a \in \Sigma, q \in \tilde{Q}_f \cup \tilde{Q}_m \cup \cup_b \tilde{T}_b),$$

$$\delta^\star(q, \not{\epsilon}, q) = 1 \ (q \in Q^\star),$$

$$\delta^\star(q, a, q') = 1 \ \left(\begin{array}{l} S(q', a) = \{q\}, a \in \Sigma, \\ q' \in Q \setminus (D_a \cup N_a) \end{array} \right),$$

$$\delta^\star(q, a, q') = \frac{1}{\sqrt{k}} exp(\frac{2\pi i}{k} st) \left(\begin{array}{l} q \in S(q'', a), \\ q' \in R_{aq''} \cup \{q''\}, \\ q'' \in D_a \end{array} \right),$$

where $k \stackrel{\triangle}{=} |S(q'', a)| \geq 2$ and q is the s-th state in $S(q'', a)$ and q' the t-th state in $R_{aq''} \cup \{q''\}$ and q'' the last (i.e., the k-th) state in $R_{aq''} \cup \{q''\}$ (we assume fixed orderings in $S(q'', a)$ and in $R_{aq''} \cup \{q''\}$).
And for $q, q' \in Q^\star$ and $a \in \Sigma \cup \{\not{\epsilon}, \$\}$, for which $\delta^\star(q, a, q')$ is not thus far defined, $\delta^\star(q, a, q') = 0$.

We illustrate the definition of $\delta^\star$ in Figure 5.1 and Figure 5.2.

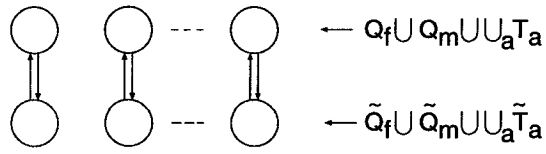


Figure 5.2: Definition of $\delta^\star(\cdot, \$, \cdot)$ between q and $\tilde{q}$.

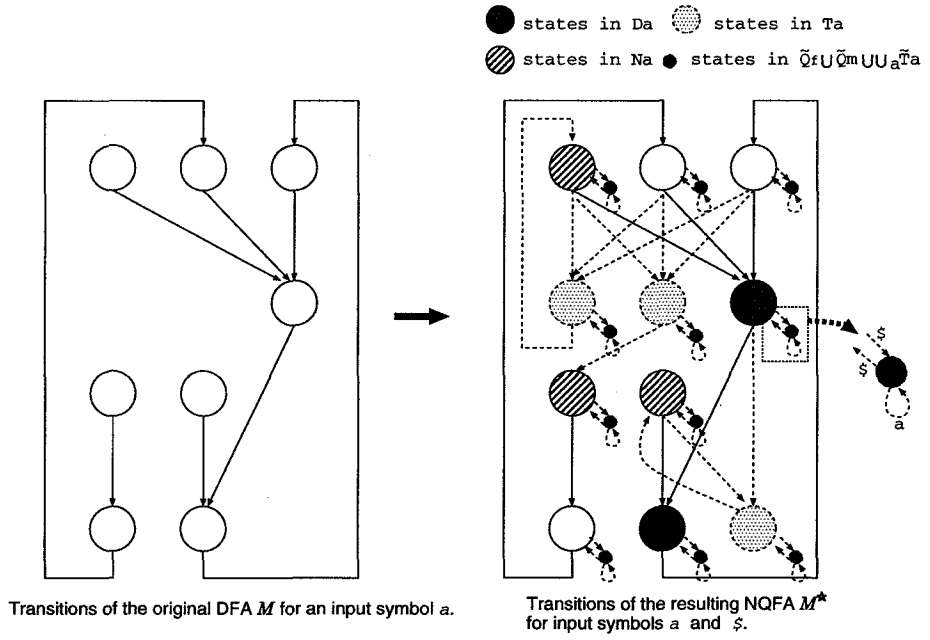


Figure 5.3: An example of M and corresponding $M^\star$.

