Chicago Journal of Theoretical Computer Science

The MIT Press

Volume 1999, Article 5

Complexity of Problems on Graphs Represented as OBDDs

ISSN 1073–0486. MIT Press Journals, Five Cambridge Center, Cambridge, MA 02142-1493 USA; (617)253-2889; *journals-orders@mit.edu, journals-info@mit.edu*. Published one article at a time in LaTeX source form on the Internet. Pagination varies from copy to copy. For more information and other articles see:

- *http://mitpress.mit.edu/CJTCS/*
- *http://www.cs.uchicago.edu/publications/cjtcs/*
- *ftp://mitpress.mit.edu/pub/CJTCS*
- *ftp://cs.uchicago.edu/pub/publications/cjtcs*

The *Chicago Journal of Theoretical Computer Science* is abstracted or indexed in *Research Alert,®SciSearch, ®Current Contents®/Engineering Computing & Technology*, and *CompuMath Citation Index.®*

The *Chicago Journal of Theoretical Computer Science* is a peer-reviewed scholarly journal in theoretical computer science. The journal is committed to providing a forum for significant results on theoretical aspects of all topics in computer science.

Complexity of Problems on Graphs Represented as OBDDs

Joan Feigenbaum
AT&T Labs – Research
Room C203, 180 Park Avenue
Florham Park, NJ 07932 USA
jf@research.att.com

Sampath Kannan
Computer and Information Sciences
University of Pennsylvania
Philadelphia, PA 19104 USA
kannan@central.cis.upenn.edu

Moshe Y. Vardi
Computer Science
Rice University
Houston, TX 77251 USA
vardi@cs.rice.edu

Mahesh Viswanathan
Computer and Information Sciences
University of Pennsylvania
Philadelphia, PA 19104 USA
maheshv@saul.cis.upenn.edu

6 August 1999

Abstract

To analyze the complexity of decision problems on graphs, one normally assumes that the input size is polynomial in the number of vertices. Galperin and Wigderson [GW83] and, later, Papadimitriou and Yannakakis [PY86] investigated the complexity of these problems when the input graph is represented by a polylogarithmically succinct circuit. They showed that, under such a representation, certain trivial problems become intractable and that, in general, there is an exponential blowup in problem complexity. Later, Balcázar, Lozano, and Torán [Bal96, BL89, BLT92, Tor88] extended these results to problems whose inputs are structures other than graphs.

In this paper, we show that, when the input graph is represented by an ordered binary decision diagram (OBDD), there is an exponential blowup in the complexity of most graph problems. In particular, we show that the GAP and AGAP problems become complete for PSPACE and EXP, respectively, when the graphs are succinctly represented by OBDDs.

An extended abstract of this paper appears in the proceedings of the 1998 Symposium on Theoretical Aspects of Computer Science.

1 Introduction

The efficiency of algorithms is generally measured as a function of input size (see [CLR89]). In analyses of graph-theoretic algorithms, graphs are usually assumed to be represented either by adjacency matrices or by adjacency lists. Nevertheless, many problem domains, most notably, computer-aided verification (refer to [Bry86, BCM+92, Kur94a]), involve extremely large graphs that have regular, repetitive structure. This regularity can yield very succinct encodings of the input graphs, and hence one expects a change in the time complexity or space complexity of the graph problems.

The effect of succinct input representations on the complexity of graph problems was first formalized and studied by Galperin and Wigderson [GW83]. They discovered that, when adjacency matrices are represented by polylogarithmically sized circuits, many computationally tractable problems become intractable. Papadimitriou and Yannakakis [PY86] later showed that such representations generally have the effect of exponentiating the complexity (time or space) of graph problems. Following this line of research, Balcázar, Lozano, and Torán [Bal96, BL89, BLT92, Tor88] extended these results to problems whose inputs are structures other than graphs and provided a general technique to compute the complexity of problems with inputs represented by succinct circuits [BLT92]. They also provided sufficiency conditions for problems that become intractable when inputs are represented in this way. Veith [Vei95, Vei96] showed that, even when inputs are represented using Boolean formulae (instead of circuits), a problem's computational complexity can experience an exponential blowup. He also gave sufficiency conditions for when the problems become hard.

The possibility of representing extremely large graphs succinctly has attracted a lot of attention in the area of computer-aided verification (e.g., [Bry86, BCM+92, Kur94a]). In this domain, graphs are represented by *ordered binary decision diagrams* (OBDDs). OBDDs are special kinds of rooted, directed acyclic graphs that are used to represent Boolean formulae. Because of their favorable algorithmic properties, they are widely used in the areas of digital design, verification, and testing (see [Bry92, BCM+92, McM93]). Experience has shown that OBDD-based algorithmic techniques scale up to industrial-sized designs (see [CGH+95]), and tools based on such techniques are gaining acceptance in industry (refer to [BBDG+94]). Although OBDDs provide canonical succinct representations in many practical situations, they are exponentially less powerful than Boolean circuits, in the formal

sense that Boolean functions exist that have polynomial-sized circuit representations but do not have subexponential-sized OBDD representations (see [Pon95a, Pon95b]). (On the other hand, the translation from OBDDs to Boolean circuits is linear [Bry86].) Thus, the results of [BL89, BLT92, GW83, PY86, Tor88, Vei95, Vei96] do not apply to OBDD-represented graphs. Furthermore, even though Boolean formulae are, in terms of representation size, less powerful than circuits, they are still more succinct than OBDDs. Translation from OBDDs to formulae leads to at most a quasi-polynomial ($n^{\log n}$) blowup, whereas there are functions (e.g., multiplication of binary integers) that have polynomial-sized formulae but require exponential-sized OBDDs. Indeed, while the satisfiability problem is NP-complete for Boolean formulae, it is in nondeterministic logspace for OBDDs (see [Bry86]). Therefore, the results in [Vei95, Vei96] do not apply to our case.

In this paper, we show that, despite these theoretical limitations on the power of OBDDs to encode inputs succinctly, using them to represent graphs nonetheless causes an exponential blowup in problem complexity. That is, the well-studied phenomenon of exponential increase in computational complexity for graph problems with inputs represented by Boolean circuits or formulae (see [BL89, BLT92, GW83, PY86, Tor88, Vei95, Vei96]) also occurs when the graphs are represented by OBDDs. Graph properties that are ordinarily NP-complete become NEXP-complete. The graph accessibility problem (GAP) and the alternating graph accessibility problem (AGAP) for OBDD-encoded graphs are PSPACE-complete and EXP-complete, respectively. Both GAP and AGAP are important problems in *model checking*, a domain in which OBDDs are widely used (refer to [BCM$^+$92, EL86, KV96, Kur94b]).

