Analysis and Manipulation of Boolean Functions in Terms of Decision Graphs *

Jordan Gergov and Christoph Meinel

FB IV - Informatik
Universität Trier
Postfach 3825
D-5500 Trier

Abstract. We investigate the question whether and to what extend the solution of central tasks of digital logic circuit design of a given Boolean function f benefits from a representation of f in terms of certain restricted decision graphs or branching programs.

Introduction

One of the fundamental problems in computer-aided circuit design is the task of representing logic functions. Although, in principle, any valid representation is allowed, some representations may be preferred because they are
– more efficient in memory,
– more efficient to manipulate, or
– more indicative of the complexity of the final implementation.
The search of an optimal trade–off between these competing objectives – succinct representation of Boolean functions and feasible manipulation algorithms – is a central theme of logic synthesis.
The most fundamental concept in the description of logic functions is that of the *truth table*. Truth tables define a function by listing the output value for each possible input combination. Since, for n independent input variables, the size of such a truth table is proportionally to 2^n such representations are unfeasible already for moderate n. This situation has motivated many investigations of alternative representations for logic functions. Well studied and of great importance in circuit design are *two-level normal forms* such as the *disjunctive* (DNF) or the *conjunctive normal form* (CNF). In circuit theory multi-level representations, such as *Boolean formulas* or *circuits* have found considerable importance. Although, due to an easy counting argument, for each of the mentioned types of representations it can be shown that almost all Boolean functions require representations of exponential length many functions of practical interest (e. g. all functions that can be computed in polynomial time) possess at least succinct (i. e. polynomial length) circuit representations. Unfortunately, it is rather difficult to analyse and manipulate Boolean functions that are represented by

* Proc. of the *Workshop on Graph–Theoretic Concepts in Computer Science*, LNCS 657, 310–320, Springer Verlag, 1993

Boolean circuits. For that reason in computer-aided circuit design one is interested in other representations that are, on one hand, more succinct than two-level representations and, on the other hand, more easy to analyse and manipulate than Boolean circuits.

One candidate which seems to satisfy these two conditions is the concept of decision graphs or branching programs. Branching programs, introduced by Lee [Lee59], are known from complexity theory as a favorite computational model for computing logic functions [e. g. Mei89]. The use of branching programs as data structure for representing Boolean functions was proposed by Akers [Ake78]. Indeed, recently, an extremely restricted type of branching program, so-called restricted *OBDD*s [Bry86], has proved to be of great importance in logic synthesis [e. g. BCMD90, CG85, FFK88, MKLC88]. In order to get more succinct branching program representations that are, nevertheless, quite easy to analyse and to manipulate Meinel [Mei91] proposes to investigate the use of some more general types of restricted branching programs than *OBDD*s are. On the basis of the results presented in this paper Gergov and Meinel were able to prove that read-once-only BPs can be analysed and manipulated similar efficient as OBDDs [GM92].

The paper is organized as follows. In the first section, we cite the decision graph or branching program concept and introduce some types of restricted branching programs. Our main concern in doing this is to provide some insight in the differences between the various models. The second section is devoted to the discussion of some algorithms for analysing and manipulating Boolean functions in terms of the different branching programs. In detail we investigate

– the satisfiability and tautology problem,
– the equivalency and nonequivalency problem,
– the simultaneous satisfiability problem, and
– various synthesis problems.

1 Representing Boolean Functions in Terms of Decision Graphs

1.1 Branching Programs and Their Complexity

A *decision graph* or *branching program* (*BP*) [e. g. Mei89] is a directed acyclic graph where each node has outdegree 2 or 0. There is a distinguished node, the *source*, which has indegree 0. Nodes with outdegree 0 are called *sinks* and are labelled by Boolean constants 0 or 1. The remaining nodes are labelled by Boolean variables taken from a set $X = \{x_1, \ldots, x_n\}$. The two edges starting in a non-sink node are labelled by 0 and 1, respectively.