Now we define $Q^\star_{\rm non}$, $Q^\star_{\rm acc}$, and $Q^\star_{\rm rej}$ as follows:

$$\begin{aligned} Q^\star_{\rm acc} &\triangleq \tilde{Q}_f \\ Q^\star_{\rm rej} &\triangleq \cup_a T_a \cup_a \tilde{T}_a \cup (\tilde{Q} \setminus Q^\star_{\rm acc}), \\ Q^\star_{\rm non} &\triangleq Q^\star \setminus (Q^\star_{\rm acc} \cup Q^\star_{\rm rej}) \ \ (= Q). \end{aligned}$$

We define $M^\star$ to be

$$M^\star = (Q^\star, \Sigma, \Gamma, \delta^\star, q_0, Q^\star_{\rm acc}, Q^\star_{\rm rej}, Q^\star_{\rm non}),$$

where $\Gamma = \Sigma \cup \{¢, \$\}$. We illustrate an example of M and corresponding $M^\star$ in Figure 5.3.

Remark: We have replaced the transition from $S(q,a)$ to q in M by a so-called quantum Fourier transform from $S(q,a)$ to $R_{aq} \cup \{q\}$ (in case $|S(q,a)| \geq 2$).

We claim that $M^\star$ recognizes the language recognized by M and that $M^\star$ is well-formed.

Well-formedness of the Resulting NQFA

As the cases for $a \in \{¢, \$\}$ are clear, let $a \in \Sigma$ be arbitrarily fixed in the following.

Let $S^\star(q, a)$ be defined on $M^\star$ similarly to the way $S(q, a)$ was defined on M, that is, $S^\star(q, a) \triangleq \{u \in Q^\star \mid \delta^\star(u, a, q) \neq 0\}$.

It is not hard to see that $|S^\star(q, a)| \geq 2$ if $q \in T_a \cup D_a$, and $|S^\star(q, a)| = 1$ otherwise. This implies that $S^\star(q, a) \neq \emptyset$ for any $q \in Q^\star$. Also it is not hard to see that for any $q_1, q_2 \in Q^\star$, either $S^\star(q_1, a) = S^\star(q_2, a)$ or $S^\star(q_1, a) \cap S^\star(q_2, a) = \emptyset$.

We check the value of

$$\begin{aligned} & \sum_{q'} \overline{\delta^\star(q', a, q_1)} \delta^\star(q', a, q_2) \\ = & \sum_{q' \in S^\star(q_1, a) \cap S^\star(q_2, a)} \overline{\delta^\star(q', a, q_1)} \delta^\star(q', a, q_2). \end{aligned}$$

First, we assume that $q_1 = q_2$.

Case 1 : $|S^\star(q_1, a)| = 1$

For $q' \in S^\star(q_1, a)$, by the definition of $\delta^\star$, $\delta^\star(q', a, q_1) = 1$. Thus,

$$\begin{aligned} & \sum_{q'} \overline{\delta^\star(q', a, q_1)} \delta^\star(q', a, q_1) \\ = & \sum_{q' \in S^\star(q_1, a)} \overline{\delta^\star(q', a, q_1)} \delta^\star(q', a, q_1) \\ = & \ 1. \end{aligned}$$

Case 2 : $|S^\star(q_1, a)| \geq 2$

In this case, there is a state $q'' \in D_a$ such that $S^\star(q_1, a) = S(q'', a)$ and $q_1 \in R_{aq''} \cup \{q''\}$.

For $q' \in S^\star(q_1, a) = S(q'', a)$, $\delta^\star(q', a, q_1) = \frac{1}{\sqrt{k}} exp(\frac{2\pi i}{k} cj)$, where $k = |S^\star(q_1, a)|$ and $j, c \in \{1, 2, \ldots, k\}$ are such that q' is the j-th state in $S^\star(q_1, a) = S(q'', a)$ and q_1 is the c-th state in $R_{aq''} \cup \{q''\}$. Thus,

$$\begin{aligned} & \sum_{q' \in S^\star(q_1, a)} \overline{\delta^\star(q', a, q_1)} \delta^\star(q', a, q_1) \\ = & \sum_{q' \in S(q'', a)} \overline{\delta^\star(q', a, q_1)} \delta^\star(q', a, q_1) \\ = & \ \frac{1}{k} \cdot k = 1 \end{aligned}$$

Next, we assume that $q_1 \neq q_2$.

Case 1 : $S^\star(q_1, a) \neq S^\star(q_2, a)$

In this case, $S^\star(q_1, a) \cap S^\star(q_2, a) = \emptyset$. Thus, $\sum_{q'} \overline{\delta^\star(q', a, q_1)}\delta^\star(q', a, q_2) = 0$.

