Exercises - Solutions

Chapter 1

Exercise 1.1. The first two definitions are equivalent, since we follow in both cases the unique path leading from v to a sink and using only a_i-edges leaving x_i-nodes. Shannon's decomposition rule $f_v(a) = a_i f_1(a) + \overline{a}_i f_0(a)$ prescribes that we choose the value computed at the a_i-successor of an x_i-node. Moreover, the value of a sink is equal to its label. Shannon's decomposition rule can be rewritten for an x_i-node v with a 0-successor representing f_0 and a 1-successor representing f_1 by $f_v(a) = ite\big(a_i, f_1(a), f_0(a)\big)$ leading to the special ite straight line programs described in Definition 1.1.5.ii.

Exercise 1.2. If the depth is bounded by $c \log n$, the size is bounded by $n^c - 1$. The output gate has two predecessors which each have two predecessors and so on. Some of the gates are perhaps counted more than once. Sometimes we may read an input before we have considered $c \log n$ levels. In any case, the number of gates is not larger than the number of inner nodes of a binary tree whose depth is bounded by $c \log n$.

Exercise 1.3. The ideas of the solution of this exercise are also used in the proof of Theorem 14.2.1. Let the number of levels of the considered BPs be bounded by the polynomial $p(n)$. Let v_{ij} be the jth node at level i, $0 \leq i \leq p(n)$, $1 \leq j \leq 5$. In particular, let $v_{0,1}$ be the source and let $v_{p(n),1}$ be the only 1-sink. Let R_{kl} be a 5×5-matrix containing at position (s,t) the Boolean function accepting all inputs such that we reach v_{lt} if we start at v_{ks}. If $l = k+1$ and v_{ks} is an x_i-node, then $R_{k,k+1}$ contains at position (s,t) the function 1 iff both edges leaving v_{ks} lead to $v_{k+1,t}$, the function 0 iff no edge leaving v_{ks} leads to $v_{k+1,t}$, the function x_i iff the 1-edge but not the 0-edge leaving v_{ks} leads to $v_{k+1,t}$ and the function $\overline{x}_i$ otherwise. All these functions can be represented in depth 1. The essential idea is that R_{kl} where $k < m < l$ is the Boolean matrix product of $R_{km}$and R_{ml}. The Boolean matrix product C of an $m_1 \times m_2$-matrix A and an $m_2 \times m_3$-matrix B is defined by

$$c_{ij} = \bigvee_{1 \leq r \leq m_2} a_{ir} \wedge b_{rj}, \quad 1 \leq i \leq m_1, 1 \leq j \leq m_3.$$

We reach v_{lt} from v_{ks} iff there is some r such that we reach v_{mr} from v_{ks} and v_{lt} from v_{mr}. In our case, $m_1 = m_2 = m_3 = 5$. Such a Boolean matrix product can be realized by a Boolean circuit of constant size and depth. We have to realize $R_{0,p(n)}$ and do this by recursively computing $R_{0,\lceil p(n)/2 \rceil}$ and $R_{\lceil p(n)/2 \rceil, p(n)}$ and by multiplying the results. This leads to a balanced tree with $p(n)$ subcircuits realizing the Boolean matrix product of 5×5-matrices. Hence, the depth is bounded by $O\big(\log p(n)\big) = O\big(\log n\big)$. It is obvious that this method also works for polynomial-size BPs whose width is bounded by an arbitrary constant c.

Exercise 1.4. The number of different objects which can be described by words whose bit length is bounded by 2^{n-1} is smaller than $2^{2^{n-1}+1}$. The number of Boolean functions $f \in B_n$ equals 2^{2^n}. Finally,

$$2^{2^{n-1}+1}/2^{2^n} = 2 \cdot 2^{-2^{n-1}}$$

tends even double exponentially fast to 0 as $n \to \infty$.

Exercise 1.5. See the proof of Theorem 4.4.3 for a solution of this exercise. The resulting BDD is even an OBDD.

Exercise 1.6. See the proof of Theorem 2.3.3 for a solution of this exercise.

Chapter 2

Exercise 2.1. The first n instructions realize $\overline{x}_1, \ldots, \overline{x}_n$. If a circuit gate G realizes a function h, we realize h and $\overline{h}$. Let g_1 and g_2 be the functions represented at the direct predecessors of G. Then $h = g_1 \oplus g_2 \oplus a$ for some $a \in \{0,1\}$ or $h = (g_1 \oplus a) \wedge (g_2 \oplus b) \oplus c$ for some $a, b, c \in \{0,1\}$. We already have computed g_1, $\overline{g}_1$, g_2, and $\overline{g}_2$. Then $g_1 \oplus g_2 = \mathit{ite}(g_1, \overline{g}_2, g_2)$ and $g_1 \wedge g_2 = \mathit{ite}(g_1, g_2, 0)$ and the other cases can be handled similarly to compute h and $\overline{h}$ with two instructions.

Exercise 2.2. Let f be represented by a formula of size $L(f)$ and depth d. An EXOR-gate computing $g = g_1 \oplus g_2$ can be replaced by $(g_1 \wedge \overline{g}_2) + (\overline{g}_1 \wedge g_2)$, similarly for negated EXOR-gates. We obtain a formula for f without EXOR- and NOT-EXOR-gates whose depth is bounded by $2d$. This implies that its size is bounded by 2^{2d}. Hence, the main step is to construct a formula for f whose depth is bounded by $d^* = O\big(\log L(f)\big)$. Afterwards, it is sufficient to apply the simulation described above.

Spira has proved (already in 1971) that a binary Boolean formula F of size l can be simulated by a formula whose depth is bounded above by $c \log(l+1)$ where $c = 2\log^{-1}(3/2) \approx 5.13$. This statement can be proved by induction on l. The statement is trivial if $l \leq 2$. If $l \geq 3$, we consider the two subformulas F_1 and F_2 which are combined in F by the last gate. Then $f = f_1 \otimes f_2$ for a Boolean operator $\otimes$ and the functions f_1 and f_2 represented by F_1 and F_2 resp. Let l_1 be the size of F_1 and l_2 the size of F_2. W.l.o.g. $l_1 \leq l_2$. Since $l_1 + l_2 + 1 = l$, also $0 \leq l_1 \leq l/2 - 1/2$ and $1 \leq (l-1)/2 \leq l_2 \leq l - 1$. We start at the output of F_2 and choose always the predecessor which is the root of the larger subtree. Ties can be broken arbitrarily. Let v be the last node on this path which is the root of a tree with at least $\lceil l_2/3 \rceil$ nodes. The subformula with root v is called F_0, it represents f_0 and has size l_0. We know that $l_0 \geq \lceil l_2/3 \rceil$. Since the two subformulas of F_0 contain at most $\lceil l_2/3 \rceil - 1$ nodes each,

$$l_0 \leq 2(\lceil l_2/3 \rceil - 1) + 1 \leq 2l_2/3 + 1/3 \leq 2l/3 - 1/3.$$

Let $F_{2,a}$ be the formula obtained from F_2 by replacing F_0 by the constant $a \in \{0,1\}$ and let $f_{2,a}$ be the function represented by $F_{2,a}$. The size of $F_{2,a}$ equals

$$l_2 - l_0 \leq l_2 - \lceil l_2/3 \rceil \leq 2l_2/3 \leq 2l/3 - 1/3.$$

It is easy to see that

$$f_2 = (f_0 \wedge f_{2,1}) + (\overline{f}_0 \wedge f_{2,0})$$

and

$$f = f_1 \otimes f_2 = f_1 \otimes \big[(f_0 \wedge f_{2,1}) + (\overline{f}_0 \wedge f_{2,0})\big]$$

can be computed from f_1, f_0, $f_{2,1}$, and $f_{2,0}$ by a formula of depth 3. Since the formula size of each of the functions f_1, f_0, $f_{2,1}$, and $f_{2,0}$ is bounded by $2l/3 - 1/3$, we may apply the induction hypothesis to these formulas. Hence, f

can be represented by a formula whose depth is bounded by

$$c\log(2l/3 - 1/3 + 1) + 3 = c\log(2(l+1)/3) + 3 = \\ c\log(l+1) + c\log(2/3) + 3 = c\log(l+1).$$

The last equality follows from the definition of c.