In Section 2, we formally define OBDDs and present some known results about them. In Section 3, we discuss the problem in greater detail and compare Papadimitriou and Yannakakis's result to ours. Finally, in Sections 5–7, we give our technical results.

2 Preliminaries

Definition 1 *A binary decision diagram (BDD) is a single-rooted, directed acyclic graph in which*

- *Each internal node (i.e., a node with nonzero outdegree) is labeled by a Boolean variable.*

- *Each internal node has outdegree* 2. *One of the outgoing edges is labeled* 1 *(the "then-edge") and the other is labeled* 0 *(the "else-edge").*

- *Each external node (i.e., a node with zero outdegree) is labeled* 0 *or* 1.

Let $X = \{x_1, x_2, \ldots, x_n\}$ be the set of Boolean variables that occur as labels of nodes in a given BDD B. Each assignment $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ of Boolean values to these variables naturally defines a computational path—the one that leads from the root to an external node and has the property that, when it reaches a node labeled x_i, it follows the edge labeled α_i, for all i.

Definition 2 *A BDD B represents the Boolean function $f(x_1, x_2, \ldots, x_n)$ if, for each assignment $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ to the variables of f, the computation path defined by α terminates in an external node that is labeled by the value $f(\alpha_1, \alpha_2, \ldots, \alpha_n)$.*

Definition 3 *Two nodes u and v of a BDD are equivalent if the BDDs rooted at u and v represent the same Boolean function. A BDD in which no two different nodes are equivalent is called reduced.*

Definition 4 *Let $<$ be a total ordering on a set X. An ordered binary decision diagram (OBDD) over $(X, <)$ is a reduced BDD with node-label set X such that, along any path from the root to an external node, there is at most one occurrence of each variable, and the order in which the variables occur along the path is consistent with the order $(X, <)$. The size of an OBDD is the number of internal nodes in it.*

Definition 5 *An OBDD O represents the graph $G = (V, E)$ if O represents the Boolean function* adj, *where*

$$\mathrm{adj}(v_1, v_2) = \begin{cases} 1 & \text{if and only if } \langle v_1, v_2 \rangle \in E \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 1 (Bryant [Bry86]) *For each Boolean function f and ordering $(X, <)$ of the set of variables X, there is a unique (up to isomorphism) OBDD over $(X, <)$ that represents f.*

Theorem 2 (Bryant [Bry86]) *Let F and G be OBDDs over $(X, <)$ representing functions f and g, respectively. Let the size of F be m, the size of G be n, and $\langle \mathrm{op} \rangle$ be a binary Boolean operation. Then there is an OBDD over $(X, <)$ of size at most mn and constructable in time polynomial in m and n that represents $f \langle \mathrm{op} \rangle g$.*

Definition 6 *Let $L = (G, <)$ be a linear order on the gates of a circuit, where the inputs and outputs are classified as special instances of gates. We say that the* forward cross section of the circuit at gate g *is the number of wires connected to the output of some gate g_1 AND an input of some gate g_2 such that $g_1 \leq g$ and $g < g_2$. The* reverse cross section of the circuit at gate g *is the number of wires connected to an output of some gate g_1 and an input of some gate g_2 such that $g_2 \leq g$ and $g < g_1$.*

Definition 7 *The forward width of a circuit under order L, denoted w_f, is the maximum, over all gates g, of the forward cross section at g. Similarly, the reverse width of the circuit under order L, denoted by w_r, is the maximum, over all gates g, of the reverse cross section at g.*

Theorem 3 (Berman [Ber89]) *For a circuit and gate ordering with $w_r = 0$, there exists a variable ordering such that the OBDD size is bounded by $n2^{w_f}$, where n is the number of inputs to the circuit.*

Notation 1 *We are interested in complexity classes $\mathbf{C}$ that have universal Turing machines and complete problems. Let $U_{\mathbf{C}}$ denote the universal Turing machine for the complexity class $\mathbf{C}$. Let $\mathcal{L}(U_{\mathrm{C}})$ be the language accepted by the machine $U_{\mathbf{C}}$; that is, $\mathcal{L}(U_{\mathrm{C}}) = \{x : x$ encodes a $\mathbf{C}$-bounded Turing machine M and an input y such that M accepts $y\}$.*

For an n-bit number x, we refer to the ith bit by $x^{(i)}$, where $x^{(n)}$ is the most significant bit.

3 Problem statement

Papadimitriou and Yannakakis [PY86] show that any NP-complete graph property π to which satisfiability is reducible by a *projection*, in the sense of Skyum and Valiant [SV82], becomes NEXP-complete when problem instances are encoded as circuits. They do this by first constructing a circuit

that computes the clause-literal incidence matrix of a formula $F(x)$ (i.e., given a clause and a literal, the circuit decides whether the literal occurs in the clause in $F(x)$) such that $F(x)$ is satisfiable if and only if $x \in \mathcal{L}(U_{\text{NEXP}})$. Then, using the properties of projection, they construct a circuit representing a graph $G(x)$ such that $G(x)$ has a property π if and only if $x \in \mathcal{L}(U_{\text{NEXP}})$.

When graphs are represented by OBDDs, such reductions are not immediately obtainable, for two basic reasons. First, OBDDs are strictly less powerful than Boolean circuits, in the sense that there are Boolean functions that have small circuit representations but no small OBDD representations. In particular, the function that computes the ith bit of the product (or quotient) of two binary numbers cannot be represented by a small OBDD (see [Pon95a, Pon95b]). Second, the size of OBDDs is sensitive to the ordering of the variables of the function, and the OBDD representing the Boolean combination of two OBDDs can be constructed quickly only when the ordering of the variables is consistent in the two OBDDs. Hence, for a result equivalent to Papadimitriou and Yannakakis's to hold for graphs represented by OBDDs, we must construct reductions f such that the jth bit of $f(x)$ can be found by a small OBDD given j as the input, assuming that the ith bit of x can be found by a small OBDD given i as the input. Furthermore, all OBDDs involved must read the bits in consistent order.

4 Cook's theorem

In order to show that satisfiability is NP-hard, Cook [Coo71] constructed a Boolean formula $F(x)$, given a string x encoding a nondeterministic Turing machine decription and an input string, such that $F(x)$ is satisfiable if and only if the machine accepts the input in polynomial time. If we view computation as a sequence of machine descriptions,[1] then the formula essentially encodes the following: Each machine description is correct and follows from the previous description in the sequence, and the machine finally enters an accepting state. We now describe this formula $F(x)$ more formally.