A branching program P over $X = \{x_1, \ldots, x_n\}$ can be used to *represent* or *compute* a Boolean function $f \in I\!B_n$ in the following way. Each input $w = w_1 \ldots w_n \in \{0,1\}^n$ defines a *computation path* through P that starts at the

source. If this path reaches a node v which is labelled by the variable $x_i \in X$ then it follows the edge labelled by w_i. Since P is finite and acyclic finally a sink is reached whose label gives the value $f(w)$ on w. P is said to be a BP-representation of f.
Indeed, each Boolean function can be represented by a BP: Due to the classical Shannon expansion formula a given function $f(x_1, \ldots, x_n)$ can be written as

$$f(x_1, \ldots, x_n) = x_1 f(1, x_2, \ldots, x_n) \vee \overline{x}_1 f(0, x_2, \ldots, x_n).$$

Hence, a BP P for f can be easily constructed from BPs P_δ, $\delta \in \{0, 1\}$, for $f_\delta(x_2, \ldots, x_n) = f(\delta, x_2, \ldots, x_n)$, by introducing a new source v_0, labeling v_0 by x_1, and connecting v_0 with the sources of P_0 and P_1 via two edges labelled by 0 and 1, respectively. Let us only remark that this procedure results, for each Boolean function $f \in \mathbb{B}_n$ in a tree program of size proportional to 2^n. Fortunately, many functions possess much more compact BP-representations.
The most important complexity measure for BPs is their *size* which equals the number of non-sink nodes. In order to characterize the size of a BP in dependence of the number n of input variables we consider *sequences* of BPs.
Let $BP(f)$ be the *minimal size* of a BP that represents f. Of course, for practical purposes the set $\mathcal{P}_{BP}$ of (sequences of) Boolean functions having polynomial size BP-representations is of special interest. Its characterization in terms of logarithmic space-bounded Turing machines is well-known.

Fact 1. [PŽ83]
Polynomial size branching programs and logarithmic space-bounded nonuniform Turing machines are of the same computational power,

$$\mathcal{P}_{BP} = DSPACE(log\, n)/poly. \quad \square$$

Hence, many Boolean functions of practical interest possess polynomial size BP-representations. The class of these functions includes, for example, *storage access functions, sorting and order statistics, arithmetic functions, computations with matrices, Boolean convolution, computations with permutations, solving certain graph problems, symmetric functions* etc.
However, similar as in the case of other representations of Boolean functions (e. g. *two-level normal forms, decision trees, formulas, Boolean circuits*) Shannon's counting argument proves that almost all Boolean functions require exponential size BP-representations.

1.2 Restricted Types of Branching Programs

BPs and BP-representations have been investigated often under additional constraints. Such investigations were motivated by very practical demands as well as by pure theoretical questions. Designing CAD-tools for logic synthesis one has to represent Boolean functions in such a way that certain tasks such as
– optimizing representations,

– synthesizing new functions, or
– testing satisfiability, tautology, or equivalence
can be carried out as efficient as possible. Of course, the complexity of these tasks depends on the Boolean functions under consideration while the efficiency of the algorithms that solve these tasks depends on the representation of that functions. Hence, there is a fundamental trade-off between succinctness of the representation and efficiency of algorithms. In order to cope with this situation for practical purposes it is often much more useful to work with restricted types of BP-representations than with unrestricted ones.
On the other hand, in order to estimate the efficiency of the representations of Boolean functions one has, beside of giving upper bounds, to prove (nontrivial) lower bounds for concrete functions. While, up to now, this central task of complexity theory remains unsolved for general branching programs and Boolean circuits first exponential lower bounds could be obtained for some restricted types of branching programs [e. g. Žak84, Bry86, KMW88, Weg88, DM89, MW91].
In the following we introduce important types of restricted BPs.

Decision Trees
Decision trees (DT) are BPs whose underlying graph is a tree. $DT(f)$ denotes the size of a minimal DT that represents f. By $\mathcal{P}_{DT}$ we denote the set of Boolean functions representable by polynomial size decision trees. As already mentioned, by means of the Shannon expansion for each Boolean function there is DT-representation.
For a survey of DTs and DT–applications see e. g. Moret [Mor82].