Exercise 2.3. There are $2^{n-|S_i|}$ different assignments to the variables outside S_i. The number of functions on S_i is equal to $2^{2^{|S_i|}}$. Hence,

$$\log s_i \leq \min\{n- \mid S_i \mid, 2^{|S_i|}\}$$

and

$$u_i := (\log s_i)/\log\log s_i \leq \min\{(n- \mid S_i \mid)/\log(n- \mid S_i \mid), 2^{|S_i|}/ \mid S_i \mid\}.$$

Since the sets S_i are disjoint, there are at most $2n\log^{-1} n$ sets S_i whose size is larger than $\frac{1}{2}\log n$. For these sets we estimate u_i above by $n\log^{-1} n$ and the total contribution of these sets to Nechiporuk's lower bound (Theorem 2.2.4) is bounded above by $2n^2\log^{-2} n$.

For the other sets , $\mid S_i \mid \leq \frac{1}{2}\log n$ and we estimate u_i above by $2^{|S_i|}/ \mid S_i \mid$. The function $x \to 2^x/x$, $x \geq 2$, is convex. Hence we get the largest contribution if as many $\mid S_i \mid$ as possible are equal to $\frac{1}{2}\log n$. Hence, the total contribution is bounded $O(n^{3/2}\log^{-2} n)$.

Exercise 2.4. We apply the result of Exercise 2.6 that the BP size and, by Theorem 2.1.3, also the circuit size of DSA_n is bounded by $O(n)$. Hence the circuit size of $x_{|y|}$ is $O(n)$. The same argument proves that the circuit size of $x_{|y|+c}$ is $O(n)$. Altogether, we obtain a circuit of size $O(n\log n)$ computing $x_{|y|}, \ldots, x_{|y|+k-1}$. Using the outputs of this circuit as address variables and $x_0, \ldots, x_{n-1}$ as data variables we can compute $\mathrm{ISA}_n(x,y) = x_{\alpha(x,y)}$ with $O(n)$ further gates.

Exercise 2.5. See the proof of Theorem 6.1.3 for a solution of this exercise. The resulting BP is even an FBDD and a DT.

Exercise 2.6. See the proof of Theorem 4.3.2 for a solution of this exercise. The resulting BP is even an OBDD and a DT.

Exercise 2.7. Let S_i contain all variables $x_{j,j+i}$, $1 \leq j \leq n$, where the indices are taken mod n with representatives in $\{1, \ldots, n\}$. The $\mathbb{Z}_2$-determinant does not change by interchanging rows or columns. Hence, $s_1 = \cdots = s_n$ and it is sufficient to prove that $s_n \geq 2^{(n^2-n)/2}$. Afterwards, we can apply Theorem 2.2.4.

We replace the variables below the main diagonal by fixed constants and the variables above the main diagonal by all possible constants. We like to prove

that the functions

$$g_c(x_{11},\ldots,x_{nn}) = \det_n \begin{bmatrix} x_{11} & c_{12} & c_{13} & \cdots & c_{1,n-2} & c_{1,n-1} & c_{1n} \\ 1 & x_{22} & c_{23} & \cdots & c_{2,n-2} & c_{2,n-1} & c_{2n} \\ 0 & 1 & x_{33} & \cdots & c_{3,n-2} & c_{3,n-1} & c_{3n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & 1 & x_{n-1,n-1} & c_{n-1,n} \\ 0 & 0 & 0 & \cdots & 0 & 1 & x_{nn} \end{bmatrix}$$

are different for different settings of the c-constants. This is sufficient, since we have $(n^2-n)/2$ c-constants. The statement is obvious for $n=2$, since $g_c(x_{11},x_{22}) = x_{11}x_{22} \oplus c_{12}$. The case $n=3$ can be handled by considering the eight assignments to c_{12}, c_{13}, and c_{23}.

For $n>3$, we apply matrix operations which do not change the determinant. We multiply the second row by x_{11} and add the result to the first row. The new first row equals

$$(0, x_{11}x_{22} \oplus c_{12}, x_{11}c_{23} \oplus c_{13}, \cdots, x_{11}c_{2n} \oplus c_{1n}).$$

The first column contains only one non-zero entry, namely an entry 1, in the second row. Hence, we erase the first column and the second row. Then we set $x_{11}=1$. This results in an $(n-1)\times(n-1)$-matrix of the same type as described above. By induction hypothesis, $g_c \not\equiv g_{c'}$ if $c_{ij} \neq c'_{ij}$ for some $i \geq 3$ or $c_{1k} \oplus c_{2k} \neq c'_{1k} \oplus c'_{2k}$ for some $k \geq 3$.

We can apply similar arguments to the last two columns instead of the first two rows. This leads to the conclusion that $g_c \not\equiv g_{c'}$ if $c_{ij} \neq c'_{ij}$ for some $j \leq n-2$ or $c_{k,n-1} \oplus c_{kn} \neq c'_{k,n-1} \oplus c'_{kn}$ for some $k \leq n-2$.

We are left with the situation that $c_{ij} = c'_{ij}$ for all $(i,j) \notin \{(1,n-1),(1,n),(2,n-1),(2,n)\}$ and $c_{1k} \oplus c_{2k} = c'_{1k} \oplus c'_{2k}$ for $k \in \{n-1,n\}$. Now we set $x_{11}=0$. Then we can erase the first column and the second row of the matrix. The results for c and c' are $(n-1)\times(n-1)$-matrices M and M' which agree at all entries except perhaps the last two entries of the first row. Since $c \neq c'$ and $c_{1k} \oplus c_{2k} \neq c'_{1k} \oplus c'_{2k}$ for $k=n-1$ and $k=n$, we obtain three cases:
$c_{1,n-1} \neq c'_{1,n-1}$ and $c_{1n} = c'_{1n}$, $c_{1,n-1} = c'_{1,n-1}$ and $c_{1n} \neq c'_{1n}$, and $c_{1,n-1} \neq c'_{1,n-1}$ and $c_{1n} \neq c'_{1n}$. We compute the determinants of M and M' by expansion according to the first row. We obtain for the first $n-3$ entries the same values for M and M'. For the entry at position $n-2$ we obtain $x_{nn}c_{1,n-1}$ and $x_{nn}c'_{1,n-1}$ resp. For the entry at position $n-1$ (the last position) we obtain c_{1n} and c'_{1n} resp. Hence, $g_c(x) \oplus g_{c'}(x) = x_{nn}c_{1,n-1} \oplus x_{nn}c'_{1,n-1} \oplus c_{1n} \oplus c'_{1n}$ which is different from 0 in all three considered cases.

Exercise 2.8. We set $S_i = \{x_{i,i+1},\ldots,x_{in}\}$ for $i \in \{1,\ldots,n-1\}$. Let s_i be the number of subfunctions of 3-CLIQUE$_n$ on S_i. In order to obtain a lower bound on s_i, we replace all variables deciding about the existence of edges which are adjacent to one of the vertices $1,\ldots,i-1$ by 0. Hence, we consider a graph on the vertex set $V_i = \{i,\ldots,n\}$. For the variables x_{kl}, $i+1 \leq k < l \leq n$, we choose the $2^{\lceil (n-i)/2 \rceil \lfloor (n-i)/2 \rfloor}$ different assignments leading to a bipartite

graph on V_i where the first part contains the vertices $i+1,\ldots,i+\lceil(n-i)/2\rceil$. Bipartite graphs do not contain a 3-clique. We claim that these assignments lead to different subfunctions on S_i. Let a and a' be two such assignments and w.l.o.g. $a_{kl}=1$ and $a'_{kl}=0$. Let $x_{ik}=x_{il}=1$ and $x_{ij}=0$ for all other j. Together with a we obtain a 3-clique on $\{i,k,l\}$ and together with a' we do not obtain any 3-clique. Hence, $\log s_i \geq \lceil(n-i)/2\rceil\lfloor(n-i)/2\rfloor$. Moreover, $(\log s_i)/\log\log s_i \geq (\frac{1}{8}(n-i)^2-O(n))/\log n$. Since the sum of all $(n-i)^2$, $1\leq i\leq n-1$, equals $\frac{1}{3}n^3-O(n^2)$, Nechiporuk's lower bound (Theorem 2.2.4) leads to the lower bound $\frac{1}{24}n^3/\log n-O(n^2)$.