When describing the machine at some instant, we group the state of the machine and the index of the next step taken by the machine, with the symbol scanned to form a single composite symbol whose appearance also indicates the head position. For each time instant i, for each tape cell j, and

[1] A machine description is a sequence that describes the values of the tape cells and the position of the input head.

for each symbol X, which can either be a symbol of the tape alphabet or a composite symbol, we create a Boolean variable $V_{i,j,X}$ to indicate that the content of cell j at time i is X. If $p(n)$ denotes the polynomial time bound on the machine, then the formula $F(x)$ is a conjuction of the following four formulae:

(A) $\varphi_A = (V_{0,0,\triangleright}) \wedge (V_{0,1,a_1} \vee V_{0,1,a_2} \vee \cdots \vee V_{0,1,a_l}) \wedge \bigwedge_{i=2}^{n} V_{0,i,y^{(i)}} \wedge V_{0,n+1,\triangleleft} \wedge \bigwedge_{i=n+2}^{p(n)} V_{0,i,B}$. This basically says that at the start (time 0), the tape contains the input between the left ($\triangleright$) and right end-markers ($\triangleleft$), and the rest of the tape is blank. The symbols a_i encode the start state, first input symbol, and various nondeterministic choices available initially.

(B) $\varphi_B = \bigvee_j (\bigvee_{X \in F} V_{p(n),j,X})$. This says that at time $p(n)$ the machine is one of the final states. (F is the set of composite symbols encoding a final state and a tape symbol.)

(C) $\varphi_C = \bigwedge_{i,j} [\bigvee_X V_{i,j,X} \wedge \neg(\bigvee_{X \neq Y} (V_{i,j,X} \wedge V_{i,j,Y}))]$. This says that at each time and for each cell, there is exactly one symbol.

(D) $\varphi_D = \bigwedge_{i,j} [\bigvee_{(W,X,Y,Z) \in \mathrm{Quad}} (V_{i,j-1,W} \wedge V_{i,j,X} \wedge V_{i,j+1,Y} \wedge V_{i+1,j,Z})]$, where Quad is the set of all quadruples (W, X, Y, Z) such that if the symbols in the $(j-1)$th, jth, and $(j+1)$th cells of the tape are W, X, and Y, respectively, then at the next time instant the symbol at the jth cell is Z. So this says that the symbol at each time and for each cell follows correctly from those at the previous time instance.

5 A small OBDD for a NEXP tableau

Applying Cook's theorem to the tableau of a nondeterministic, exponential-time Turing machine produces a Boolean formula with exponentially many clauses and literals such that the formula is satisfiable if and only if the tableau represents a valid, accepting computation. In this section, we show that this formula can be represented succinctly by an OBDD. This means that we can fix an enumeration of the clauses and literals such that, given the indices of a clause and of a literal, we can determine whether the literal occurs in the clause. Our proof exploits the great regularity of the formula in question. As can be seen in Figure 5, a literal corresponding to cell j at time $i-1$ occurs only in the clauses corresponding to cells $j-1$, j, and $j+1$ at

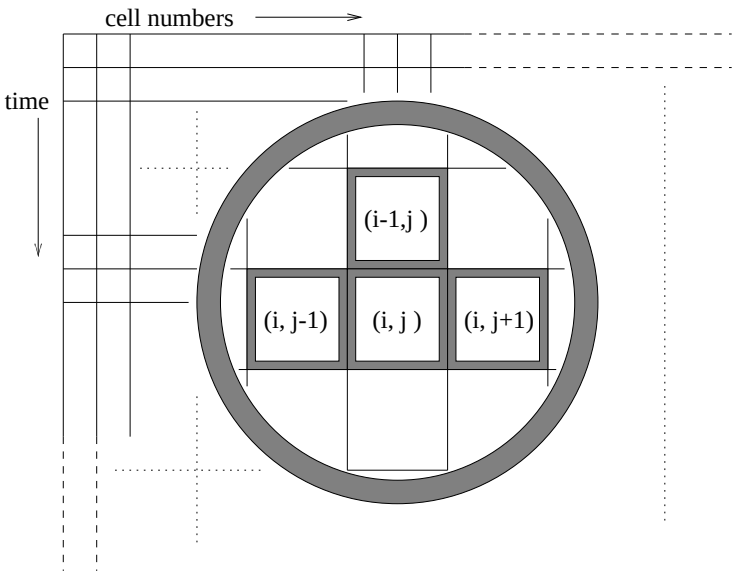

Figure 1: A NEXP Tableau. Rows represent contents of tape cell at a particular time instance. Time increases as you go down.

time i. This implies that by a suitable ordering of the clauses, we can ensure the following: Roughly speaking, for a given input x, there is a small, fixed constant c such that a literal with index l occurs only in clauses with indices between $(l-1)c+1+K$ and $lc+K$, where K is some (not necessarily small) number.

We begin by proving (in Lemmas 1 and 2) that this range check can be computed by a small OBDD. The constructed OBDD uses an ordering of variables wherein bits of the literal index and bits of the clause index are interleaved. This construction plays a central role in the proof of the main theorem of this section.

Lemma 1 *For a fixed integer Y, there is a circuit with $w_f = \log\ Y + 2$ and $w_r = 0$ that, given an ordering $x^{(n)} < x^{(n-1)} < \cdots < x^{(1)}$ on input bits, computes the ith bit of x/Y ("/" denotes integer division), for all i. Similarly, for a fixed Y, there is a circuit with $w_f = \log\ Y + 2$ and $w_r = 0$ that, given an ordering $x^{(n)} < x^{(n-1)} < \cdots < x^{(1)}$ on input bits, computes the ith bit of x mod Y, for all i.*

Proof A circuit that does long division by repeated subtraction computes both the quotient and the remainder and has the above properties. ■

Lemma 2 *Let f be a function such that $f(x)^{(i)}$ depends only on $x^{(n)}, x^{(n-1)}, \ldots, x^{(i)}$. Given an ordering $x^{(n)} < y^{(n)} < x^{(n-1)} < \cdots < y^{(1)}$ on input bits, there is a circuit that checks whether $f(x \pm K) = y$, for a fixed integer K, with $w_r = 0$ and $w_f = 2W + 4$, where W is the forward width of the circuit that computes $f(x)$ given the same ordering on input bits.*

Proof We now present the construction for a circuit computing the function $f(x + K)$ in detail. The case of $f(x - K)$ is similar, and we present a sketch of the proof at the end.

The difficulty arises in computing $x + K$, using the given ordering on the input bits. If we used the reverse ordering, going from least significant bits to most significant bits, designing a circuit with the desired widths would be simple. Unfortunately, the circuit computing f does so with a different ordering.

