数理解析研究所講究録
第 754 巻 1991 年 65-71

A Class of Logic Functions Expressible by Polynomial-Size Binary Decision Diagrams
多項式サイズの二分決定グラフで表現可能な論理関数のクラス

Nagisa ISHIURA and Shuzo YAJIMA
石浦菜岐佐　　矢島脩三

Department of Information Science, Faculty of Engineering, Kyoto University
京都大学工学部情報工学教室

Abstract

In this paper we discuss properties of logic functions expressible by BDD's of feasible size. We define a class of logic functions expressible by BDD's whose size (number of nodes) are bounded by a polynomial of the number of input variables. We derive some properties of this class through the discussion on the relation between polynomial-size BDD's and Turing machines. We also focus on the relation between polynomial-size BDD's and combinational circuits and show that polynomial size BDD's can be synthesized into $O(\log^2 n)$ depth combinational circuits.

1 Introduction

A binary decision diagram (BDD) [Bry86] is one of representation forms of logic functions. It is widely used in application programs for logic design verification, test generation and logic synthesis, owing to its properties that there exists a unique canonical form for each logic function and that many practical logic functions are expressible by BDD's of feasible size. However, in the worst case the size of the BDD of a logic function is known to be exponential of the number of input variables. There are few discussions on a problem of what kind of logic functions are expressible by BDD's of feasible size [Bry90]. In this paper we focus on this point so as to clarify the efficiency and limitation of application programs using logic function manipulation based on BDD's. We define a class of logic functions expressible by BDD's whose size (the number of nodes) is bounded by a polynomial of the number of input variables. We derive some properties of this class through the discussion on the relation between polynomial-size BDD's and *Turing machines*. We also focus on the relation between polynomial-size BDD's and combinational circuits and show an upper bound of the depth of combinational circuits that realize logic functions expressed by polynomial-size BDD's. Main results of this paper are as follows:

(1) Let *PolyBDD* be a class of logic functions expressible by uniform BDD's whose size is bounded by a polynomial of n, where n is the number of the input variables. Let *LOGIREG* be a class of logic functions which is computable by an LSIA, where an LSIA is a one-way off-line Turing machine which has $O(\log n)$ bounded working tape and has an ability to know n without reading the input tape. We have shown *PolyBDD* = *LOGIREG*.

(2) From the properties of LSIA's we derive that symmetric functions, threshold functions and selector functions etc. are in *PolyBDD*.

(3) *PolyBDD* is shown to be included by *DLOGSPACE* directly from the definition of an LSIA. Since it is know that *DLOGSPACE* is included by NC^2 (a class of logic functions realized by uniform combinational circuits of depth $O(\log^2 n)$), we can conclude that logic functions

expressed by polynomial-size BDD's can be synthesized into $O(\log^2 n)$ depth combinational circuits. We also show a concrete procedure of constructing an $O(\log^2 n)$ depth combinational circuit from a polynomial size BDD.

2 Family of Binary Decision Diagrams

2.1 Binary Decision Diagram (BDD)

We define a *binary decision diagram (BDD)* over $\mathcal{B} = \{0, 1\}$ as follows.

Def 2.1 A binary decision diagram over $\mathcal{B}$ is a 6-tuple $B = (X, N, s, l, e^0, e^1)$, where
$X = \{x_1, x_2, \cdots, x_n\}$ is a *totally ordered set of variables*, where
$x_1 < x_2 < \cdots < x_n$ holds,
N is a set of *nodes*,
$s \in N$ is the *initial node*,
$l : N \rightarrow (X \cup \mathcal{B})$ represents the *label* of a node,
$e^0, e^1 : N \rightarrow N$ represents a set of *0-edges and 1-edges*, respectively, where
$\forall v \in N$ s. t. $l(v) \in X : l(e^0(v)) \in \mathcal{B}$ or $l(v) < l(e^0(v))$, and
$l(e^1(v)) \in \mathcal{B}$ or $l(v) < l(e^1(v))$, and
$\forall v \in N$ s. t. $l(v) \in \mathcal{B} : e^0(v) = e^1(v) = v$. □

The set N is divided into two subsets; N_V and N_R, where $N_V = \{v \mid v \in N, l(v) \in X\}$ and $N_R = \{v \mid v \in N, l(v) \in \mathcal{B}\}$. ($N = N_V \cup N_R$ and $N_V \cap N_R = \phi$). A node in N_V is called a *variable node* and a node in N_R is called a *value node*. N_R consists of just two nodes; r_0 and r_1, where $l(r_0) = 0$ and $l(r_1) = 1$. We call r_0 and r_1 the *0-node* and the *1-node*, respectively. We denote the index of a variable x_i in X as $\iota(x_i)$. Namely $\iota(x_i) = i$. A pair of nodes $(v, e^0(v))$ and $(v, e^1(v))$ are called the *0-edge* and *1-edge* of node v. The *size* of BDD B, denoted as size(B), is defined as the number of nodes of B. Namely, size(B) $= |N|$, where $|N|$ is the number of elements in set N.