Ordered Branching Programs
Let $\pi = (i_1, \ldots, i_n)$ be an order of the set $\{1, \ldots, n\}$ of indices of the variables x_i tested in a BP P. P is an *ordered branching program with respect to* π ($\pi BP1$) if on every source–to–sink path, the input variables are tested according to π. Investigating the complexity of $\pi BP1$–representations, we consider π as a sequence of orders $\pi = (\pi_n)$, $n \in I\!N$. If π remains unspecified we speak shortly of OBDDs (ordered binary decision diagrams).
Since the Shannon expansion provides an $OBDD$ (actual a πDT) each Boolean function can be represented by means of an $OBDD$. By $\mathcal{P}_{OBDD}$ we denote the set of all Boolean functions representable by means of polynomial size $OBDDs$.

Read-k-Times-Only Branching Programs
BPs where each variable appears ("is read"), on each path, at most k, $k \in I\!N$, times are called *read-k-times-only branching programs* (BPk). The set of Boolean functions that can be represented by a polynomial size BPk is denoted by $\mathcal{P}_{BPk}$. Of special interest is the case $k = 1$. In that case we speak usually of *read-once-only branching programs* ($BP1$).
Since the $BP1s$ generalize DTs as well as $OBDDs$ each Boolean function can be represented by means of a $BP1$, and, hence, by means of a BPk. Particularly,

for each $k,\ k \geq 1$, we have

$$\mathcal{P}_{DT},\ \mathcal{P}_{OBDD}\ \subseteq\ \mathcal{P}_{BPk}.$$

Results of [Weg88,BHR91,DM89] show that we have

$$\mathcal{P}_{DT},\ \mathcal{P}_{OBDD}\ \subset\ \mathcal{P}_{BPk}.$$

Synchronous Branching Programs

A branching program P is said to be *synchronous* (sBP) if for each node v of P all source-to-v paths are of the same length. The nodes of P having the same distance to the source constitute a *level* of P.

By adding dummy nodes, each BP easily can be synchronized, possibly squaring the size but keeping the length fixed. Hence, each Boolean function can be represented by means of a sBP. Moreover, if $\mathcal{P}_{sBP}$ denotes the set of Boolean functions representable by polynomial size $sBPs$ then we have

$$\mathcal{P}_{sBP} = \mathcal{P}_{BP}.$$

Width-Bounded Branching Programs

Let P be a synchronous branching program over X and let $w(P)$ be the maximal size of the levels of P. P is said to be *of width* k for $k \in I\!N$ (*width-k BP*) if $w(P) \leq k$. Since each disjunctive normal form can be simulated by a *width-2 BP* each Boolean function can be represented by a *width-k BP*, $k \geq 2$. By $\mathcal{P}_{width-k\,BP}$ we denote the set of Boolean functions representable by polynomial size *width-k BPs*. If k remains unspecified we speak of *width-bounded branching programs* ($bwBP$). According to Barrington [Bar89] we have $\mathcal{P}_{bwBP} = NC^1$.

Oblivious Branching Programs

Oblivious branching programs (oBP) are $sBPs$ with the property that all nodes of a level are labelled by one and the same input variable. Since, for instance, $OBDDs$ are $oBPs$ each Boolean function can even be represented by an oBP of linear length. By $\mathcal{P}_{length-l\ oBP}$ we denote the set of Boolean functions representable by polynomial size $oBPs$ of length $l(n)$ (the length of a oBP P is the number of levels in P).

For *length-$O(n)$ oBP*-representations of explicitely defined Boolean Funcitons exponential lower size-bounds could be proved by means of arguments from communication complexity theory [AM86, KW89].