In order to compute the ith bit of the sum, we need to know whether $(x^{(i-1)}x^{(i-2)} \cdots x^{(1)} + K^{(i-1)}K^{(i-2)} \cdots K^{(1)})$ yields a carry, and we cannot know this until we compute the sum.

We overcome this obstacle using essentially the same idea as is used in carry-look-ahead adders. A bit position i is a carry *generator* if $x^{(i)}$ and $K^{(i)}$ are 1; it is a carry *propagator* if exactly one of these bits is 1; it is a carry *killer* if both of these bits are 0.

Initially, we compute $f(x+K)^{(n)}$ in two ways, one assuming that there is a carry into the nth position and the other assuming that there is no carry into the nth position. We compare each value of $f(x+K)^{(n)}$ with $y^{(n)}$ terminating any computation path that leads to inequality. Inductively, suppose that we compute at most two different values for $f(x + K)^{(i)}$ and compare both with $y^{(i)}$. If position $i - 1$ is a generator, we abandon the path in which there is no carry into position i and continue the path in which there is a carry into position i in two ways—one assuming there is a carry into position $i - 1$ and the other assuming not. Similarly, if position $i - 1$ is a propagator, we continue the path assuming a carry into position i by assuming that there is a carry into position $i - 1$, and we continue the path assuming no carry into position i by assuming that there is no carry into position $i - 1$. Finally, if position $i - 1$ is a carry killer, we abandon the path assuming that there is a carry into position i, and once more we are left with at most two paths

to continue. Clearly, only one of these two paths is correct at the end, and if along this path we have determined that the bits of $f(x + K)$ are equal to the bits of y, we output a 1; otherwise we output a 0. Our computation is an appropriate interleaving of the carry-look-ahead adder, the circuit that computes f hypothesized in the lemma, and a circuit that checks equality of the bits of $f(x + K)$ and the bits of y.

To compute $f(x - K)$ we proceed in a similar manner. ■

Notation 2 *Consider the language $\mathcal{L}(U_{\text{NEXP}})$. Let $F(x)$ be the CNF Boolean formula obtained by the exponential version of Cook's construction, in which $F(x)$ is satisfiable if and only if $x \in \mathcal{L}(U_{\text{NEXP}})$.*

Theorem 4 *Let g_x be the Boolean function that decides whether a given literal occurs in a given clause in $F(x)$; that is, $g_x(Cl, Lt) = 1$ if and only if the literal whose index is Lt occurs in the clause of $F(x)$ whose index is Cl. There is an OBDD of size polynomial in the length of x that represents the function g_x.*

Proof As we saw in Section 4 there are four categories of clauses in $F(x)$:

(A) Clauses that state that, at time 0, the tape contains the input string, and the machine is in the initial state

(B) Clauses that state that, at time $2^{p_M(n)}$, the machine is in one of the final states

(C) Clauses that state that, at time i, $0 \leq i \leq 2^{p_M(n)}$, each tape cell contains exactly one symbol of the tape alphabet

(D) Clauses that state that, at time i, $0 \leq i \leq 2^{p_M(n)}$, the contents of the tape cells and the state of the machine follow from those at the previous time $i - 1$ by a valid "move" of the machine

We first list all the clauses in category A, then those in category B, and then, finally, those in categories C and D. The clauses in categories C and D are interleaved so that all clauses in these two categories referring to the same cell of the tableau occur together in the enumeration starting from $\langle 0, 0\rangle$, that is, cell 0 and time 0, and proceeding in row major order.

An important property becomes clear in the proof: There is an integer W such that, for each pair i, j, the number of clauses in category C and D

for time i and tape cell j is W, and W depends only on the alphabet size and the maximum nondeterministic branching possible at any state. Hence, for the above listing of the clauses, determining the time and cell to which a given clause number refers involves dividing by this fixed constant W and not by a number that is a function of i or j. This is very important, because the function that determines the quotient when one number is divided by another does not, in general, have a small OBDD representation. Because there is a fixed constant W, we can use Lemma 1.

The literals $V_{i,j,X}$ have the same meaning as in Section 4, and they are numbered in the following order: $V_{0,0,\sigma_1}, \neg V_{0,0,\sigma_1}, \ldots, V_{0,0,\sigma_m}, \neg V_{0,0,\sigma_m}, V_{0,1,\sigma_1}, \ldots, \neg V_{0,2^{p_M(n)},\sigma_m}, V_{1,0,\sigma_1}, \ldots, \neg V_{2^{p_M(n)},2^{p_M(n)},\sigma_m}$, where $\Gamma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ is the set of tape symbols and composite symbols.

Notation 3 *We use $\#(X)$ to denote the index of X, where X is a literal, clause, or symbol in the enumeration.*

We now show that, for the set of clauses in any category, g_x when restricted to this set of clauses can be represented by an OBDD of polynomial size. Since g_x is a logical OR of all these (constant number of) Boolean functions, Theorem 2 implies that there is a polynomial-sized OBDD that represents g_x.

In the rest of the proof, we only consider OBDDs with the following ordering on the variables: $Cl^{(k)}, Lt^{(k)}, Cl^{(k-1)}, Lt^{(k-1)}, \ldots, Cl^{(1)}, Lt^{(1)}$, where Cl is the index of a clause and Lt is the index of a literal.

Case A Each clause in this category consists of a single literal. Clauses corresponding to cell positions $\langle 0, 0\rangle$ to $\langle 0, n+1\rangle$ specify that the top row faithfully represents the input string $y^{(1)}, \ldots, y^{(n)}$ along with suitable endmarkers and the start state, while clauses corresponding to cell positions $\langle 0, n+2\rangle, \ldots, \langle 0, 2^{p_M(n)}\rangle$ specify that these symbols are B, the blank symbol.

Note that the OBDD is allowed to be of size polynomial in n. Thus we can compute g_x in the case when the clause number is between 0 and $n+1$ by using an OBDD that resembles a trie. Upon reading a symbol, simply branch to the node in the trie that represents all possible continuations of the clause and literal indices that would make g_x evaluate to 1. The size of the trie (and hence also the size of the OBDD) is $O(n)$.

For clauses whose number is between $n+2$ and $2^{p_M(n)}$, check that Lt mod $2m$ produces a number that encodes the blank symbol. Lemma 1 implies that

there is an OBDD, of small size, that can compute the ith bit of $Lt \bmod 2m$. Now checking if Lt encodes a blank symbol just involves checking that each bit $Lt \bmod 2m$ is "correct," and since the number of bits in $Lt \bmod 2m$ is bounded by the input size ($|x|$), this conjunction has a small (polynomial in $|x|$) sized OBDD.