Exercise 2.9. The function 3-CLIQUE$_n$ is the disjunction of all $x_{ij}x_{ik}x_{jk}$, $1\leq i<j<k\leq n$. The number of inner nodes of a BP for $x_{ij}x_{ik}x_{jk}$ can be bounded by 3. Now we apply the result of Case 1 in the proof of Theorem 2.1.4. That result holds for the conjunction of functions but it can be obtained in a similar way for disjunctions. This leads to an upper bound on the BP size of $3\binom{n}{3}+2$ (the term 2 describes the 2 sinks).

Exercise 2.10. In order to represent MUL$_{k,n}$ it is sufficient to consider $(k+1)n$ z-levels (or more precisely $1+\cdots+(k+1)$ z-levels if $k\leq n-1$ and a similar formula can be derived for $k\geq n$) and we obtain the upper bound $4(k+1)n^2$.

Exercise 2.11. We start with the BP for MUL$_{2n-1,n}$ described in the proof of Theorem 2.3.5. It has size $O(n^3)$. After having tested the z-variables of column l, we have stored the bit p_l of the product $p=(p_{2n-1},\ldots,p_0)$ and the carry to the next column. In this situation we test the lth bit of the third number of the input for multgraph$_n$. If it equals p_l, we go on. Otherwise, we reach the 0-sink.

Exercise 2.12. There are s paths $p_1,\ldots,p_s$ from the root of the DT to a 1-leaf. Each path p_i corresponds to a monomial m_i which equals 1 on input a iff the input a activates the corresponding path. Obviously, $f=m_1+\cdots+m_s$. Moreover, $m_im_j=0$ if $i\neq j$. This follows, since p_i and p_j split at some node v. Let x_k be the label of v. Then m_i contains x_k while m_j contains $\overline{x}_k$ or vice versa.

Exercise 2.13. Let $f=m_1+\cdots+m_s$, where $m_1,\ldots,m_s$ are the minterms of f. We test the variables in the ordering $x_1,\ldots,x_n$. Whenever the corresponding subfunction is equal to a constant, we stop the computation with the corresponding leaf. We get at most s paths leading to a 1-leaf. A path can lead to at most n 0-leaves. Altogether, the number of leaves is bounded by $(s+1)n$. The statement does not hold for $s=0$ (we need 1 leaf) and for $s=1$ (we need $n+1$ leaves for a minterm). For $s\geq 2$, we either have only one path to a 1-leaf and $n+1\leq sn$ or each path to a 1-leaf contains a node followed by two inner nodes and contributes at most 1 1-leaf and $n-1$ 0-leaves to DT(f).

Chapter 3

Exercise 3.1. Let s_i be the number of different subfunctions $f_{j|x_1=a_1,\dots,x_{i-1}=a_{i-1}}$, $1 \leq j \leq m$, $a_1, \dots, a_{i-1} \in \{0,1\}$, essentially depending on x_i. We construct a π-OBDD representing $f = (f_1, \dots, f_m)$ with s_i x_i-nodes, $1 \leq i \leq n$, and s_{n+1} sinks. For each f_j, $1 \leq j \leq m$, we use the same approach as described before Theorem 3.1.4 for single-output functions. If subfunctions for f_j and $f_{j'}$ are equal, they are represented by the same node. On the other hand, the proof of Theorem 3.1.4 shows that each considered subfunction has to be represented at some node. Since each node can represent only one function, the statement is proved. There is no choice how to direct the edges.

Exercise 3.2. The algorithm is described essentially already before Theorem 3.1.5. For an efficient implementation, it is possible to work with a DFS traversal starting at the pointers for the output functions. Nodes not reachable from these pointers can be eliminated. We use an extra array for all nodes. Whenever we traverse an edge (v, w), we distinguish whether w is reached for the first time or has been reached before. For the first case, we check whether the 0-edge leaving w is complemented. Whenever this is the case, we remove the compl-bit at the 0-edge leaving w, negate the compl-bit at the 1-edge leaving w and the compl-bit on the edge (v, w). In the second case, we look up the information whether the 0-edge leaving w has been complemented in the beginning. Whenever this is the case, we negate the compl-bit on the edge (v, w). The correctness of this algorithm is obvious. The algorithm needs time $O(1)$ for each edge.

Exercise 3.3. We consider the parity function $x_1 \oplus \cdots \oplus x_n$ and the variable ordering $x_1, \dots, x_n$. (We obtain the same result for each variable ordering.) The reduced π-OBDD contains one x_1-node, two x_i-nodes, $2 \leq i \leq n$, representing $x_i \oplus \cdots \oplus x_n$ and $x_i \oplus \cdots \oplus x_n \oplus 1$ resp., and two sinks. The size equals $2n+1$. Using complemented edges, the nodes representing $x_i \oplus \cdots \oplus x_n$ and $x_i \oplus \cdots \oplus x_n \oplus 1$ can be merged, since one function is the negation of the other one. The same holds for the sinks. Hence, the size is reduced to $n+1$ and $2n+1 = 2(n+1)-1$.

Exercise 3.4. A function f is called monotone if $a \leq b$ implies $f(a) \leq f(b)$. Here we define that $0 \leq 1$ and $a \leq b$ iff $a_i \leq b_i$ for all i. It is easy to see that subfunctions of monotone functions are monotone again. If a monotone function is not constant, by definition, $f(0, \dots, 0) = 0$ and $f(1, \dots, 1) = 1$. In particular, the negation of such a function is not monotone. Hence, for a non-constant monotone function the use of complemented edges saves exactly one node, namely the 1-sink.

Exercise 3.5. W.l.o.g. $\pi = id$. We use the main ideas of the proof of Theorem 3.1.4. For $g := f_{|x_1=a_1,\dots,x_i=a_i}$ we have used $\alpha(g)$ to describe the smallest index k such that f essentially depends on x_k. Here we define $\alpha(g) := x_{i+1}$, if $i < n$, and $\alpha(g) = \text{const}$, if $i = n$. Then we use the same construction as before Theorem 3.1.4 to obtain a complete π-OBDD representing f with as many

x_i-nodes as there are different subfunctions $f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$. Moreover, each subfunction has to be represented and, in a complete π-OBDD, it has to be represented by an x_i-node or by a sink, if $i = n+1$. If a complete π-OBDD for f contains exactly nodes for the subfunctions as described above, the connections by the edges are uniquely determined.

Exercise 3.6. W.l.o.g. $\pi = id$. We construct a complete π-OBDD for f from the reduced π-OBDD G for f. Let v be a node of G. If v is labeled by x_i, we create new nodes $v_1, \ldots, v_{i-1}$. If v is a sink, we create new nodes $v_1, \ldots, v_n$. Hence, we create for each node at most n new nodes and the number of old and new nodes is at most by a factor of $n+1$ larger than the number of old nodes. Now we construct a complete π-OBDD for f based on the nodes of G and the nodes newly created. Let v be labeled by x_i (where we "label" a sink by a dummy variable x_{n+1}). Then the edges leaving the new node v_j, $j < i-1$, lead to the new node v_{j+1} and the edges leaving the new node v_{i-1} lead to the old node v. Hence, the nodes v_j represent the same function as v. An old edge from a node w labeled by x_j to a node v labeled by x_i is redirected to v_{j+1} if $i - j \geq 2$. Here, we again assume sinks to be labeled by x_{n+1}. Moreover, we assume that pointers for the functions to be represented are edges starting at a node labeled by x_0. The new OBDD respects the variable ordering $\pi = id$, is complete, represents f, and its size is bounded above by $(n+1)|G|$.

Exercise 3.7. The constant function 1 has the OBDD size 1. A complete OBDD obviously needs $n+1$ nodes. It follows from the solution of Exercise 3.6 that the factor $n+1$ is only necessary for constant functions.

Exercise 3.9. The synthesis algorithm for π-OBDDs with complemented edges is based on the synthesis algorithm for π-OBDDs without complemented edges. A negation can be performed by negating the compl-bit on the pointer for the considered function. In general, $f = g \otimes h$ and we perform a simultaneous DFS traversal through the OBDDs representing g and h resp. Instead of node pairs (v, w) we consider $((v, a), (w, b))$ where a and b indicate whether we have reached v resp. w via a complemented edge. We may define terminal cases, among them at least the cases where v and w are sinks. In the same way as in the usual synthesis algorithm we omit tests and "wait" in one of the OBBDs. Otherwise, if $a = 0$, we look for the successors v_0 and v_1 and take into account the compl-bit on the edges (v, v_0) and (v, v_1). If $a = 1$, we negate the compl-bits on the edges. If a backtracking step leads to a 0-edge with a compl-bit, we integrate the solution to Exercise 3.2 into this synthesis algorithm. It is obvious how to integrate the reduction into the synthesis process.

