数理解析研究所講究録
第1148巻 2000年 94-99

Recognizing Ordered Tree-Shellable Boolean Functions Based on OBDDs

Yasuhiko TAKENAGA (武永 康彦)

Department of Computer Science,
The University of Electro-Communications

Abstract In this paper, we consider the complexity of recognizing ordered tree-shellable Boolean functions when Boolean functions are given as OBDDs. An ordered tree-shellable function is a positive Boolean function such that the number of prime implicants equals the number of paths from the root node to a 1-node in its ordered binary decision tree representation. We show that given an OBDD, it is possible to recognize in quadratic time a function ordered tree-shellable with respect to the variable ordering of the OBDD.

1 Introduction

A tree-shellable function is a positive Boolean function defined by the relation between its prime implicants and binary decision tree (BDT) representation: the number of prime implicants equals the number of paths from the root to a leaf labeled 1 in its binary decision tree representation [6]. An ordered tree-shellable function is a special case of a tree-shellable function and its prime implicants have the similar relation with an ordered BDT. In this paper, we deal with the complexity of recognizing ordered tree-shellable functions.

An ordered tree-shellable function has the following good properties. First, if a Boolean function is shellable, one can easily solve the union of product problem [2], which is the problem of computing the reliability of some kind of systems. Second, if a Boolean function is tree-shellable, it is easy to compute its dual.

When a Boolean function is given as its DNF representation, it is NP-complete to check if the function is ordered tree-shellable [3]. If a variable ordering π is given, it is possible to check if the function is ordered tree-shellable with respect to π within polynomial time.

In this paper, we consider the case when a Boolean function is given as its Ordered Binary Decision Diagram (OBDD) representation. An OBDD [1, 4] is a directed acyclic graph that represents a Boolean function. As OBDDs are widely used in many applications due to their good properties, it is worth considering the case when an OBDD is given as an input of recognition problems [5]. We show that it is possible to check if the function is ordered tree-shellable with respect to the variable ordering of the given OBDD in quadratic time.

2 Basic Definitions

Let $B = \{0, 1\}$, n be a natural number, and $[n] = \{1, 2, ..., n\}$. Especially, $[0] = \emptyset$. Let π be a permutation on $[n]$. π represents a total order of integers.

Let $f(x_1, ..., x_n)$ be a Boolean function. We denote $f \geq g$ if $f(x) = 1$ for any assignment $x \in \{0, 1\}^n$ which makes $g(x) = 1$. An *implicant* of f is a product term $\bigwedge_{i \in I} x_i \bigwedge_{j \in J} \overline{x_j}$ which satisfy $\bigwedge_{i \in I} x_i \bigwedge_{j \in J} \overline{x_j} \leq f$, where $I, J \subseteq [n]$. An implicant which satisfies $\bigwedge_{i \in I - \{s\}} x_i \bigwedge_{j \in J} \overline{x_j} \not\leq f$ for any $s \in I$ and $\bigwedge_{i \in I} x_i \bigwedge_{j \in J - \{t\}} \overline{x_j} \not\leq f$ for any $t \in J$ is called a *prime implicant* of f.

An expression of the form $f = \bigvee_{k=1}^{m} \left(\bigwedge_{i \in I_k} x_i \bigwedge_{j \in J_k} \overline{x_j} \right)$ is called a *disjunctive normal form Boolean formula* (DNF), where $I_k, J_k \subseteq [n]$ and $I_k \cap J_k = \emptyset$ for $k = 1, ..., m$. A *positive* DNF (PDNF) is a DNF such that $J_k = \emptyset$ for all k. If f can be represented as a PDNF, it is called a positive Boolean function. A PDNF is called *irredundant* if $I_k \subseteq I_l$ is not satisfied for any k, l $(1 \leq k, l \leq m, k \neq l)$. For an irredundant PDNF, let $PI(f)$ be the set of all I_k. $PI(f)$ represents the prime implicants of f. In the following of this paper, we consider only positive functions and we assume that a function is given as an irredundant PDNF $f = \bigvee_{k=1}^{m} \bigwedge_{i \in I_k} x_i$.