The OR of these two OBDDs computes g_x in case A.

Case B The clauses in this category are

$$\bigvee_j \left(\bigvee_{X \in F} V_{2^{p_M(n)},j,X} \right),$$

where F is the set of composite symbols that encode a final state and symbol pair.

Here we need to test that $Cl = 2^{p_M(n)} + 1$, $Lt \mod 2m$ is a symbol that encodes a final state, and that Lt is an index of a literal corresponding to time $2^{p_M(n)}$, that is, $Lt/2m \geq \langle 2^{p_M(n)}, 0 \rangle$. The first two tests clearly have small OBDDs. Because $|F| \leq m$ (the size of the set of tape symbols and composite symbols), the OBDD that decides whether a number ($\leq 2m$) is a symbol in F has at most m paths and so is small. Thus we also have a small OBDD representation for g_x in this case.

Case C The clauses in this case are

$$\bigvee_X V_{i,j,X}$$
$$\neg V_{i,j,X} \vee \neg V_{i,j,Y}, \qquad \text{where} \quad X \neq Y.$$

Recall that we interleaf the clauses in categories C and D. Thus part of the tests in both these categories involve checking whether the "block" (i.e., the $\langle i, j \rangle$ pair) is "correct."

The block and offset within a block, for the index of a clause, can be determined as follows:

$$\begin{aligned} (Cl - K_1)/K_2 &= \text{block and} \\ (Cl - K_1) \mod K_2 &= \text{offset,} \end{aligned}$$

where $K_1 = 2^{p_M(n)} + 1$ is the number of clauses in category A and B, and K_2 is the number of clauses in category C and D for a fixed i and j.

Because K_2 is a constant that depends only on the alphabet size and the maximum nondeterministic branching possible in M, Lemmas 1 and 2 and Theorem 3 imply that the OBDDs that compute each bit of the block and offset are small. Again, since the total number of bits in the block and offset are bounded by the length of the input, $|x|$, checking if the block (or offset) is equal to some value can be done by a small OBDD.

Subcase 1 In the case $(Cl - K_1) \bmod K_2 = 1$, we check that at least one symbol occurs in each cell position. We see whether $(Cl - K_1)/K_2 = Lt/2m$, $(Cl - K_1) \mod K_2 = 1$, and $Lt \mod 2m$ is odd (i.e., literal is positive). Each of these tests has a small OBDD, and so subcase 1 presents no problems.

Subcase 2 In the case $1 < (Cl - K_1)/K_2 \bmod K_2 \leq \binom{m}{2} + 1$, we test whether $(Cl - K_1)/K_2 = Lt/2m$ and $Lt \mod 2m$ occurs in the clause whose offset is $(Cl - K_1) \mod K_2$.

For each value of $(Cl - K_1) \mod K_2$, the OBDD that checks for the occurrence of $Lt \mod 2m$ has two paths, because each such clause has two literals, and thus it is small. The desired OBDD has $O(m^2)$ paths, where, in the ith path, one checks for the case $(Cl - K_1) \bmod K_2 = i$. The OBDD in subcase 2 is thus polynomial-sized.

Case D The Boolean formula that states that any transition of the machine from one configuration to another must follow from a valid "move" of the machine looks like

$$\bigvee_{(W,X,Y,Z)\in \text{Quad}} (V_{i,j-1,W} \wedge V_{i,j,X} \wedge V_{i,j+1,Y} \wedge V_{i+1,j,Z}),$$

where Quad is the set of all quadruples (W, X, Y, Z) with the following property: If the symbols in the $(j-1)$th, jth, and $(j+1)$th cells of the tape are W, X, and Y, respectively, then at the next time instant the symbol at the jth cell will be Z.

Distributing the $\vee$ over the $\wedge$'s gives the following set of clauses:

$$V_{i,j-1,W_1} \vee V_{i,j-1,W_2} \vee \cdots \vee V_{i,j-1,W_k},$$
$$V_{i,j-1,W_1} \vee V_{i,j-1,W_2} \vee \cdots \vee V_{i,j,X_1},$$
$$\vdots$$
$$V_{i+1,j,Z_1} \vee V_{i+1,j,Z_2} \vee \cdots \vee V_{i+1,j,Z_k},$$

where $|\text{Quad}| = k$.

Subcase 1 We have the case of $(Cl - K_1)/K_2 = Lt/2m + 1$, that is, the one in which the literal refers to the $(j-1)$th cell and time i (or $\langle i, j-1 \rangle$). We must check whether the symbol encoded by Lt is one of the W_i's occurring in the clause. The clauses (for each i and j) are listed as above. There is a close correspondence between the offset of a clause (i.e., $(Cl - K_1) \mod K_2$) represented in base 4 and the presence of a literal $V_{i,j-1,W_i}$ in the clause: If the ith most significant bit of the base-4 representation of the offset is a 0, then $V_{i,j-1,W_i}$ is present in the clause. Otherwise it is not. So checking if Lt is present in the clause Cl involves checking if certain bits of the offset are 0. (A base-4 representation can be obtained by grouping pairs of bits in the binary representation.) Since the bits of the offset can be obtained by a small OBDD, there is a small OBDD for this subcase.

Subcases 2 through 4 are listed below. Arguments showing that there are small OBDDs for these cases are similar to the one in subcase 1.

Subcase 2 $(Cl - K_1)/K_2 = Lt/2m$.

Subcase 3 $(Cl - K_1)/K_2 = Lt/2m - 1$.

Subcase 4 $(Cl - K_1)/K_2 = Lt/2m - 2^{p_M(n)}$.

■

6 NP-complete graph problems

Theorem 4 can be used to prove that most classical NP-complete graph problems are NEXP-complete when graphs are represented by OBDDs. We give one example of such a proof; others are quite similar.

We now consider the independent set problem. Recall that, in the decision version of this problem, we are given a graph G and an integer k in binary and are asked if G has an independent set of size at least k. For the succinct version of the problem, we assume that we are given an OBDD O representing the adjacency relation of a graph and an integer k in binary and are asked if the graph represented by O has an independent set of size at least k.

Theorem 5 *The independent set problem for graphs represented by OBDDs is NEXP-complete.*

Proof Consider the standard reduction from 3-SAT to independent set. In this reduction, we create a graph in which there is a node for each occurrence of each literal and two nodes are adjacent if and only if they either correspond to two literals in the same clause or correspond to two complementary literals x and $\bar{x}$ occurring in two different clauses. Let $G_{F(x)}$ be the graph obtained by such a reduction from the formula $F(x)$, for some string x that encodes a $2^{p_M(n)}$-time bounded Turing machine M and an input y. Let each node of $G_{F(x)}$ be named by the clause-literal pair corresponding to it.