Exercise 3.10. We consider m functions $f_1, \ldots, f_m$ where f_i is defined as multiplexer on its private address vector $x^i = (x_0^i, \ldots, x_{k-1}^i)$ and the common data vector $z = (z_0, \ldots, z_{n-1})$. We use a variable ordering π respecting the ordering $x^1, \ldots, x^m, z$ of the vectors. The π-OBDD size of each f_i is equal to $2n+1$. Let $f = f_1 + \cdots + f_m$. We consider the subfunctions with respect to the different assignments to the variables in $x^1, \ldots, x^m$. We obtain $n + \binom{n}{2} + \cdots + \binom{n}{m}$

different subfunctions namely all functions $x_{i_1} + \cdots + x_{i_r}$ where $1 \leq r \leq k$ and $1 \leq i_1 < \cdots < i_r \leq n$. This number is a lower bound on the OBDD size of f.

Exercise 3.11. Let the variable ordering π be given by $x_0, \ldots, x_{k-1}$, $y_0, \ldots, y_{k-1}, z_0, \ldots, z_{n-1}, s$ where $n = 2^k$. Let f_n be the conjunction of s and the multiplexer on x and z and let g_n be the conjunction of $\overline{s}$ and the multiplexer on y and z. Obviously, $f_n \wedge g_n = 0$. The π-OBDD size of each of the functions f_n and g_n equals $2n + 2$. Now we consider $f_n + g_n$ and their subfunctions with respect to the different assignments to the x- and y-variables. We obtain the $n(n-1)$ different subfunctions $sx_i + \overline{s}x_j, i \neq j$, and, moreover the n different subfunctions $sx_i + \overline{s}x_i = x_i, 1 \leq i \leq n$.

Exercise 3.12. If $f = [(g \oplus a) \wedge (h \oplus b)] \oplus c$, $f = ite(g, h_1, h_2)$ where one of the functions h_1 and h_2 is a constant and the other one is h or $\overline{h}$. Hence, the simultaneous DFS traversal runs through the OBDD representing g, the OBDD representing h where perhaps the labeling of the sinks has to be negated, and an OBDD representing a constant. This leads to the same node pairs as in the binary synthesis (if we ignore the component representing the constant). All terminal cases considered in the binary synthesis can be adapted to the ternary case. If $f = g \oplus h \oplus a$, $f = ite(g, h \oplus a, h \oplus \overline{a})$. Hence, we run through the OBDD representing g, through the OBDD representing h, and virtually through an OBDD representing $\overline{h}$. Whenever we reach the node v in the OBDD for h, we may interpret this as reaching $\overline{v}$ in an OBDD for $\overline{h}$. The 0-sink in the OBDD for h corresponds to the 1-sink in the OBDD for $\overline{h}$ and vice versa. Again the terminal cases can be adapted.

Exercise 3.13. It is obvious that $f \leq g$ iff $f \wedge \overline{g} \equiv 0$. Hence, the property $f \leq g$ can be checked by the application of the synthesis algorithm to construct a π-OBDD for $f \wedge \overline{g}$ and to check whether we construct a π-OBDD representing the constant 0.

Exercise 3.14. We have one processor for each node. The processor corresponding to the inner node v determines the label x_i of v and the value a_i of x_i. Then it computes the a_i-successor of v and writes it into a node array at position v. The processor corresponding to the sink v writes v into the position v of the node array. In both cases, the processor remembers the node written in the last round. Then we repeat for $\lceil \log n \rceil$ times the following procedure for all processors. The processor for node v looks at the array position v' where v' is the node written in the last round. If the processor for node v reads v^*, the processor writes v^* into position v and remembers v^*. It is obvious that this algorithm is an exclusive write (EW) one. Moreover, after i steps, the processor at the root knows the node in distance 2^i on the path activated by the given input. If a sink is reached in less then 2^i steps, the processor knows this sink. The length of the activated path is bounded by n. Hence, after $\lceil \log n \rceil$ steps the processor at the root knows the solution of the evaluation problem.

Exercise 3.15. The satisfiability problem is trivial for reduced OBDDs. In the general case, we have to determine whether there is a directed path from

the source to the 1-sink. We are faced with the well-known transitive closure bottleneck for PRAMs. We know that the length of paths in an OBDD is bounded above by n. It is easy to construct the adjacency matrix A of the graph which describes the OBDD. We may forget the labeling of the nodes and edges. The Boolean matrix product D of two Boolean $m \times m$-matrices B and C is defined by

$$d_{ij} = b_{i1}c_{1j} + b_{i2}c_{2j} + \cdots + b_{im}c_{mj}.$$

Each d_{ij} can be computed by m processors, one responsible for $b_{ik}c_{kj}$, in constant time. First, the processor computes the product $b_{ik}c_{kj}$. Then the processor writes (common writing) 1 into an array for the result iff $b_{ik}c_{kj} = 1$. This array is initialized with 0-entries. Hence, the disjunction can be computed within one step. The different d-entries can be computed independently (since common reading is allowed). Hence, m^3 processors can realize the Boolean matrix product in constant time. We like to compute the n-th power of A assuming that there is an edge from each sink leading back to the sink. Hence, in this case $m = |G_f|$ and we have to perform $\lceil \log n \rceil$ matrix multiplications to compute sequentially $A^2, A^4, A^8, \ldots, A^{2^{\lceil \log n \rceil}}$.

Exercise 3.16. We work with $|G_f| \cdot |G_g|$ processors and, therefore, we have one processor for each node pair $(v, w) \in V_f \times V_g$. We compute the OBDD $G_h = G_f \times G_g$ (see the definition below Theorem 3.3.4) as the result of the binary synthesis. The processor responsible for (v, w) has to compute the label of this node and the 0-successor and the 1-successor of this node. The processor reads $x_j := \text{label}(v)$ and $x_k := \text{label}(w)$ (the label of a sink is interpreted as x_{n+1}) and the label of (v, w) is defined as x_i where $i = \min\{j, k\}$. If $i = n+1$, the node (v, w) is a sink whose label equals $\text{label}(v) \otimes \text{label}(w)$ ($\otimes$ is the operator of the binary synthesis). If $i \leq n$, the processor computes the successors v_0, v_1, w_0, and w_1 of v and w resp. The processor defines the 0-successor of (v, w) as (v_0^*, w_0^*) where $v_0^* = v_0$, if v is labeled by x_i, and $v_0^* = v$ otherwise. The node w_0^* and the 1-successor (v_1^*, w_1^*) are defined similarly.

Exercise 3.17. Let $s = |G_f|$. We apply the solution of Exercise 3.15 to eliminate all nodes not reachable from the source. Let $G = (V, E)$ be this OBDD containing only nodes reachable from the source. The next aim is to determine for each each node pair (v, w) whether $f_v = f_w$. We apply the solution of Exercise 3.16 to construct an OBDD G^* for $f \oplus f$. Remember that this OBDD contains the node pair (v, w) representing $f_v \oplus f_w$. We again apply the solution of Exercise 3.15 to the OBDD G^*. Since the size of G^* is bounded by s^2, we know that s^6 processors and time $O(\log s)$ are sufficient for this purpose. Afterwards, we can determine for each (v, w) whether $f_v \oplus f_w = 0$, i.e., whether $f_v = f_w$. The last step is to construct the reduced OBDD. For each node v of G we have an array describing which nodes represent the same function as v. Using s processors for each node v the node $l(v)$ with the largest number among the nodes equivalent to v can be determined in time $O(\log s)$. Here we assume a topological ordering of the nodes (which otherwise can be easily computed). The reduced OBDD G^{red} for f contains all the nodes v where $v = l(v)$. These

nodes have the same labels as in G. The 0-successor of v in G^{red} is $l(v_0)$ if v_0 is the 0-successor of v in G, similarly for the 1-successor. The whole algorithm runs in time $O(\log s)$ using s^6 processors.