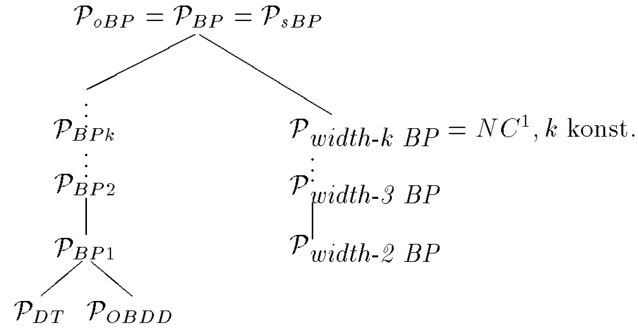

Figure 1. *Inclusion structure.*

1.3 Inclusion structure

Figure 1 shows the inclusion structure between the important sets $\mathcal{P}_{DT}$, $\mathcal{P}_{OBDD}$, $\mathcal{P}_{BP1}$, $\mathcal{P}_{BPk}$, $\mathcal{P}_{width-k\,BP}$, $\mathcal{P}_{BP}$, $\mathcal{P}_{sBP}$, and $\mathcal{P}_{oBP}$ of Boolean functions representable by restricted BPs. For more details we refer to [Mei91].

2 Analysis and Manipulation of Boolean Functions in Terms of Branching Programs

In the following section we consider the question whether and to what extend the solution of central tasks of the digital logic circuit design of a given Boolean function $f \in \mathbb{B}_n$ benefits from a representation of f in terms of certain decision graphs or branching programs. In detail, we investigate
- the satisfyability and tautology problem,
- the equivalency and nonequivalency problem,
- the simultaneous satisfiability problem, and
- various synthesis problems.

2.1 Satisfiability and Tautology

The *satisfiability problem* for a Boolean function f, $f \in \mathbb{B}_n$, consists in the decision whether f is different from the constant *zero*-function. Similarly, the *tautology problem* for f consists in the decision whether f is equal to the constant *one*-function. Both problems are central problems in logic synthesis.
Let *X-BP* be a set of certain restricted BPs. By SAT_{X-BP} and $TAUT_{X-BP}$ we denote the *satisfiability problem* and the *tautology problem* for branching programs from the set *X-BP*, respectively.
In the following we identify some properties of the defining branching programs from which it depends whether SAT_{X-BP} and $TAUT_{X-BP}$ are *feasible* (i.e. computable in polynomial time) or *unfeasible* (e.g. NP-complete).

Proposition 2.

1. *If X-BP denotes a set of BPs all of whose paths are computation paths then SAT_{X-BP} is feasible.*
2. *If X-BP denotes a set of BPs all of whose paths are computation paths then $TAUT_{X-BP}$ is feasible.*

Proof.
Let P be a BP from *X-BP* all of whose paths are computation paths. W.l.o.g. we assume that P possesses merely one 1-sink.

1. The decision whether P is satisfiable is equivalent to the decision whether there is a computation path leading from the source v of P to the 1-sink u. Since each path of P is a computation path one has merely to test whether there is a path from v to u. This decision requires the solution of the *graph accessibility problem GAP* for (P, v, u) which is known to belong to NL and, consequently, to P. Hence, the satisfiability problem for *X-BPs* can be efficiently solved in polynomial time (measured in the size of P).
2. P represents the constant *one* function if there is no computation path from the source v to the 0-sink w. Hence, the solution of the graph accessibility problem *all-path-GAP* [Mei89] (P, u, w) provides a solution of the tautology problem. Since *all-path-GAP* belongs to $co{-}NL = NL \subseteq P$ the tautology problem for branching programs all of whose paths are computation paths can be efficiently solved in polynomial time, too. □

Proposition 3.

1. $SAT_{width-2\ sBP3}$ *is NP-complete.*
2. $TAUT_{width-2\ sBP3}$ *is co-NP-complete.*

Proof.
In order to prove Proposition 3 we reduce the *NP*-complete Problem 3CNF to $SAT_{width-2\ sBP3}$. Similarely we prove the *co-NP*-completeness of $TAUT_{width-2\ sBP3}$. □

Proposition 4.

1. $SAT_{width-3\ oBP2}$ *is NP-complete.*
2. $TAUT_{width-3\ oBP2}$ *is co-NP-complete.*

Proof.
In order to prove Proposition 3 we reduce the *NP*-complete problem 3CNF to $SAT_{width-3\ oBP2}$. □

Concerning the classes of restricted BP-representations introduced in Section 1 we obtain

Theorem 5.