Claim 1 *Given two vertices (Cl_1, Lt_1) and (Cl_2, Lt_2) of graph $G_{F(x)}$, there is a polynomial-sized OBDD that decides whether these vertices are adjacent.*

Proof In order to check whether (Cl_1, Lt_1) is adjacent to (Cl_2, Lt_2), we check whether (Cl_1, Lt_1) and (Cl_2, Lt_2) are "valid" vertices in $G_{F(x)}$.

(a) If either one of these vertices is not valid, we declare them to be adjacent.

(b) If both are valid, we check whether they are adjacent as per the reduction described above.

A node (Cl, Lt) is valid if and only if the literal whose index is Lt occurs in the clause Cl, that is, if and only if $g_x(Cl, Lt) = 1$. From the previous section, we know that there is a small OBDD that computes g_x, and so we have a small OBDD to check whether a node is valid.

From the construction of $G_{F(x)}$, we know that two valid vertices are adjacent if and only if either they correspond to literals in the same clause or they correspond to complementary literals in different clauses; that is, either $Cl_1 = Cl_2$ or $|Lt_1 - Lt_2| = 1$ and $(Lt_1 - 1)/2 = (Lt_2 - 1)/2$. By Lemma 2, both of these checks can be done by small OBDDs whose orderings of the variables are consistent with that of the OBDD for g_x. ■

Thus the adjacency relation for graph $G_{F(x)}$ can be represented by a small OBDD. Furthermore this OBDD can be constructed in polynomial time. $G_{F(x)}$ has an independent set of size K, where K is the number of clauses in $F(x)$, if and only if $F(x)$ is satisfiable if and only if $x \in \mathcal{L}(U_{\text{NEXP}})$.

Hence, the independent set problem for graphs represented by OBDDs is NEXP-complete. ■

Papadimitriou and Yannakakis [PY86] prove the general theorem that, if the reduction from SAT to an NP-complete problem π is a projection, then π becomes NEXP-complete when the input is represented by a circuit. The fact that the reduction is a projection is a sufficient condition for their result, but it appears far from necessary.

Here we state an analogous result. Let f be a reduction from SAT to an NP-complete problem π. Suppose there is a constant k such that, for all j, $f(x)^{(j)}$ is a function only of the bits $x^{(j_1)}, \ldots, x^{(j_k)}$. Moreover, suppose that there is a Mealy machine that takes the bits of j in some canonical order as input and produces the bits of $j_1, \ldots, j_k$ in most significant to least significant order.[2] More precisely, there is a finite automaton that, on reading the ith bit of j, produces the ith most significant bit of $j_1, \ldots, j_k$. We refer to the above class of reductions as padding reductions. Note that the class of padding reductions is incomparable with the class of projections.

We state the following theorem and provide a sketch of the proof.

Theorem 6 *Let f be a padding reduction from SAT to a problem π in NP. Then π is NEXP-complete when its instances are presented as OBDDs.*

Proof (sketch) Consider the instance $f(F(x))$ of problem π, where as before $F(x)$ denotes the Boolean formula obtained when Cook's theorem is applied to a NEXP tableau. We need to show that there is a small OBDD O that, when given j, outputs the value of $f(F(x))^{(j)}$, or in other words, that there is a small OBDD representation for the instance $f(F(x))$.

Essentially, since f is an padding reduction, on reading the first bit of j we know the most significant bits of $j_1, \ldots, j_k$, and so on. So if we have k copies of the OBDD representation of $F(x)$ (constructed in Section 5), we can step through these OBDDs simultaneously as we read each bit of j. Once we compute $F(x)^{(j_i)}, \ldots, F(x)^{(j_k)}$, we can obtain the value of $f(F(x))^{(j)}$. Since k is a constant, this construction yields a polynomial-sized OBDD. ■

Padding reductions can be found for a number of graph problems such as clique, vertex cover, and so on. Such reductions are obtained by taking the standard reductions and padding the target instances so that each of its indices has sufficient information to allow the reconstruction of the indices on which it depends. For example, in the case of the independent set problem,

[2] A Mealy machine is a finite automaton that produces an output on reading an input symbol; see [HU79].

we padded the indices of the vertices of the graph obtained by the standard reduction. We labeled the vertices by a clause-literal pair, and this helped in constructing a small OBDD for the adjacency relation.

7 The GAP and AGAP problems

In this section, we examine two graph problems that are crucially important in computer-aided verification. We show that both experience exponential blowup in worst-case complexity when instances are represented as OBDDs.

Problem (GAP) The graph accessibility problem is as follows.

> *Input:* A directed graph G and vertices s and t in the graph.
>
> *Output:* Is there a directed path from s to t?

Theorem 7 *GAP is PSPACE-complete when the graph G is represented by an OBDD.*

Proof Let p be a polynomial and x be a string that encodes a $p_M(|y|)$-space-bounded Turing machine M and an input y. Without loss of generality, we may assume that the machine M has a single accepting configuration C_f.

Consider the configuration graph $G_{M,y} = (V_{M,y}, E_{M,y})$ corresponding to the machine M and input y, where

> $V_{M,y} = \{v_C \mid C$ is a possible configuration of the machine, that is, C is a string of symbols, of which one is a composite symbol encoding a state of machine M and a tape symbol, and all the rest are tape symbols $\}$;
>
> $E_{M,y} = \{\langle v_{C_1}, v_{C_2}\rangle \mid$ the machine M can go from configuration C_1 to a configuration C_2 in one step$\}$.

If C_i is the initial configuration of machine M on input y, then M accepts y if and only if the node labeled by C_f is reachable from the node labeled by C_i in $G_{M,y}$. In other words, $x \in \mathcal{L}(U_{\text{PSPACE}})$ if and only if the GAP problem on $G_{M,y}$ has the answer "yes."

Claim 2 *The edge relation $E_{M,y}$ in graph $G_{M,y}$ has a small OBDD representation.*

Proof We need to show that the function e, where

$$e(C_1, C_2) = \begin{cases} 1 & \text{if and only if } \langle v_{C_1}, v_{C_2} \rangle \in E_{M,y} \\ 0 & \text{otherwise} \end{cases}$$

can be represented by a small OBDD. Computing the function e entails

(a) checking that C_1 and C_2 are possible configurations, that is, there is exactly one composite symbol in each of C_1 and C_2, and

(b) checking whether the configuration C_2 can be reached from C_1 in one step by the machine M.