Exercise 3.18. W.l.o.g. $\pi = id$. Let G^* be the reduced $\{0,1,*\}$-representation of f. We consider f as completely specified function $f : \{0,1\}^n \to \{0,1,*\}$. Let v be a node of G^* labeled by x_i. Then f_v is a subfunction for some assignment $(a_1, \ldots, a_{i-1})$ to the first $i-1$ variables. G^* has at most three sinks with labels from $\{0,1,*\}$. Let G_{on} and G_{dc} be reduced π-OBDDs representing f_{on} and f_{dc} resp. and let G' be the product graph after the elimination of not reachable nodes. The product graph has sinks labeled by $(0,0),(0,1),(1,0)$ and $(1,1)$. Since it is impossible that $f_{on}(a) = f_{dc}(a) = 1$, the sink $(1,1)$ is not reachable. The sink $(0,0)$ corresponds to the 0-sink of G^*, $(1,0)$ corresponds to the 1-sink, and $(0,1)$ corresponds to the $*$-sink. This identification of the images leads to a correspondence of f and (f_{on}, f_{dc}). The partial assignments a and b lead to different subfunctions of f_{on} or f_{dc} iff they lead to different subfunctions of f. This implies that the OBDD obtained from G' by reduction is isomorphic to G^*. The last step is to prove that G' is reduced. If the partial assignments a and b lead to the same subfunctions of (f_{on}, f_{dc}), they do so for f_{on} and f_{dc} and we reach for a and b the same node in G_{on} and the same node in G_{dc}. This implies that we reach for a and b the same node in G' and G' is reduced.

Exercise 3.19. The solution of this exercise has been obtained by Benedikt Stockebrand in his Master Thesis.

We start with general considerations. Let $m = 2^k$, $z = (z_{k-1}, \ldots, z_0)$, and $x = (x_1, \ldots, x_n)$. We fix the variable ordering $z_{k-1}, \ldots, z_0, x_1, \ldots, x_n$. We define $e_{i,r}$, $0 \le i \le 2^r - 1$, on r variables by $e_{i,r}(y_{r-1}, \ldots, y_0) = 1$ iff $|y| = i$. Let $(f_0, c_0), \ldots, (f_{m-1}, c_{m-1})$ be a sequence of incompletely specified Boolean functions on x such that $c_i c_j = 0$ and $f_i \neq f_j$, if $i \neq j$. Let (f^*, c^*) be defined by

$$f^*(x,z) = \bigvee_{0 \le i \le m-1} e_{i,k}(z) f_i(x)$$

and

$$c^*(x,z) = \bigvee_{0 \le i \le m-1} e_{i,k}(z) c_i(x).$$

We apply the tsm heuristic levelwise top-down.
Claim: After k levels we consider (F, C) defined by

$$F(x,z) = \bigvee_{0 \le i \le m-1} f_i(x) c_i(x)$$

and

$$C(x,z) = \bigvee_{0 \le i \le m-1} c_i(x).$$

First we prove the claim. It is obvious that

$$c^*_{|z_{k-1}=0}(x,z) = \bigvee_{0 \le i \le m/2-1} e_{i,k-1}(z_{k-2}, \ldots, z_0) c_i(x)$$

and

$$c^*_{|z_{k-1}=1}(x,z) = \bigvee_{0\le i\le m/2-1} e_{i,k-1}(z_{k-2},\ldots,z_0)c_{i+m/2}(x)$$

are disjoint. Hence, by Lemma 3.6.3, the new care function is $c^{**} = c^*_{|z_{k-1}=0} + c^*_{|z_{k-1}=1}$ and the new function is $f^{**}(x,z) = f^*_{|z_{k-1}=0}c^*_{|z_{k-1}=0} + f^*_{|z_{k-1}=1}c^*_{|z_{k-1}=1}$. Let f'_i and c'_i, $0 \le i \le m/2-1$, be defined by

$$f'_i(x) = f_i(x)c_i(x) + f_{i+m/2}(x)c_{i+m/2}(x)$$

and

$$c'_i(x) = c_i(x) + c_{i+m/2}(x).$$

Then we can conclude that

$$f^{**}(x,z) = \bigvee_{0\le i\le m/2-1} e_{i,k-1}(z_{k-2},\ldots,z_0)f'_i(x)$$

and

$$c^{**}(x,z) = \bigvee_{0\le i\le m/2-1} e_{i,k-1}(z_{k-2},\ldots,z_0)c'_i(x).$$

Moreover, $c'_ic'_j = 0$ and $f'_i \ne f'_j$, if $i \ne j$. Hence, we are in the same situation as in the beginning and can argue on all the levels in a similar way. It follows that we finally obtain (F,C). (End of the proof of the claim.)

Now we define $f_0, c_0, \ldots, f_{m-1}, c_{m-1}$ on $(x,s) = (x_{m-1},\ldots,x_0,s_{k-1},\ldots,s_0)$ using this sequence as variable ordering. Let

$$f_i(x,s) = x_i$$

and

$$c_i(x,s) = e_{i,k}(s).$$

The conditions $c_ic_j = 0$ and $f_i \ne f_j$, if $i \ne j$, are fulfilled. The OBDD for f^* starts with a complete binary tree on the z-variables (size $O(m)$) followed by OBDDs for f_i of size 1 each. The whole OBDD size of f^* is $O(m)$. The OBDD size of each e_i is $O(k) = O(\log m)$ and, therefore, the OBDD size of c^* is $O(m \log m)$.

Now we investigate the functions F and C obtained after k applications of the tsm heuristic. The conditions of the claim are fulfilled. The function C is the constant 1. The function F is the multiplexer with a bad variable ordering leading to an OBDD size of $\Omega(2^m)$, an exponential blow-up. Since $C = 1$, the function is completely specified and the further application of the tsm heuristic does not change anything.

Exercise 3.20. We apply the solution of Exercise 3.18. The care function c is the negation of the don't care function. Hence, the statement of Exercise 3.18 holds also for c instead of f_{dc}. Instead of a representation of (f,c) we work with the $\{0,1,*\}$-representation of the incompletely specified function whose OBDD

size is bounded by the product of the OBDD size for f and the OBDD size for c and, therefore, polynomially bounded in the input size. The osm criterion uses (F_2, c_2) as common extension of (F_1, c_1) and (F_2, c_2) if the conditions of the criterion are fulfilled. This implies that no new OBDD nodes are created, only some pointers are redirected which may lead to the application of reduction rules.

Exercise 3.21. Let G be a reduced π-OBDD representing f. For each $i \in \{1, \ldots, n\}$, we construct in time $O(|G|)$ π-OBDDs for $f_{|x_i=0}$ and $f_{|x_i=1}$. Afterwards, we apply the solution of Exercise 3.13 to check whether $f_{|x_i=0} \leq f_{|x_i=1}$. We claim that f is monotone iff $f_{|x_i=0} \leq f_{|x_i=1}$ for all i. The whole algorithm runs in time $O(n|G|^2)$. It is obvious that $f_{|x_i=0} \leq f_{|x_i=1}$ if f is monotone. Now we assume that $f_{|x_i=0} \leq f_{|x_i=1}$ for all i. We have to prove that $f(a_1, \ldots, a_n) \leq f(b_1, \ldots, b_n)$ if $a_i \leq b_i$ for all i. Let i_1 be the smallest index where $a_{i_1} = 0$ and $b_{i_1} = 1$. Then $f(a_1, \ldots, a_n) \leq f(a_1, \ldots, a_{i_1-1}, 1, a_{i_1+1}, \ldots, a_n)$, since $f_{|x_{i_1}=0} \leq f_{|x_{i_1}=1}$. We go on in the same way until we have switched from a to b.

Chapter 4

Exercise 4.1. We consider the carry bit s_{n+1} of $\mathrm{ADD}_{n+1}(\overline{x}_{n-1}, \ldots, \overline{x}_0, 1, y_{n-1}, \ldots, y_0, 1)$. If $|x| < |y|$, there is some index i such that $x_i = 0$, $y_i = 1$, and $x_j = y_j$ for $j > i$. This implies that $\overline{x}_i = y_i = 1$ and exactly one of $\overline{x}_j$ and y_j is equal to 1 for $j > i$. Obviously, this leads to a carry. If $|x| = |y|$, we use the same arguments. Here the carry is generated from the least significant 1-entries. If $|x| > |y|$, there is some index i such that $x_i = 1$, $y_i = 0$, and $x_j = y_j$ for $j > i$. This implies that $\overline{x}_i = y_i = 0$ and exactly one of $\overline{x}_j$ and y_j is equal to 1 for $j > i$. Obviously, this prevents a carry.