Let A be a set of assignments for X, where assignment a for X is a vector in $\mathcal{B}^{|X|}$. We denote the i-th element of a as a_i, and call it an assignment to variable x_i (the i-th variable). For a given assignment a, we define $T(a)$ which represents the set of nodes reachable from s under the assignment a.

$$v \in T(a) \text{ iff } v = s \text{ or } \exists u \in T(a) \text{ s. t. } e^{a_{\iota(l(u))}}(u) = v.$$

The output value of the logic function represented by a BDD is defined using this set of reachable nodes.

Def 2.2 We define the following f_B as the logic function expressed by BDD B:

$$f_B(a) = 1 \text{ iff } r_1 \in T(a). \qquad \square$$

Next, we define a *description* of a BDD. We assume that each element e in X and N has an identifier which is denoted as id(e). Let us call 4-tuple D_v = (id(v), id($l(v)$), id($e^0(v)$), id($e^1(v)$)) a *description* of node $v \in$. We can describe all the information of a BDD as the set of descriptions of the nodes of the BDD.

Def. 2.3 We define the D_B as the *node set description* of BDD B,where $D_B = \{D_v \mid v \in N\}$. □

2.2 BDD Family and Its Uniformity

In order to discuss the size of BDD's with respect to the number of input variables, we define a family of BDD's.

Def 2.4 BDD family $\{B_n\}$ is a sequence of BDD's $B_1, B_2, B_3, \cdots$, where $|X_n| = n$ holds for each $B_n = (X_n, N_n, s_n, l_n, e^0{}_n, e^1{}_n)$. □

Let $\{f_n\}$ be a sequence of logic functions where $f_n : \mathcal{B}^n \rightarrow \mathcal{B}$ (an n-variable logic function). We can consider that $\{f_n\}$ expresses a language L over $\mathcal{B}$ by the following correspondence:

$$b_1 b_2 \cdots b_n \in L \text{ iff } f_n(b_1, b_2, \cdots, b_n) = 1.$$

Similarly we define a language for a BDD family.

Def 2.5 The language accepted by BDD family $\{B_n\}$ is defined as follows and denoted as $L_{\{B_n\}}$.

$$b_1 b_2 \cdots b_n \in L_{\{B_n\}} \text{ iff } f_{B_n}(b_1, b_2, \cdots, b_n) = 1. \qquad \square$$

In this paper, we discuss correspondence between BDD families and *Turing machines.* For this purpose we define *uniformity* of a BDD family following after the uniformity of a combinational circuit family [Ruz81].

Def 2.6 BDD family $\{B_n\}$ is uniform if the description of the n-th BDD B_n can be generated from a binary representation of n by an $O(\log \mathrm{size}(B_n))$ space bounded off-line Turing machine. □

As a class of languages accepted by a BDD family of feasible size, we define a class of *PolyBDD.*

Def 2.7 *PolyBDD* is a class of languages accepted by a uniform BDD family $\{B_n\}$, which satisfies $\mathrm{size}(B_n) \leq \mathrm{poly}(n)$, where $\mathrm{poly}(n)$ is a polynomial of n. □

3 Relation between Polynomial Size BDD's and Log-Space Automata

3.1 Log-Space Automaton

We will refer a one-way off-line Turing machine with $O(\log n)$ bounded working tape as a *log space automaton (LSA).* An input to an LSA is given on its input tape which is *read-only.* An LSA can read the symbols on the input tape *only once in the given order* (one-way). Instead, an LSA can read and write the working tape. Namely, an LSA can be regarded as an automaton provided with $O(\log n)$ working tape. We also define an abstract machine referred to as a *log space input-size-look-ahead automata (LSIA):* An LSIA is an LSA which has an ability to know the length of a given input sequence without scanning the input sequence. The length of an input sequence is given as the initial value on the working tape in binary representation.

Def 3.1 *LOGREG* and *LOGIREG* are classes of languages which can be accepted by an LSA and an LSIA, respectively. □

The main result of this paper is that *PolyBDD* is equivalent to *LOGIREG*.

Th 1 *PolyBDD* = *LOGIREG*.

($LOGIREG \subseteq PolyBDD$):
Since an LSIA has only $O(\log n)$ memory, all the states of an LSIA can be represented by p nodes where $p \leq \text{poly}(n)$. Then using $p \times n$ nodes we can construct a state transition diagram without loop, which is the very n-th BDD. Since transitions are computable by an $O(\log n)$ space bounded Turing machine, the BDD family is uniform.