1. *Let* $X\text{-}BP \in \{oDT,\ DT,\ OBDD,\ BP1\}$.
   Then SAT_{X-BP} *and* $TAUT_{X-BP}$ *are feasible.*

2. *Let* $X\text{-}BP \in \{BPk,\ width\text{-}k\,BP\ (k \geq 2),\ oBP,\ sBP,\ BP\}$.
Then SAT_{X-BP} *and* $TAUT_{X-BP}$ *are* NP*- resp. co-*NP*-complete.* □

2.2 Equivalence Problem

The *general equivalence problem* (sometimes also called *formal verification*) consists in the verification whether two given representations of Boolean functions denote the same function. This problem often arises in the context of checking whether the overall effect of some logic synthesis tool has altered the function under consideration. Concerning unrestricted BP-representations then, similar as in the case of other representations this problem is known to be *co-*NP-complete [CHS74].
The question arises whether it is possible to design feasible algorithms to solve the equivalence problem for restricted BP-representations. Let *X-BP* be a set of certain restricted BPs. By EQU_{X-BP} we denote the *equivalence problem* for branching programs from the set *X-BP*. The dual problem is denoted by $NONEQU_{X-BP}$.
Since, for the trivial programs 1 and 0, we have

$$TAUT_{X-BP}(P) = EQU_{X-BP}(P,1) \quad \text{and} \quad SAT_{X-BP}(P) = NONEQU_{X-BP}(P,0)$$

Propositions 3 and 4 provide

Proposition 6.

1. $EQU_{width-2\ sBP3}$ *is co-NP-complete.*
2. $NONEQU_{width-2\ sBP3}$ *is NP-complete.* □

Proposition 7.

1. $EQU_{width-3\ oBP2}$ *is co-NP-complete.*
2. $NONEQU_{width-3\ oBP2}$ *is NP-complete.* □

On the other side, there are, for some restricted BP-representations, polynomial time algorithms (measured in the size of the program) that solve (in one case at least approximately) the equivalence problem.

Proposition 8.

1. EQU_{DT} *and* $NONEQU_{DT}$ *are feasible.*
2. $EQU_{\pi BP1}$ *and* $NONEQU_{\pi BP1}$ *are feasible.*

Proof.
Due to results of the next section we can perform the parity synthesis ($\oplus$-SYN) of decision trees and $\pi BP1s$ in polynomial time. Hence, performing the $\oplus$-SYN on the input programs and testing the resulting program for unsatisfiablility (satisfiability) (Proposition 2) we efficiently solve the considered equivalence (nonequivalence) problem. □

Concerning $BP1$-representations Blum, Chandra and Wegman have given an algorithm that solves the equivalence problem probabilistically in polynomial time.

Proposition 9. [BCW80]
EQU_{BP1} can be decided probabilistically in polynomial time. □

Altogether we get

Theorem 10.

1. *Let X-$BP \in \{oDT,\ DT,\ \pi BP1,\ OBDD\}$.*
   Then EQU_{X-BP} and $NONEQU_{X-BP}$ are feasible.
2. *Let X-$BP = BP1$.*
   Then EQU_{X-BP} and $NONEQU_{X-BP}$ can be decided probabilistically in polynomial time.
3. *Let X-$BP \in \{BPk,\ width\text{-}k\,BP\ (k \geq 2),\ oBP,\ sBP,\ BP\}$.*
   Then EQU_{X-BP} and $NONEQU_{X-BP}$ are not feasible(co-NP-complete and NP-complete, respectively). □

2.3 Simultaneous Satisfiablility

Let *X-BP* be a set of certain restricted BPs. In the context of logic synthesis it is often desirable to test whether two BPs from *X-BP* compute (simultaneously) the value 1 for an some input. By CP_{X-BP} ("common paths") we denote the restriction of this problem to BPs from X-BPs.

Proposition 11.
$CP_{width-3\ OBDD}$ is NP-complete.

Proof.
The proof is given by a reduction from $3CNF$ to $CP_{width-3\ OBDD}$. □

Immediately we obtain

Corollary 12.
If $OBDD \subseteq X$-BP then CP_{X-BP} is NP-complete. □

In the next section the feasibility of CP_{DT} and $CP_{\pi BP1}$ will become evident due to Proposition 15.