Case 2 : $S^\star(q_1, a) = S^\star(q_2, a)$

In this case, there exists a state $q'' \in D_a$ such that $q_1, q_2 \in R_{aq''} \cup \{q''\}$ and $S^\star(q_1, a) = S^\star(q_2, a) = S(q'', a)$. Thus

$$\begin{aligned}
& \sum_{q'} \overline{\delta^\star(q', a, q_1)}\delta^\star(q', a, q_2) \\
= & \sum_{q' \in S(q'', a)} \overline{\delta^\star(q', a, q_1)}\delta^\star(q', a, q_2) \\
= & \sum_{1 \leq j \leq k} \overline{\frac{1}{\sqrt{k}} exp\left(\frac{2\pi i}{k} j c_{q_1}\right)} \frac{1}{\sqrt{k}} exp(\frac{2\pi i}{k} j c_{q_2}) \\
= & \sum_{1 \leq j \leq k} \frac{1}{k} exp\left(\frac{2\pi i}{k}(c_{q_2} - c_{q_1})j\right) \\
= & \ 0,
\end{aligned}$$

where $k = |S^\star(q_1, a)|$, and c_{q_1} and c_{q_2} are non identical integers determined by q_1 and q_2, respectively.

Therefore, the NQFA is well-formed by Lemma 13.

Recognition of the language

We show that $M^\star$ recognizes the language recognized by M in the following.

The outcomes of observations for $M^\star$ are never "acc" before reading the right end-marker \$. Thus, the outcome is only "rej" or "non" when it reads a symbol in $\Gamma \setminus \{\$\}$. We consider one step from $|q\rangle$ reading $a \in \Gamma \setminus \{\$\}$, where $q \in Q^\star_{\text{non}}$. The probability of obtaining "non" after the step is not 0, and if "non" is obtained, then the state collapses to the state $|q'\rangle$ that consists of a single configuration q', where $\delta(q, a, q') = 1$. Thus, the probability of having only "non" (as outcomes) until reading the right end-marker \$ is not 0, and in such a case, the sequence of the state transitions of $M^\star$ is the same as that of M for the same input word. Therefore, when M reaches one of the accepting states (Q_f) after reading an input word w, $M^\star$ reaches the same state after reading w with probability greater than 0. When M does not reach any of the accepting states after reading w, $M^\star$ ends up with "rej" as the outcome in the middle of reading w or reaches one of the states in $Q \setminus Q_f$ after reading w.[1] Therefore, $M^\star$ recognizes the language recognized by M. □

[1]The sequence of the state transitions is the same as that of M for w.

By Theorem 8 and Theorem 9, we can say that the class of languages recognizable by NQFA's properly contains the class of all regular languages.

5.4 Conclusion

In this chapter, we have introduced (1-way) non-deterministic quantum finite automata, and have compared them with classical finite automata. As a result, we have shown that the class of languages recognized by NQFA's properly contains the class of all regular languages. This means that non-deterministic quantum finite automata are more powerful than classical non-deterministic finite automata, and also they are more powerful than 1-way quantum finite automata.

It remains to compare NQFA's with push down automata, and to study whether NQFA's can be more compact than classical finite automata.

Chapter 6

Conclusions

In this dissertation, we investigated expressive power of several decision diagrams and quantum computation models. In Chapter 2, we showed that lower bounds on the size of binary moment diagrams representing division are exponential. This follows known experimental results.

In Chapter 3, we showed that the problem of finding the best decomposition type list for shared OKFDD's is NP-hard. This means that even if shared OKFDD's can represent given functions efficiently, it might be time consuming to find such small representations.

Decision diagrams are used in various area, and it is required to represent functions as small as possible. Therefore it is important to investigate expressive power of decision diagrams. It remains to investigate upper/lower bounds on the size of other decision diagrams.

In Chapter 4 and 5, we investigated the expressive power of quantum computation models. It has been shown that quantum computation models have much more expressive power than classical computation models. We showed that there is some function that can be computed by bounded-width ordered quantum branching programs, but cannot be computed by classical counterparts. We also showed that non-deterministic quantum finite automata can be strictly more powerful than classical deterministic/non-deterministic finite automata. This result also means that non-deterministic quantum finite automata is strictly more powerful than 1-way quantum finite automata since 1-way quantum finite automata can recognize a proper subset of regular languages.