Exercise 4.2. We like to compute the sum of the n-bit numbers $x^1, \ldots, x^n$ using MUL_{2n^2}. The first factor p_1 is the concatenation of $0^n, x^1, 0^n, x^2, \ldots, 0^n, x^n$ where 0^n consists of n zeros ($\lceil \log n \rceil$ zeros would be sufficient). The second factor p_2 repeats the string $0^{2n-1}1$ for n times. Using the school method for multiplication we have to compute the sum of p_1, p_1 shifted by $2n$ positions, p_1 shifted by $4n$ positions, ... , p_1 shifted by $(n-1)2n$ positions. If we write these numbers into a table like in Figure 2.3.1a, we find $3n$ positions containing $0^n x^1 0^n, 0^n x^2 0^n, \ldots, 0^n x^n 0^n$. The zeros at the end ensure that no carry from less significant positions has influence on the positions where we compute the sum of $x^1, \ldots, x^n$. The zeros at the beginning ensure that we find the carry of this addition in the result.

Exercise 4.3. We use a similar idea as for the solution of Exercise 4.2. Here $p_1 = 0^n x_1 0^n x_2 0^n \ldots 0^n x_n$ and $p_2 = 0^n y_n 0^n y_{n-1} 0^n \ldots 0^n y_1$. Then we find in the table after the bitwise multiplication a column containing (besides zeros) the entries $x_1y_1, x_2y_2, \ldots, x_ny_n$. We compute the sum of this column without a carry from less significant positions. The last bit of this sum is equal to $x_1y_1 \oplus \cdots \oplus x_ny_n$.

Exercise 4.4. The branching program value problem BPVP is defined on inputs (G, a) such that $a \in \{0,1\}^n$ for some n and G is a BP with n^2 inner nodes and 2 sinks such that the inner node with number $jn + i$, $0 \leq j \leq n-1$, $1 \leq i \leq n$, is labeled by x_i. The output is the result of the evaluation problem on the BP G (the source is the node with number 1) and the input a.

Let $f = (f_n) \in$ P-BP. Then f_n can be represented by a BP G_n of polynomial size $p(n)$. W.l.o.g. $p(n) \geq n$. It is possible to embed G_n into a BP G'_n as described in the definition of BPVP. If we work with $p(n)$ variables, we only use the first n variables and consider the further variables as dummy variables. The jth node v of G_n is simulated by the node $(j-1)p(n) + i$ if the label of v is equal to x_i. In order to compute $f_n(a)$, we can apply BPVP to the input consisting of G'_n and $a' = (a_1, \ldots, a_n, 0, \ldots, 0)$. Hence, $f \leq_{\mathrm{rop}}$ BPVP.

We still have to prove that BPVP $\in$ P-BP. We may repeat the test of variables. We check the variables in the following ordering: x_1, encoding of node 1, x_2, encoding of node 2, ... , x_n, encoding of node n, x_1, encoding of node $n+1$, ... , x_n, encoding of node $2n$, ... , x_1, encoding of node $(n-1)n+1$, ... , x_n, encoding of node n^2. For each pair (x_i, encoding of node $(j-1)n+i$)

we use the following gadget. We compute the binary number of the x_i-successor of this node. Each leaf of this gadget is replaced with the source of the gadget of the corresponding successor node, if the successor is an inner node, or by the corresponding sink. The source of the gadget for (x_1, encoding of node 1) is the source of the whole BP. It is obvious that we represent BPVP in polynomial size.

Exercise 4.5. Obviously, the number of sinks is reduced from 2 to 1. Using the remarks following the proof of Lemma 4.4.2 we conclude that it is sufficient to consider the OBDD for a single sum bit s_i. Let $i \leq n-1$. There is only one y_j-node, $0 \leq j \leq i$, and no saving is possible on the y-levels. On the x_j-level, $0 \leq j < i$, we know c_{j-1}. If $j < i$ and $x_j = y_j = \cdots = x_i = y_i = 0$, then $s_i = 0$ independently from c_{j-1}. Hence, the subfunction for $c_{j-1} = 0$ is not the negation of the subfunction for $c_{j-1} = 1$. If $j = i$, we have to represent $x_i \oplus y_i$ and its negation $x_i \oplus y_i \oplus 1$. This leads to a further saving of one node. Altogether, the reduced π-OBDD with complemented edges for s_i has two inner nodes less than the reduced π-OBDD without complemented edges. We omit similar arguments for s_n.

Exercise 4.6. The proof of Theorem 4.4.3 describes which functions are represented in the reduced π-OBDD. We have two x_{n-1}-nodes representing s_n and s_{n-1} and, obviously, $s_n \neq \overline{s}_{n-1}$. We have three x_i-nodes, $0 \leq i \leq n-2$, representing s_i, c_i, and $\overline{c}_i$. Hence, we save one node on each of these levels, altogether $n-1$ nodes. On the y_{n-1}-level, the functions $c_{n-2}y_{n-1}$, $c_{n-2}+y_{n-1}$, $c_{n-2} \oplus y_{n-1}$, and $c_{n-2} \oplus \overline{y}_{n-1}$ are represented and one node can be saved. On the y_i-level, $1 \leq i \leq n-2$, we may save three of the six nodes representing $c_{i-1} \oplus y_i$, $c_{i-1} \oplus \overline{y}_i$, $c_{i-1}y_i$, $c_{i-1} + y_i$, $\overline{c_{i-1}y_i}$, and $\overline{c_{i-1} + y_i}$. On the y_0-level we may save one of the two nodes representing y_0 and $\overline{y}_0$. Moreover, we save one of the two sinks. Altogether, we save $n-1+1+3(n-2)+1+1 = 4n-4$ nodes and we can represent ADD_n, $n \geq 2$, with $5n-1$ nodes.

Exercise 4.7. We consider the inputs $x = (\mathrm{sign}(x), x_{n-1}, \ldots, x_0)$ and $y = (\mathrm{sign}(y), y_{n-1}, \ldots, y_0)$ and want to represent the sum $s = (\mathrm{sign}(s), s_n, s_{n-1}, \ldots, s_0)$. A sign bit 1 indicates that the number is non-negative while a sign bit 0 indicates that the number is non-positive. We choose the variable ordering

$$\mathrm{sign}(x), \mathrm{sign}(y), x_{n-1}, y_{n-1}, x_{n-2}, y_{n-2}, \ldots, x_0, y_0.$$

This sign(x)-level contains $n+2$ nodes, since the $n+2$ outputs are different functions essentially depending on sign(x). The sign(y)-level contains $2n+4$ nodes, since all considered subfunctions are different and depend essentially on sign(y). First, we investigate the output sign(s). If sign(x) = sign(y), we reach the sink with label sign(x). If sign(x) = 1 and sign(y) = 0, we define sign(s) = 1 iff $|x| \geq |y|$ (sign(s) is only incompletely specified, since the number 0 has two representations). If sign(x) = 0 and sign(y) = 1, we define sign(s) = 1 iff $|x| < |y|$. Hence, we have to represent the predicate $|x| \geq |y|$ and its negation (which is useful if we work with complemented edges). In order to represent the predicate $|x| \geq |y|$, we need the following number of nodes:

- one x_i-node for the situation $x_{n-1} = y_{n-1}, \ldots, x_{i+1} = y_{i+1}$, (in all other cases we have reached a sink), this node represents the predicate $|(x_i, \ldots, x_0)| \geq |(y_i, \ldots, y_0)|$,

- two y_i-nodes for the situations $x_{n-1} = y_{n-1}, \ldots, x_{i+1} = y_{i+1}$ and $x_i = 0$ resp. $x_i = 1$, if $i \geq 1$, and one y_0-node (since for $x_0 = 1$ we know the result).

We have $3n + 6$ nodes on the sign-levels and further $6n - 2$ nodes on the x- and y-levels for the representation of sign(s).