($PolyBDD \subseteq LOGIREG$):
Using an $O(\log n)$ working tape, an LSIA can simulate the $O(\log n)$ space bounded Turing machine to generate a member of a uniform BDD family. In other words an LSIA can compute the next node of a node for a given assignment. Since the number of nodes in a BDD is bounded by a polynomial of n, $O(\log n)$ space is enough to reach final node from the initial node s. □

3.2 Properties of *PolyBDD*

We can lead properties of the logic functions represented by polynomial size BDD's directly from properties of LSIA's.

Let *REG* and *DLOGSPACE* be classes of languages which can be accepted by a finite automaton and a log-space Turing machine (an off-line Turing machine with $O(\log n)$ bounded working tape), respectively. Obviously $REG \subseteq LOGIREG \subseteq DLOGSPACE$. The difference between *REG* and *LOGIREG* is due to the working tape of an LSIA and the difference between *LOGIREG* and *DLOGSPACE* is due to the restriction that an LSIA can read the input tape only once. Logic functions which can be computed by a sequential machine with constant number of registers, such as *parity* and *carry*, are expressible by polynomial size BDD's. An LSA has an $O(\log n)$ working tape besides a finite control. Using this working tape, an LSA can accept more complex languages.

(1) $\{0^n 1^n\}$ belongs to *PolyBDD*.
With the $O(\log n)$ working tape, an LSA can count and compare the number of 0's and 1's in a given sequence.

(2) All the symmetric functions belong to *PolyBDD*. The output of a symmetric function depends only on the number of 1's in the inputs. Then it is computable using the $O(\log n)$ working tape.

(3) Threshold functions belong to *PolyBDD* if the magnitude of each weight is bounded by a polynomial of n.

(4) Let $\text{int}(w)$ be the integer value of binary representation w, $\text{weight}(\sigma)$ be the number of 1's in sequence σ, and $\sigma[k]$ be the k-th alphabet of σ. Then the selector function $\{w\sigma \mid |w| = \lceil \log|\sigma| \rceil, \sigma[|w| + \text{int}\ (w) + 1] = 1\}$ belongs to *PolyBDD*.

The essence of the above properties is that an LSIA can *count number* up to $\text{poly}(n)$ using the $O(\log n)$ working tape. It can count the position or the number of 1's in a given sequences. However it can not memorize a whole input sequence itself because that requires working tape of length $O(n)$.

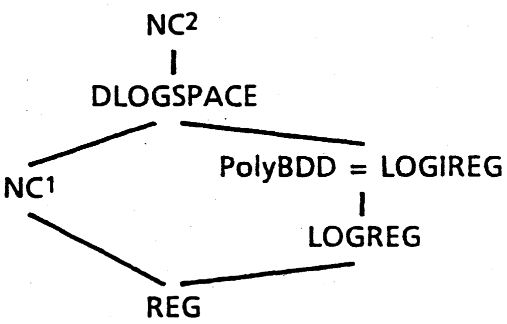

Figure 1: Relations among classes.

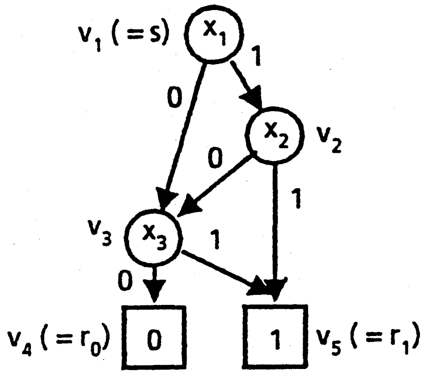

Figure 2: A BDD.

(5) $\{ww \mid w \in \mathcal{B}^*\}$ and $\{ww^R \mid w \in \mathcal{B}^*\}$ (where w^R is a *reversed sequence* of w) does not belong to *PolyBDD*.
In order to accept ww by scanning an input sequence *only once*, the first half of the sequence must be memorized, which requires an $O(n)$ working tape.

(6) Let $x_1x_2\cdots x_k \circ y_1y_2\cdots y_k = x_1y_1x_2y_2\cdots x_ky_k$ where $x_i, y_i \in \mathcal{B}$. Then the shift function defined as $\{sw \mid w = (x_1x_2\cdots x_k)\circ(0^{\mathrm{int}(s)}x_1x_2\cdots x_{k-\mathrm{int}(s)})\}$ does not belong to *PolyBDD*.

Another example is that the selector function in (4) does not belong to *PolyBDD* if we define the selector function as $\{\sigma w \mid \cdots\}$. This is also an example to show that the size of the BDD representing the same function can vary depending on ordering of input variables.