3 Graph Representations of Boolean Functions

3.1 Binary Decision Tree

A *Binary Decision Tree* (BDT) is a labeled tree that represents a Boolean function. A leaf node of a BDT is labeled by 0 or 1 and called a value node. Any other node is labeled by a variable and called a variable node. Let *label*(v) be the label of node v. Each node except leaf nodes has two outgoing edges, which are called a *0-edge* and a *1-edge*. Let $edge_0(v), edge_1(v)$ denote the nodes pointed to by the 0-edge and the 1-edge of node v respectively. The value of the function is given by traversing from the root node to a leaf node.

A path from the root node to a leaf node labeled 1 is called a *1-path*. A path P of a BDT is represented as a sequence of literals. If the k-th edge on a 1-path P is the 1-edge (0-edge, resp.) from the node labeled by x_i, positive literal x_i (negative literal $\overline{x_i}$, resp.) is the k-th element of P. For simplicity, we denote $\tilde{x}_i \in P$ when $\tilde{x}_i$ is included in the sequence representing P, where $\tilde{x}_i$ is either x_i or $\overline{x_i}$. Let $pos(P_k)$ ($neg(P_k)$, resp.) be the set of indices of variables whose positive (negative, resp.) literals are in P_k.

When the 0-edge and the 1-edge of node v point to the nodes representing the same function, v is called to be a *redundant node*. In the following of this paper, we assume that a BDT has no redundant node.

If there is a total order of variables which is consistent with the order that variables appear on any path from the root to a leaf, it is called an *ordered* BDT (OBDT). The total order of variables for an OBDT is called the *variable ordering.* If $label(v)$ is the k-th element of the variable ordering, we say that k is the level of v and denote $level(v) = k$. Let the level of value node be $n + 1$.

3.2 Ordered Binary Decision Diagram

An *Ordered Binary Decision Diagram* (OBDD) [1, 4] is a directed acyclic graph that represents a Boolean function. Intuitively, an OBDD is obtained by combining the nodes of an OBDT which represent the same function into a single node. The nodes of an OBDD consist of *variable nodes* and two *value nodes.* Similarly to an OBDT, there is a total ordering of variables for an OBDD, which is called a *variable ordering.*

When two nodes i and j have the same label and represent the same function, they are called *equivalent nodes.* When $edge_1(i) = edge_0(i)$, node i is called a *redundant node.* An OBDD which has no equivalent nodes and no redundant nodes is called a *reduced* OBDD. It is known that a Boolean function is uniquely represented by a reduced OBDD, provided that the variable ordering is fixed. In the following of this paper, we assume w.l.o.g. that an OBDD means a reduced OBDD. The size of an OBDD is the total number of nodes.

4 Ordered Tree-Shellable Boolean Function

Definition A positive Boolean function f is *tree-shellable* when it can be represented by a BDT with exactly $|PI(f)|$ 1-paths.

Definition A positive Boolean function f is *ordered tree-shellable* with respect to π if it can be represented by an OBDT with variable ordering π which has exactly $|PI(f)|$ 1-paths. f is ordered tree-shellable if there exists π such that f is ordered tree-shellable with respect to π. We call π to be the *shelling variable ordering* of f.

Proposition 1 If $f = \bigvee_{k=1}^{m} \bigwedge_{i \in I_k} x_i$ is tree-shellable, there exists a BDT T representing f which satisfy the following conditions.

- T has m 1-paths $P_1, ..., P_m$.
- Each P_k corresponds to a term I_k by the rule that $i \in I_k$ iff $x_i \in P_k$.

As an ordered tree-shellable function is tree-shellable, Proposition 1 also holds for ordered tree-shellable functions.

The next corollary is clear from the proof of Theorem 4 of [6].

Corollary 2 Let T be an OBDT with variable ordering π that represents a Boolean function f. f is ordered tree-shellable with respect to π iff there exists I_t which satisfy $I_t \subsetneq pos(P_i) \cup \{l\}$ for any 1-path P_i of T and any $\overline{x_l} \in P_i$,

5 Checking Ordered Tree-Shellability Based on OBDDs

Theorem 3 Given an OBDD with variable ordering π, it is possible to check if the Boolean function represented by the OBDD is ordered tree-shellable with respect to π or not within polynomial time.