If sign(x) = sign(y), the outputs $s_n, \ldots, s_0$ have the same values as for ADD$_n$. W.l.o.g. we assume $n \geq 2$. Then the proof of Theorem 4.4.3 shows that we need the following nodes

- 2 x_{n-1}-nodes representing s_n and s_{n-1},

- 3 x_i-nodes, $0 \leq i \leq n-2$, representing s_i, c_i, and $\overline{c}_i$,

- 4 y_{n-1}-nodes representing $c_{n-2}y_{n-1}, c_{n-2} + y_{n-1}, c_{n-2} \oplus y_{n-1}$, and $c_{n-2} \oplus \overline{y}_{n-1}$,

- 6 y_i-nodes, $1 \leq i \leq n-2$, representing $c_{i-1} \oplus y_i, c_{i-1} \oplus \overline{y}_i, c_{i-1}y_i$, $c_{i-1} + y_i, \overline{c_{i-1}y_i}, \overline{c_{i-1} + y_i}$, and

- 2 y_0-nodes representing y_0 and $\overline{y}_0$.

For $i \geq 1$, these nodes cannot be merged with nodes for the representation of the predicate $|x| \geq |y|$, the reason is that the above functions are symmetric with respect to x_{i-1} and y_{i-1} (their roles can be interchanged without changing the output) which is not the case for the predicate $|x| \geq |y|$.

If sign$(x) \neq$ sign(y), we have to represent $(s_n^*, s_{n-1}^*, \ldots, s_0^*) = \big||x| - |y|\big|$. In particular, $s_n^* = 0$. Using the school method for subtraction, we need another type of carry. This carry c_i^* equals 1 iff $|(x_i, \ldots, x_0)| < |(y_i, \ldots, y_0)|$. We can conclude that

$$s_i^* = x_i \oplus y_i \oplus c_{i-1}^*$$

and

$$c_i^* = \overline{x}_i(y_i + c_{i-1}^*) + x_i y_i c_{i-1}^*.$$

Hence, we need

- 1 x_{n-1}-node representing s_{n-1}^*,

- 3 x_i-nodes, $0 \leq i \leq n-2$, representing s_i^*, c_i^*, and $\overline{c}_i^*$,

- 2 y_{n-1}-nodes, representing $y_{n-1} \oplus c_{n-2}^*$ and $\overline{y}_{n-1} \oplus c_{n-2}^*$,

- 6 y_i-nodes, $1 \leq i \leq n-2$, representing $y_i \oplus c^*_{i-1}$, $\overline{y}_i \oplus c^*_{i-1}$, $y_i + c^*_{i-1}$, $\overline{y_i + c^*_{i-1}}$, $y_i c^*_{i-1}$, and $\overline{y_i c^*_{i-1}}$, and

- 2 y_0-nodes, representing y_0 and $\overline{y}_0$.

The x-nodes for the representation of sign(s) represent $c^*_0, \ldots, c^*_{n-2}$, $\overline{c}^*_0, \ldots,$ $\overline{c}^*_{n-2}$, and $\overline{c}^*_{n-1}$. Hence, we may merge all nodes except the 3 nodes on the x_{n-1} and y_{n-1}-level with the corresponding nodes for the case sign(x) $\neq$ sign(y). All other functions are different as long as $i \geq 1$. Alltogether, we can estimate the OBDD size up to an additive constant by $3n$ (sign levels), $6n$ (x-levels), and $12n$ (y-levels). Hence, the OBDD size equals $21n \pm O(1)$.

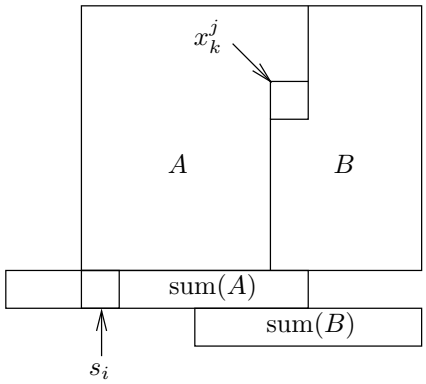
Exercise 4.8. It is sufficient to store the partial sum of the values of the considered bits, e.g., the bit x^i_j has the value $x^i_j 2^j$. The output has length $n + \lceil \log n \rceil$. Hence, the number of different partial sums is bounded by $O(n2^n)$. For the representation of s_i it is sufficient to store the last $i+1$ bits of the partial sum. Hence, the total size of one level is bounded above by

$$2^1 + 2^2 + \cdots + 2^{n\lceil \log n \rceil} = O(n2^n).$$

leading to an upper bound of $O(n^3 2^n)$.

For a lower bound we consider the output s_{n-1}. After having tested the first row, we may have 2^n different partial sums. Before we start to test the last row, we have to distinguish all partial sums. This follows from the consideration of ADD_n and the bad variable ordering $x_{n-1}, \ldots, x_0, y_{n-1}, \ldots, y_0$ in Lemma 4.5.1. Hence,we have at least $n^2 - 2n$ levels of size 2^n each leading to a lower bound of $\Omega(n^2 2^n)$.

Exercise 4.9. The sum bit s_i essentially depends on all the input bits x^j_k, $1 \leq j \leq n$, $0 \leq k \leq i$. We describe the situation after the test of x^j_k in the following figure. We have tested the variables of A. In order to represent s_i



only the bits at the positions $k, \ldots, i$ of the current partial sum are of interest. The value of s_i is influenced by the variables of B only via its partial sum called sum(B). If we compute the sum of sum(A) and sum(B) and if we are interested

in s_i, we may forget the bits of sum(B) at all positions $p < k$. Let p^* be the highest position where sum(B) may have the value 1. Then $p^* - k \leq \lceil \log n \rceil$. Moreover, if sum$(A)$ has a 0-entry at one of the positions $p^* + 1, \ldots, i - 1$, sum(B) cannot have influence on the final value of s_i. In these cases we have reached a sink. We investigate the other cases. Then it is sufficient to store the value of s_i and the bits of sum(A) at the positions $k, \ldots, p^*$. This leads to an upper bound of $O(n)$ on the size of each x-level for each output and that implies an upper bound of $O(n^3)$ for each output bit. For most output bits and most levels we also obtain an $\Omega(n)$ lower bound leading to an $\Omega(n^3)$ lower bound for most outputs. If the width of B is at least 2, there at least $\lceil \log n \rceil - 2$ carry positions which can take each value. If the width of A is at least $\lceil \log n \rceil$ we have to distinguish $\Omega(n)$ situations.

The crucial observation is that we can merge many nodes in an SBDD for all outputs. Let us consider s_i and $s_{i'}$ and the situation after the test of x_j^k. Let a and a' be assignments of the tested variables. We consider the subfunction of s_i with respect to a and the subfunction of $s_{i'}$ with respect to a'. Let a lead to sum(A) and let a' lead to sum$'(A')$. If sum(A) has at position i the same value as sum$'(A')$ at the position i, if these values are followed by ones up to position p^*+1 and if sum(A) and sum$'(A')$ have the same values at the positions $k, \ldots, p^*$, the assignments a for s_i and a' for $s_{i'}$ lead to the same SBDD node. This implies an upper bound of $O(n^3)$ for the SBDD size.

Exercise 4.10. If we replace the first k bits and the last l bits of the two factors by zero, we obtain the multiplication of $(n-k-l)$-bit numbers. The former output at position $(n-1+l-k)$ is the new middle bit of multiplication. We always choose $l = 0$ or $k = 0$. Hence, by Theorem 4.5.2, the OBDD size of $\mathrm{MUL}_{i,n}$, $i \leq n-1$, is bounded below by $2^{(i+1)/8}$. This lower bound is exponential, if $i = \Omega(n^\epsilon)$, and not polynomial, if $\omega(\log n)$. We obtain the same lower bound for $\mathrm{MUL}_{2n-1-i,n}$ as for $\mathrm{MUL}_{i,n}$.

Exercise 4.11. If $i = O(\log n)$, $\mathrm{MUL}_{i,n}$ essentially depends on $O(\log n)$ variables and the OBDD size is bounded above by $2^{O(\log n)} = n^{O(1)}$.

Exercise 4.13. The solution of this exercise is contained in the proof of Theorem 5.3.1.