The property predicts that it may be difficult to express logic functions computed by *sequential circuits* even if the logic functions of their combinational part are expressible by BDD's of feasible size. If an output of a sequential circuit depends on input sequences of length $O(n)$ and the number of its registers is larger than $O(\log n)$, there can be cases where the sequential function does not belong to *PolyBDD*even if the combinational function belongs to *PolyBDD*.

4 Relation between BDD's and Combinational Circuits

We also investigated the relation between *PolyBDD* and other classes related to combinational circuits. Figure 1 is the summary of relation among the classes. NC^k is a class of logic functions which can be expressed by a uniform family of combinational circuits of depth $O(\log^k n)$ and size $O(\mathrm{poly}(n))$ under fan-in restriction [Coo85]. Since *PolyBDD* is included by *DLOGSPACE* and *DLOGSPACE* is included by NC^2, we can conclude that logic functions expressed by polynomial size BDD's can be synthesized into polynomial size and $O(\log^2 n)$ depth combinational circuits. We show a constructive proof.

As is formalized in section 2, the function represented by a BDD is defined as the reachability problem on the BDD. We will construct a combinational circuit which solves the reachability problem. For a given BDD B we define $|N| \times |N|$ matrix $A_B = [a_{i,j}]$ as follows. Intuitively $a_{i,j}$ becomes 1 if node v_j is directly reachable from node v_i under a given assignment. We call A_B the *adjacency matrix* of B.

$$
\begin{aligned}
&a_{i,j} : \mathcal{B}^n \to \mathcal{B}, \text{ where} \\
&\quad a_{i,j} = \overline{x_k} \text{ if } l(v_i) = x_k, e^0(v_i) = v_j, \\
&\quad a_{i,j} = x_k \text{ if } l(v_i) = x_k, e^1(v_i) = v_j,
\end{aligned}
$$

$$a_{i,i} = 1 \quad \text{if } l(v_i) \in R,$$
$$a_{i,j} = 0 \quad \text{otherwise.}$$

For example, the adjacency matrix of the BDD in Figure 2 is

$$A_B = \begin{bmatrix} 0 & x_1 & \overline{x_1} & 0 & 0 \\ 0 & 0 & \overline{x_2} & 0 & x_2 \\ 0 & 0 & 0 & \overline{x_3} & x_3 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Let us denote the (i,j)-element of $A_B{}^n$ (the n-th power of A_B) as $a^n{}_{i,j}$ and let $v_s = s$ and $v_r = r_1$. Then $f_B \equiv a^n{}_{s,r}$ by definition. We will show how to construct a combinational circuit of depth $O(\log^2 n)$ which computes $A_B{}^n$. A combinational circuit which computes A_B can be realized according to the above definition. Multiplication of $m \times m$ Boolean matrix is computable by a combinational circuit of depth $O(\log m)$ and size $O(m^3)$. By constructing a tree of multiplication circuits we can compute the n-th power of A_B. Since the depth of the tree is $\lceil \log n \rceil$, the total depth of the circuit is $O(\log n \log m)$. If the size of the BDD is bounded by a polynomial of n, it comes to $O(\log^2 n)$.

As for the relation between *PolyBDD* and NC^1 we have not yet obtained significant results. There exists a logic function family which belongs to NC^1 but not to *PolyBDD*. Integer multiplication belongs to NC^1 but is known to require exponential nodes in BDD representation [Bry90]. Therefore, there are two possibility; *PolyBDD* $\subset NC^1$, or *PolyBDD* and NC^1 are incomparable. This problem has a significant importance on the synthesis of multilevel circuits because polynomial size BDD's can be synthesized into $O(\log n)$ depth combinational circuits if *PolyBDD* $\subset NC^1$.

5 Conclusion

We have defined a class of logic functions expressible by polynomial-size BDD's and have investigated its properties through discussions on relation between log-space automata. We also discussed the relationship between BDD's and combinational circuits and showed a concrete procedure to synthesize $O(\log^2 n)$ depth combinational circuits form polynomial size BDD's. It remains as a future work to clarify the relation between *PolyBDD* and NC^1.

6 Acknowledgments

Authors would like to express their sincere appreciation to Prof. H. Yasuura, Prof. K. Iwama, Mr. T. Tohdo, Mr. Y. Okabe, Mr. S. Hirose and all the members of Yajima Lab. at Kyoto University for their discussions and valuable comments.

References

[Bry86] R. E. Bryant: "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, (1986).

[Bry90] R. E. Bryant: "On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication", *private communication*, (to appear in *IEEE Transactions on Computers*).

[Ruz81] Ruzzo: "On uniform circuit complexity", *JCSS* 22, pp. 365–383, (1981).

[Coo85] S. A. Cook: "Taxonomy of problems with fast parallel algorithms", *Information and Control* 64, pp. 2–22, (1985).