Recently quantum computers has attracted much attention, and many results have been shown suggesting that quantum computers can be more powerful than classical ones. However, it is still unclear what kind of problems are suitable for quantum computers solving. By investigating expressive power of quantum computation models, we might answer the question what is the secret to exploit

quantum effects in computing. It remains to investigate other quantum computation models and to extract common properties that make quantum computation models more powerful.

Bibliography

[1] L. M. Adleman, J. DeMarrais, and M. A. Huang, "Quantum computability," SIAM J. Comput., 26 (1997), 1524–1540.

[2] S.B. Akers, "Binary decision diagrams," IEEE Trans. Comput., vol.C-27, no.6, pp.509–516, 1978.

[3] A. Ambainis and J. Watrous, "Two-way finite automata with quantum and classical states," LANL e-print cs.CC/9911009, 1999.

[4] A. Ambainis and R. Freivalds, "1-way quantum finite automata: strengths, weakness and generalizations," Proc. 39th Symp. on Foundations of Computer Science, pp.332–341, 1998.

[5] B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs is NP-complete," IEEE Trans. Comput., vol.45, No.9 , pp.993–1002, Sept. 1996.

[6] A. Brodsky and N. Pippenger, "Characterizations of 1-way quantum finite automata," LANL e-print quantu-ph/9903014, 1999.

[7] R.E. Bryant, "Graph-based algorithms for boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677–691, 1986.

[8] R.E. Bryant, "On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication," IEEE Trans. Comput., vol.40, no.2, pp.205–213, 1991.

[9] R.E. Bryant and Y.-A. Chen, "Verification of arithmetic circuits with binary moment diagrams," Proc. 32nd Design Automation Conf., pp.535–541, 1995.

[10] Y.-A. Chen and R.E. Bryant, "ACV: An arithmetic circuit verifier," Proc. ICCAD-96, pp.361–365, 1996.

[11] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer," Proc. R. Soc. Lond. A, Math. Phys. Eng. Sci., vol.400, pp.96–117, 1985.

[12] R. Drechsler and B. Becker, "Ordered kronecker functional decision diagrams – a data structure for representation and manipulation of boolean functions," IEEE Trans. Comput.-Aided Des. Integrated Circuits & Syst., vol. 17, No. 10, pp. 965–973, Oct. 1998.

[13] S. Fenner, F. Green, S. Homer, and R. Pruim, "Determining acceptance possibility for a quantum computation is hard for the polynomial hierarchy," Proc. R. Soc. Lond. A, Math. Phys. Eng. Sci., vol.455, pp.3953–3966, 1999.

[14] L. Fortnow and J. Rogers, "Complexity limitations on quantum computation," J. Comput. Syst. Sci., vol.59(2), pp.240–252, 1999.

[15] L. Grover, "A fast quantum mechanical algorithm for database search," Proc. 28th Symp. on the Theory of Computing, pp.212–219, 1996.

[16] K. Hamaguchi, A. Morita, and S. Yajima, "Efficient construction of binary moment diagrams for verifying arithmetic circuits," Proc. ICCAD-95, pp.78–82, 1995.

[17] T. Horiyama and S. Yajima, "Exponential lower bounds on the size of OBDDs representing integer division," Proc. 8th International Symposium on Algorithms and Computation, pp.163–172, 1997.

[18] A. Kondacs and J. Watrous, "On the power of quantum finite state automata," Proc. 38th Symp. on Foundations of Computer Science, pp.66–75, 1997.

[19] C. Meinel, "Modified branching programs and their computational power," Lecture Notes in Computer Science 370, Springer-Verlag, Berlin, 1989.

[20] C. Moore and J. P. Crutchfield, "Quantum Automata and Quantum Grammars," Theoretical Computer Science, vol.237, pp.275–306, 2000.

[21] P. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," SIAM J. Comput., 26 (1997), 1484–1509.

[22] S. Tani, K. Hamaguchi, and S. Yajima, "The complexity of the optimal variable ordering problems of shared binary decision diagrams," Proc. International Symposium on Algorithms and Computation, pp.389–398, 1993.

[23] T. Yamakami and A. C. Yao, "NQP_C = co-$C_=P$," Inf. Process. Lett., vol.71 (2), pp.63–69, 1999.

[24] A C. Yao, "Quantum circuit complexity," Proc. 34th Symp. on Foundations of Computer Science, pp.352–361, 1993.