Proof We first give the polynomial time algorithm to check ordered tree-shellability. Let $lev(u,v) = \min\{level(u), level(v)\}$. In this algorithm, 0 represents the value node labeled 0.

[Algorithm CheckOTS]

```
1. Check if the OBDD represents a positive function. If not, it is not ordered tree-
   shellable.

2. For $i = 1$ to $n$, repeat (a) and (b).

   (a) For any node $v$ in level $i$ do:
       if $edge_0(v) \neq 0$, $A_{lev(edge_0(v),edge_1(v))} = A_{lev(edge_0(v),edge_1(v))} \cup \{(edge_0(v), edge_1(v))\}$.
   (b) For any pair $(u,v) \in A_i$ do:
         if $level(u) > i$
          $A_{lev(u,edge_0(v))} = A_{lev(u,edge_0(v))} \cup \{(u, edge_0(v))\}$
         else if $level(v) > i$
          if $edge_0(u) \neq 0$, $A_{lev(edge_0(u),v)} = A_{lev(edge_0(u),v)} \cup \{(edge_0(u), v)\}$
         else do:
          $A_{lev(edge_1(u),edge_1(v))} = A_{(edge_1(u),edge_1(v))} \cup \{(edge_1(u), edge_1(v))\}$
          if $edge_0(u) \neq 0$, $A_{lev(edge_0(u),edge_0(v))} = A_{(edge_0(u),edge_0(v))} \cup \{(edge_0(u), edge_0(v))\}$

3. The given OBDD represents an ordered tree-shellable function iff no pair of the form
   $(u,u)$ is generated in step 2.
```

In this algorithm, A_i $(2 \leq i \leq n+1)$ is a set of pairs of nodes.

We consider the time complexity of the above algorithm. Let m be the size of the given OBDD. As shown in [5], step1 can be executed in m^2 time. In step2a, through n iterations, each variable node appears exactly once. Thus, it takes $O(m)$ time. In step2b, through n iterations, the same pair may be generated many times. However, as the number of different generated pairs is less than m^2, the total number of generated pairs is less than $2m^2$. Thus, Algorithm CheckOTS runs in $O(m^2)$ time.

Now we should prove that Algorithm CheckOTS correctly checks the ordered tree-shellability of the given Boolean function. This proof consists of two stages. We first show in Lemma 5 that there exists a pair of 1-paths P_i, P_j that satisfy some condition iff the function is not ordered tree-shellable with respect to the variable ordering. Then we show that the algorithm correctly detects such pair of 1-paths.

We call a 1-path P_j which satisfy $pos(P_j) = I_i$ the *main path* of I_i. If P_j is a main path of some prime implicant, we call P_j a main path. If an OBDT T witnesses that f is ordered tree-shellable, any 1-path of T is a main path. We call a 1-path P_j which satisfy $I_i \subseteq pos(P_j)$ a *corresponding path* of I_i.

Proposition 4 1. For any prime implicant I_i, there exists a main path of I_i.
2. Any path is a corresponding path of some prime implicant.

From Proposition 4 and the definition of ordered tree-shellable functions, we can see that there exists a pair of 1-paths both of which are corresponding paths of the same prime implicant iff f is not ordered tree-shellable. The next lemma shows that we have only to detect special ones among such pairs of 1-paths.

Lemma 5 Let T be an OBDD representing f with variable ordering π. f is not ordered tree-shellable with respect to π iff there exists a pair of 1-paths P_i, P_j in T which satisfies $pos(P_i) \subsetneq pos(P_j)$ and $|pos(P_j) \setminus pos(P_i)| = 1$.