Checking to see whether a symbol is composite amounts to checking whether the index of the symbol is greater than some constant, because we list all the composite symbols in the end. Hence, check (a) involves examining only the symbols of the configuration in the order in which they occur and thus has a small OBDD representation.

Let Quad = $\{(W, X, Y, Z)|$ if W, X, and Y are the symbols in the $(j-1)$th, jth, and $(j+1)$th cells, respectively, at some time instant, then Z is the symbol in the jth cell at the next time instant$\}$. Checking whether configuration C_2 can be reached from configuration C_1 in one step involves checking whether all the symbols in C_2 arise from the corresponding symbols in C_1. That is, we need to check that for all j, $(C_1^{(j-1)}, C_1^{(j)}, C_1^{(j+1)}, C_2^{(j)}) \in$ Quad. As we saw in the proof of Theorem 4, the function that checks whether a given quadruple is in Quad can be represented by a small OBDD. We just read the symbols of C_1 and C_2 alternately and keep checking whether they "conform." Note that we need to "remember" only two symbols of C_1 as we go along. Hence, at any level in the OBDD, there are at most a constant number of nodes, and checking whether one configuration can follow from another is representable by a small OBDD. ■

Because $G_{M,y}$ can be represented by a small OBDD that can be constructed in polynomial time, the GAP problem for graphs represented by OBDDs is PSPACE-complete. ■

Definition 8 *An AND-or graph is a directed graph G with vertices labeled AND or OR. Reachability in such graphs is recursively defined as follows:*

(a) Every vertex is reachable from itself.

(b) *If u is an AND node, then v is reachable from u if and only if v is reachable from* all u_i*, such that $\langle u, u_i \rangle$ is an edge in the graph.*

(c) *If u is an OR node, then v is reachable from u if and only if v is reachable from* any u_i*, such that $\langle u, u_i \rangle$ is an edge in the graph.*

The OBDD representation of an AND-or graph is an OBDD that, given the index of two vertices (which includes their label AND or OR), determines if they are adjacent.

Problem (AGAP) The alternating graph accessibility problem is as follows.

Input: An AND-or graph G and vertices s and t in G.

Output: Is t reachable from s?

Theorem 8 *The AGAP problem for graphs represented by OBDDs is EXP-complete.*

Proof Because the AGAP problem is in P for graphs represented by adjacency matrices, it is in EXP for graphs represented by OBDDs.

Let x be a string that encodes a $2^{p_M(n)}$-time-bounded Turing machine M and an input y. We construct an AND-or graph with two special vertices s and t, such that t is reachable from s if and only if $x \in \mathcal{L}(U_{\text{EXP}})$. The construction of the graph is very similar to the construction of the circuit in the proof that circuit value is P-complete.

Once again let Quad $= \{(W, X, Y, Z) |$ if W, X, and Y are the symbols in the $(j-1)$th, jth, and $(j+1)$th cells, respectively, at some time instant, then Z is the symbol in the jth cell at the next time instant$\}$. Let $<$ be some ordering on the quadruples in Quad.

We construct the graph $G_{M,y}$ in stages, starting with the empty graph.

Stage 0 Add two AND nodes, one labeled 0 and the other 1. These nodes represent false and true, respectively.

Stage 1 For each $j, 0 \leq j \leq 2^{p_M(n)}$, and each X, where X is either a tape symbol or a composite symbol encoding a state of machine M and a tape symbol, add an OR node labeled $V_{0,j,X}$. Add the edge $\langle V_{0,j,X}, 1 \rangle$ if the jth symbol in the initial configuration of M on input y is X. Otherwise, add the edge $\langle V_{0,j,X}, 0 \rangle$.

Stage 2i For each j and k, add an AND node labeled $N_{i,j,k}$. Add edges $\langle N_{i,j,k}, V_{i-1,j-1,W}\rangle$, $\langle N_{i,j,k}, V_{i-1,j,X}\rangle$, and $\langle N_{i,j,k}, V_{i-1,j-1,Y}\rangle$, where the kth quadruple in Quad is (W, X, Y, Z), for some Z.

Stage 2i+1 For each j and symbol Z, add an OR node labeled $V_{i,j,Z}$. For each k, if (W, X, Y, Z) is the kth quadruple, for some W, X, and Y, then add the edge $\langle V_{i,j,Z}, N_{i,j,k}\rangle$.

Stage $2^{p_M(n)} + 2$ Add an OR node s. For all j, add edges $\langle s, V_{2^{p_M(n)},j,X}\rangle$, where X is a composite symbol encoding a final state and some tape symbol.

The basic idea of the construction is as follows. The node label $V_{i,j,X}$ means that, during the computation, at time i, the jth tape cell contains the symbol X. From the definition of Quad, it can be seen that

$$V_{i,j,Z} = \bigvee_{(W,X,Y,Z)\in \text{Quad}} (V_{i-1,j-1,W} \wedge V_{i-1,j,X} \wedge V_{i-1,j+1,Y}).$$

The string y is accepted if, at time $2^{p_M(n)}$, the machine reaches a final state, that is,

$$\bigvee_j \left(\bigvee_{X\in F} V_{2^{p_M(n)},j,X} \right),$$

where F is the set of composite symbols that encode a final state and symbol pair. Hence, the graph $G_{M,y}$ is such that node 1 is reachable from node s if and only if M accepts input y.

Claim 3 *The graph $G_{M,y}$ can be represented by a small OBDD.*

Proof There are no edges of the form $\langle V_{i_1,j_1,k_1}, V_{i_2,j_2,k_2}\rangle$ or $\langle N_{i_1,j_1,k_1}, N_{i_2,j_2,k_2}\rangle$. Also, the OBDD deciding whether there is an edge of the form $\langle V_{0,j,X}, 1\rangle$ is simple: Based on the value of j, just check if the jth symbol of the input is X. The case of $\langle V_{0,j,X}, 0\rangle$ is similar.