2.4 Synthesis

The process of designing large digital systems is typically one of synthesizing smaller devices in such a way that the resulting system provides the required behaviour. Hence, a fundamental task in logic synthesis is to (logical) compose Boolean functions from certain subfunctions. Of course, the efficiency of performing this task depends again on the kind in which the functions are represented.

Let X-BP be a class of restricted BPs and let $*$ be an operation for Boolean functions. By $*$-SYN_{X-BP} we denote the problem of constructing a X-BP-representation P of $*(f_1, f_2, \ldots, f_m)$ from the X-BP-representations $P_1, P_2, \ldots, P_m$ of the functions $f_1, f_2, \ldots, f_m$.
If the functions under consideration are represented by unrestricted branching programs then it is quite easy to derive a BP-representation for the resulting function.

Proposition 13.
$*$-SYN_{BP} is feasible for each binary Boolean operation $*$. $\square$

In order to give some examples, assume the function f is composed of two functions f_1 and f_2 that are represented by branching programs P_1 and P_2. Identifying the 1-sink (0-sink) of P_1 with the source of P_2 one obtains a branching program for the AND (OR) of f_1 and f_2. If one identifies both sinks of P_1 with the source of different copies of P_2 and if one interchanges the roles of the 0-sinks and the 1-sinks in the copy of P_2 that is connected with the 1-sink of P_1 one obtains a BP-representation for the EX-OR of f_1 and f_2.
The situation becomes much more difficult if we want to solve $*$-SYN_{X-BP} for a class of restricted X-BPs since it can be quite difficult to maintain the restricting assumptions in the course of the synthesis.
In the following we consider the binary synthesis. Recall, there are two different types of proper binary Boolean operations: $(x^\alpha \wedge y^\beta)^\gamma$ and $(x \oplus y)^\alpha$, $\alpha, \beta, \gamma \in \{0, 1\}$, where $x^0 = \neg x$, $x^1 = x$. Since the complement of a branching program can be constructed easily in constant time, we have to consider only two types of binary synthesis: *and–synthesis* ($\wedge$-SYN) and *parity–synthesis* ($\oplus$-SYN).

Proposition 14.
Let X-BP be a set of restricted BPs with $OBDD \subseteq X$-$BP \subseteq BP1$.
Then $\wedge$-SYN_{X-BP} is NP-hard.

Proof.
Proposition 14 is implied by Proposition 12, since we can solve CP_{X-BP} by applying an algorithm for $\wedge$-SYN_{X-BP} on the input programs and deciding SAT for the resulting program (Proposition 2). $\square$

Fortunately, there are examples of restricted types of branching programs for which quite efficient synthesis algorithms are known. As a first example of such BP–types we mention DTs. Considering πBPs for pre-defined orders π Bryant has given an algorithm that performs binary operations in time proportional to the product of the sizes of the input programs [Bry86]. Hence, as long as the number of operations to be performed is constantly bounded this algorithm provides feasible $\pi BP1$-representations of the synthesized function.

Proposition 15.
Let X-$BP \in \{DT, \pi BP1$ (the input BPs are of the same order π)$\}$.
Then $\wedge$-SYN_{X-BP} and $\oplus$-SYN_{X-BP} are feasible. $\square$

Altogether we get

Theorem 16.

1. *Let X-BP* $\in \{oBP,\ sBP,\ BP\}$.
   Then $\wedge$-SYN_{X-BP} *and* $\oplus$-SYN_{X-BP} *are feasible.*
2. *Let X-BP be with* $OBDD \subseteq X\text{-}BP \subseteq BP1$.
   Then $\wedge$-SYN_{X-BP} *is* NP*–hard.*
3. *Let X-BP* $\in \{DT,\ \pi BP1$ *(*π *is a fixed order)*$\}$.
   Then $\wedge$-SYN_{X-BP} *and* $\oplus$-SYN_{X-BP} *are feasible.* □

References

[Ake78] S. B. Akers: Binary Decision Diagrams, IEEE Trans. Computing, Vol C-27, No. 6, 509–516.