Proof [if] If there exists a pair of 1-paths P_i, P_j satisfying $pos(P_i) \subsetneq pos(P_j)$, P_i and P_j are corresponding paths of the same prime implicant. That is, at least one of them is not a main path.
[only if] Assume f is not ordered tree-shellable. Then from Corollary 2, for some path P_i and $\overline{x_l} \in P_i$, there does not exist I_t $(t \neq i)$ that satisfy $I_t \subseteq pos(P_i) \cup \{l\}$ and $I_t \not\subseteq pos(P_i)$. For such P_i and x_l, let P_j be the path traversed by the assignment such that $x_k = 1$ iff $k = l$ or $x_k \in P_i$. If P_i is a corresponding path of $I_{i'}$, P_j is also a corresponding path of $I_{i'}$ because it cannot be a corresponding path of any other prime implicant. Thus, P_j satisfies $pos(P_j) = pos(P_i) \cup \{l\}$. Thus $pos(P_i) \subsetneq pos(P_j)$ and $|pos(P_j) \setminus pos(P_i)| = 1$ are satisfied. □

In the second step, we have to show that Algorithm CheckOTS correctly detects such pair of paths. In other words, we have to prove the following lemma.

Lemma 6 Algorithm CheckOTS finds a pair of nodes (u, u) iff there exists a pair of 1-paths P_i, P_j as described in Lemma 5.

Proof [only if] We prove that for any pair (v, w) generated in the algorithm

(∗) there exist paths P_v, P_w such that P_v is a path from the source to v, P_w is a path from the source to w, $pos(P_v) \subsetneq pos(P_w)$ and $|pos(P_w) \setminus pos(P_v)| = 1$.

If it holds, when there exist a pair (u, u), P_i and P_j of Lemma 5 are obtained by appending a path from u to the value node labeled 1 to P_v and P_w.

We prove it by induction on the number of iterations in step2. In the first iteration, one pair is generated in step2a and the pair satisfies (∗). We assume that all the pairs generated in the i-th iteration $(i < s)$ of step2 satisfy condition (∗). In the s-th iteration,

a) any pair generated in step2a clearly satisfies $(*)$, and
b) a pair generated in step2b from a pair (u', v') satisfies $(*)$ by appending literals corresponding to the edges used in the algorithm to $P_{u'}$ and $P_{v'}$ because (u', v') satisfies $(*)$.
[if] Let e_i^s, e_j^s be the endpoints of the subpaths of P_i, P_j that consist of the literals of $x_1, ..., x_s$. We prove that for any s, the pair (e_i^s, e_j^s) is generated in the algorithm.

When P_i and P_j diverge at some node (labeled x_t), only P_j have the positive literal x_t. Thus, for x_k $(k > t)$, either i) both P_i and P_j has the same literal, ii) either of them has $\overline{x_k}$ or iii) neither of them has a literal of x_k. Thus, we can see that pairs of nodes are generated in step2b for all the above possible cases. Therefore, if P_i and P_j join at node u, (u, u) never fails to be generated. □

□

6 Conclusion

In this paper, we have considered the complexity of checking ordered tree-shellability of a Boolean function given as an OBDD. We have shown that given an OBDD, it is possible to recognize in quadratic time a Boolean function that is ordered tree-shellable with respect to the variable ordering of the OBDD. However, it seems difficult to check if the given function is ordered tree-shellable with respect to the other variable orderings. To make use of the merits of ordered tree-shellable functions, it is important to find classes of Boolean functions for which this problem has small complexity.

参考文献

[1] S. B. Akers, Binary Decision Diagrams, IEEE Trans. Comput. C-27 (1978) 509-516.

[2] M. O. Ball and J. S. Provan, Disjoint Products and Efficient Computation of Reliability, Operations Research 36 (1988) 703-715.

[3] E. Boros, Y. Crama, O. Ekin, P. L. Hammer, T. Ibaraki and A. Kogan, Boolean Normal Forms, Shellability and Reliability Computations, RUTCOR Research Report 3-97 (1997).

[4] R. E. Bryant, Graph-based Algorithms for Boolean Function Manipulation, IEEE Trans. Comput. C35, No.8 (1986) 677-691.

[5] T. Horiyama and T. Ibaraki, Knowledge-Base Representation and Recognition of Positive/Horn Functions on Ordered Binary Decision Diagrams, Technical Report of IEICE, COMP98-85 (1999) 17-24.

[6] Y. Takenaga, K. Nakajima and S. Yajima, Tree-Shellability of Boolean Functions, Technical Report of IEICE, COMP97-54 (1997) 71-78.