That leaves edges of only the following two forms: $\langle N_{i,j,k}, V_{i-1,j',X}\rangle$, where $j' = j$ or $j-1$ or $j+1$, and $\langle V_{i,j,X}, N_{i,j,k}\rangle$. In each case, determining whether nodes $V_{i_1,j_1,X}$ and $N_{I_2,j_2,k}$ are adjacent involves checking whether i_1, i_2 and j_1, j_2 differ by a constant and whether the symbol X occurs in the kth quadruple. That both these checks can be done by a small OBDD was seen in the proof of Theorem 4. ■

Thus the graph $G_{M,y}$ can be represented by a small OBDD that can be constructed in polynomial time. Furthermore, node 1 is reachable from s in $G_{M,y}$ if and only if $x \in \mathcal{L}(U_{\mathrm{EXP}})$. Hence, the AGAP problem is EXP-complete for graphs represented by OBDDs. ■

8 Open questions

The results that we prove in this paper are all negative: In the worst case, succinct encoding of instances using OBDDs generates problems that are hard for PSPACE, EXP, or NEXP. However, one of our main motivations for this investigation is the observed good performance of computer-aided verification tools on OBDD-encoded instances. Thus worst-case hardness results do not adequately capture the complexity of the problems on real-world instances. It would be desirable to have precise characterizations of the special cases that occur in practice and of the special cases that can be solved efficiently.

It would also be nice to have a general hardness result for OBDDs that is analogous to Papadmitriou and Yannakakis's result for circuits. In other words, is there a class of reductions such that if any problem is complete for NP via a reduction in the class, the problem is also complete for NEXP when instances are encoded as OBDDs? There has been some recent work in this direction (refer to [Vei98]). Recall from Definition 5 that an OBDD representation of a graph is an OBDD that encodes the adjacency relation on vertices. Veith [Vei98] obtains the general result for graphs that are encoded by OBDDs wherein the ordering of the variables is *fixed*, namely, the one where the bits of the first and second vertex are fed in alternating order. He shows that any problem that is complete for NP via *quantifier-free* reductions is also complete for NEXP when the instances are encoded by OBDDs with the above ordering. However, the question remains open for problems where instances are encoded by OBDDs with a different variable ordering.

9 Acknowledgement

It is a pleasure to thank Ed Clarke for helpful comments on an earlier draft of this paper. We would also like to thank Helmut Veith for communicating his results to us.

Acknowledgment of support

Kannan's work was done in part as a consultant to AT&T and supported in part by NSF grant CCR96-19910 and ONR Grant N00014-97-1-0505. Vardi's work was done as a visitor to DIMACS and Bell Laboratories as part of the DIMACS Special Year on Logic and Algorithms and was supported in part by NSF grants CCR-9628400 and CCR-9700061 and by a grant from the Intel Corporation. Viswanathan was supported by grants NSF CCR-9415346, NSF CCR-9619910, AFOSR F49620-95-1-0508, ARO DAAH04-95-1-0092, and ONR N00014-97-1-0505.

References

[Bal96] J. L. Balcázar. The complexity of search implicit graphs. *Artificial Intelligence*, 86:171–188, 1996.

[BBDG+94] I. Beer, S. Ben-David, D. Geist, R. Gewirtzman, and M. Yoeli. Methodology and system for practical formal verification of reactive hardware. In *Proceedings of the Sixth International Conference on Computer-Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 182–193. Springer-Verlag, Berlin/Heidelberg/New York, 1994.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98:142–170, 1992.

[Ber89] C. L. Berman. Ordered binary decision diagrams and circuit structure. In *Proceedings of the IEEE International Conference on Computer Design*, pages 392–395. IEEE Computer Society Press, Los Alamitos, CA, 1989.

[BL89] J. L. Balcázar and A. Lozano. The complexity of graph problems for succinctly represented graphs. In *Proceedings of Graph-Theoretic Concepts in Computer Science*, volume 411 of *Lecture Notes in Computer Science*, pages 277–285. Springer-Verlag, Berlin/Heidelberg/New York, 1989.

[BLT92] J. L. Balcázar, A. Lozano, and J. Torán. The complexity of algorithmic problems on succinct instances. In R. Baeza-Yates and

U. Manber, editors, *Computer Science*, pages 351–377. Plenum Press, New York, 1992.

[Bry86] R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, (C-35):677–691, 1986.

[Bry92] R. Bryant. Symbolic manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24:293–318, 1992.

[CGH+95] E. M. Clarke, O. Grumberg, H. Hiraishi, S. Jha, D. E. Long, K. L. McMillan, and L. A. Ness. Verification of the futurebus+ cache coherence protocol. *Formal Methods in System Design*, 6:217–232, 1995.

[CLR89] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, 1989.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on the Theory of Computing*, pages 151–158. ACM, New York, 1971.

[EL86] E. A. Emerson and C. L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings of the First IEEE Symposium on Logic in Computer Science*, pages 267–278. IEEE Computer Society Press, Los Alamitos, CA, 1986.

[GW83] H. Galperin and A. Wigderson. Succinct representations of graphs. *Information and Control*, 56:183–198, 1983.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.

[Kur94a] R. Kurshan. The complexity of verification. In *Proceedings of the Twenty-sixth ACM Symposium on the Theory of Computing*, pages 365–371. ACM, New York, 1994.

[Kur94b] R. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton University Press, Princeton, 1994.

[KV96] O. Kupferman and M. Y. Vardi. Module checking. In *Proceedings of the Eighth International Conference on Computer-Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 75–86. Springer-Verlag, Berlin/Heidelberg/New York, 1996.

[McM93] K. McMillan. *Symbolic Model Checking*. Kluwer, Amsterdam, 1993.

[Pon95a] S. Ponzio. A lower bound for integer multiplication with read-once branching programs. In *Proceedings of the Twenty-seventh ACM Symposium on the Theory of Computation*, pages 130–139. ACM, New York, 1995.

[Pon95b] S. Ponzio. *Restricted branching programs and hardware Verification*. PhD thesis, MIT, 1995.

[PY86] C. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71:181–185, 1986.

[SV82] S. Skyum and L. Valiant. A complexity theory based on Boolean algebra. In *Proceedings of the Twenty-third IEEE Symposium on the Foundations of Computer Science*, pages 244–253. IEEE Computer Society Press, Los Alamitos, CA, 1982.

[Tor88] J. Torán. Succinct representations of counting problems. In *Proceedings of the Sixth International Conference on Applied Algebra, Algebraic Algorithms, and Error-Correcting Codes*, volume 357 of *Lecture Notes in Computer Science*, pages 415–426. Springer-Verlag, Berlin/Heidelberg/New York, 1988.

[Vei95] H. Veith. Languages represented by Boolean formulas. Technical Report TR CD 85/95, TU Vienna, 1995.

[Vei96] H. Veith. Succinct representation, leaf languages, and projection reductions. In *Proceedings of the Eleventh IEEE Conference on Computational Complexity*, pages 118–126. IEEE Computer Society Press, Los Alamitos, CA, 1996.

[Vei98] H. Veith. How to encode a logical structure as an OBDD. In *Thirteenth IEEE Conference on Computational Complexity*, pages 122–131. IEEE Computer Society Press, Los Alamitos, CA, 1998.