[AM86] N. Alon, W. Maass: Meanders, Ramsey Theory and Lower Bounds, Proc. 27th ACM STOC, 1986, 30–39.

[Bar89] D. A. Barrington: Bounded-Width Polynomial-Size Branching Programs Recognize Exactly those Languages in NC^1, JCSS 38, 1989, 150–164.

[BCW80] M. Blum, A. K. Chandra, M. N. Wegman: Equivalence of Free Boolean Graphs Can Be Decided Probabilistically in Polynomial Time, IPL 10, 2, 1980, 80–82.

[BHR91] Y. Breitbart, H. B. Hunt III, D. Rosenkrantz: The Size of Binary Decision Diagrams Representing Boolean Functions, submitted to Inf. and Comp. , 1991.

[Bry86] R. E. Bryant: Graph-Based Algorithms for Boolean Function Manipulation, IEEE Trans. Computers, C-35, 8, 1986, 677–691.

[BC&90] J. R. Burch, E. M. Clarke, K. L. McMillan, D.L. Dill: Sequential Circuit Verification Using Symbolic Model Checking, Proc. 27th IEEE DAC'90, 1990, 46–51.

[CG85] E. Cerny, J. Gecse: Simulation of MOS Circuits by Decision Diagrams, IEEE Trans. Computer Aided Design, CAD-4, 4, 1985, 685–693.

[CHS74] R. L. Constable, H. B. Hunt III, S. Sahni: On the Computational Complexity of Scheme Equivalence, Proc. 8th Princeton Conf. on Information Sciences and Systems, 1974.

[DM89] C. Damm, Ch. Meinel: Separating Complexity Classes, Related to Polynomial Size Ω-decision Trees, Proc. FCT'89, LNCS 380, 127–136.

[FFK88] M. Fuita, H. Fujisawa, N. Kawoto: Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams, Proc. IEEE ICCAD'88, 1988, 2–5.

[GM92] J. Gergov, Ch. Meinel: Efficient Analysis and Manipulation of OBDDs Can Be Extended to Read-once-only Branching Programs, Forschungsbericht Nr. 92-10, Universität Trier, 1992.

[KMW88] M. Krause, Ch. Meinel, S. Waack: Separating the Eraser Turing Machine Classes $\mathcal{L}_e$, $\mathcal{NL}_e$, *co*-$\mathcal{NL}_e$ and $\mathcal{P}_e$, Proc. MFCS'88, LNCS 324, 405–413.

[KW89] M. Krause, S. Waack: On Oblivious Branching Programs of Linear Length, Proc. FCT'89, LNCS 380, 287–296.

[Lee59] C. Lee: Representation of Switching Circuits by Binary Decision Programs, Bell Systems Technical Journal 38, 1959, 985–999.

[Mei89] Ch. Meinel: Modified Branching Programs and Their Computational Power, Springer-Verlag, LNCS 370, 1989.

[Mei91] Ch. Meinel: Branching Programs - An Efficient Data Structure for Computer-Aided Circuit Design, Preprint UGH Paderborn, Nr. 93, 1991.

[MKLC89] S. Muroga, Y. Kambayashi, H. C. Lai, J. N. Culliney: The Transduction Method, IEEE Trans. Computers, C-38, 1989, 1404–1424.

[Mor82] B. M. E. Moret: Decision Trees and Diagrams, ACM Computing Surveys, 14, 2, 1982, 593–623.

[MW91] Ch. Meinel, S. Waack: Separating Complexity Classes Related to Bounded Alternating ω-Branching Programs, Preprint TU Berlin, No. 276/1991.

[PŽ83] P. Pudlak, S. Žak: Space Complexity of Computations, Preprint, Univ. Prague, 1983.

[Weg88] I. Wegener: On the Complexity of Branching Programs and Decision Trees for Clique Functions, JACM Vol. 35, 2, 1988, 461–471.

[Žak84] S. Žak: An Exponential Lower Bound for One-Time-Only Branching Programs, Proc. MFCS'84, LNCS 176, 562–566.