Exercise 4.14. We apply the read-once projections contained in the proof of Theorem 4.6.2. The OBDD size of MUL_n is bounded above by the OBDD size of SQU_{3n+2}. Hence, the OBDD size of SQU_n is bounded below by the OBDD size of $\mathrm{MUL}_{\lfloor (n-2)/3 \rfloor}$. The explicit lower bound is by Theorem 4.5.2 $2^{a(n)}$ where $a(n) = \lfloor (n-2/3 \rfloor /8 = n/24 - O(1)$. We obtain SQU_n as read-once projection from INV_{10n}. Hence, the OBDD size of INV_n is bounded below by $2^{b(n)}$ where $b(n) = \lfloor (n-2)/3 \rfloor /80 = n/240 - O(1)$. The same lower bound holds for DIV_n.

Exercise 4.15. The proof of Theorem 4.7.2 is performed by counting the different subfunctions. A non-constant symmetric function essentially depends

on all its variables and subfunctions of symmetric functions are symmetric on the remaining variables. We always use the variable ordering $x_1, \ldots, x_n$.

Threshold functions. We only have to consider the function $T_{k,n}$, since

$$T_{n+1-k,n}(x_1, \ldots, x_n) = \neg T_{k,n}(\overline{x}_1, \ldots, \overline{x}_n).$$

We need 2 sinks. We have i x_i-nodes, $1 \leq i \leq k$, since we have to count the number of ones and since we reach the 1-sink after having seen k ones. We also have i x_{n+1-i}-nodes, $1 \leq i \leq k$. If we have still i variables not yet tested, we have reached the 0-sink if we have seen less than $k - i$ ones. All other levels have a size of k for the number of $0, \ldots, k-1$ ones already seen. The total size equals

$$2(1 + \cdots + k) + (n - 2k)k + 2 = k(k+1) + nk - 2k^2 + 2 = k(n - k + 1) + 2.$$

Exact counting. The size of $E_{k,n}$ is equal to the size of $E_{n-k,n}$, since we may count zeros instead of ones. Hence, we only consider $E_{k,n}$. We have two sinks. We have i x_i-nodes, $1 \leq i \leq k+1$, since we have to distinguish whether we have seen $0, \ldots, k$ ones. Only if we have seen more than k ones we already reach the 0-sink. The last level contains 2 nodes distinguishing whether we have seen $k-1$ or k ones. A generalization of this argument shows that the last levels contain $k+1, \ldots, 2$ nodes and each other level contains $k+1$ nodes. Hence, the total size equals

$$(1 + \cdots + k) + (2 + \cdots + k) + (n - 2k + 1)(k+1) + 2 =$$

$$k(k+1) - 1 + nk - 2k^2 + k + n - 2k + 1 + 2 = nk - k^2 + n + 2.$$

Modular counting We count the number of ones mod k. The size of the levels increases from 1 to k and at the end it decreases from k to 2. Hence, the total size equals

$$(1 + \cdots + k) + (2 + \cdots + k) + (n - 2k + 1)k + 2 =$$

$$k(k+1) - 1 + nk - 2k^2 + k + 2 = nk - k^2 + 2k + 1.$$

Exercise 4.16. First, we compute the threshold value :

$$\begin{aligned} 2t = \sum_{1 \leq i \leq n} w_i &= \sum_{1 \leq i \leq n/2} 2^{i-1} + \sum_{n/2+1 \leq i \leq n} (2^{n/2} - 2^{n-i}) \\ &= 2^{n/2} - 1 + (n/2)2^{n/2} - (2^{n/2} - 1) \end{aligned}$$

and

$$t = n2^{n/2}/4.$$

If the variables are ordered according to descending weights, the variable ordering equals $x_n, \ldots, x_1$. We consider the different assignments to $x_n, \ldots, x_{n/2+1}$

with the restriction that exactly $n/4$ variables get the value 1. These are $\binom{n/2}{n/4}$ different partial assignments. The partial sums agree in the term $(n/4)2^{n/2} = t$ but they have different negative terms $-\sum_{n/2+1 \leq i \leq n} x_i 2^{n-i}$. The absolute value of this negative term has the binary representation $x_{up} = (x_{n/2+1}, \ldots, x_n)$. Let us consider two different of these assignments namely a and b, w.l.o.g. $|a_{up}| > |b_{up}|$. We claim that we obtain different subfunctions for a and b leading to the proposed lower bound. We consider the common extension by the assignment $(x_1, \ldots, x_{n/2}) = (b_{n/2+1}, \ldots, b_n)$. The assignment b together with this assignment leads to a total weight of t and the output 1 while the assignment a together with this assignment leads to a total weight of less then t and the output 0.

Exercise 4.17. We choose the variable ordering
$x_{n/2}, x_{n/2+1}, x_{n/2-1}, x_{n/2+2}, \ldots, x_1, x_n$. The weighted sum can be written as

$$\sum_{1 \leq i \leq n} w_i x_i = 2^{n/2} \sum_{n/2+1 \leq i \leq n} x_i + \sum_{1 \leq i \leq n/2} 2^{i-1}(x_i - x_{n-i+1}).$$

The value of the second term lies in the interval $[-2^{n/2}+1, 2^{n/2}-1]$. Since the first term is a multiple of $2^{n/2}$ and t is a multiple of $2^{n/2}$ (see the solution of Exercise 4.16), it is sufficient to store the value of $x_{n/2+1} + \cdots + x_n$ ($n/2+1$ possible values) and enough information to decide whether the second term will be nonnegative. If we have tested x_{n-i+1}, there are three possibilities:

- $x_{n/2} = x_{n/2+1}, \ldots, x_i = x_{n-i+1}$ (then the partial sum for the second term equals 0),
- for some j we have $x_{n/2} = x_{n/2+1}, \ldots, x_j = x_{n-j+1}$, $x_{j-1} = 1$, $x_{n-j+2} = 0$ (then we know that the second term is positive),
- for some j we have $x_{n/2} = x_{n/2+1}, \ldots, x_j = x_{n-j+1}$, $x_{j-1} = 0$, $x_{n-j+2} = 1$ (then we know that the second term is negative).

Altogether, the size of the following level, the x_{i-1}-level, is bounded above by $3(n/2+1)$ implying that the size of the next level, the x_{n-i+2}-level, is bounded by $6(n/2+1)$. Hence, the size of each level is bounded by $O(n)$ and the total size by $O(n^2)$.

Exercise 4.18. Let $e_j(z) = 1$ iff $|z| = j$. This function can be represented as a monomial. Moreover,

$$\begin{aligned} f_n(x,y,z) &= \bigvee_{0 \leq j \leq n-1} e_j(z) \wedge \bigvee_{0 \leq i \leq n-1} x_i y_{i+j} \\ &= \bigvee_{0 \leq i,j \leq n-1} e_j(z) x_i y_{i+j} \end{aligned}$$

is a DNF representation with n^2 monomials of length $k+2$ each.

Let π be an arbitrary variable ordering. W.l.o.g. n is a multiple of 4. We consider the level where for the first time $n/2$ x-variables or $n/2$ y-variables have been tested. W.l.o.g. $n/2$ x-variables have been tested. These chosen x-variables belong to $n^2/2$ pairs $x_i y_{i+j}$. Since less than $n/2$ y-variables have been tested, there are at most $n^2/4$ of these pairs such that also the y-variable has already been tested. Hence, there are at least $n^2/4$ pairs (called bad pairs) such that the x-variable is tested in the top part and the y-variable is tested in the bottom part. There are n possible values for the shift $|z|$. By the pigeon-hole-principle, there is one value leading to at least $n/4$ bad pairs. After renumbering the y-variables we can assume that $|z| = 0$. We replace the z-variables by this vector. This leads to the function $x_1y_1 + \cdots + x_ny_n$. We replace all variables not belonging to $n/4$ bad pairs by 0. After renumbering we get the function $x_1y_1 + \cdots + x_{n/4}y_{n/4}$ and all x-variables are tested before all y-variables. This leads to the lower bound $2^{n/4}$ on the OBDD size. All different assignments to the x-variables lead to different subfunctions.

Exercise 4.19. We apply Lemma 4.10.1. If k variables have been tested, at most 2^k nodes can be reached. This implies the size bound for the first $n/2$ levels of the OBDD. If $k > n/2$, the length of the window is bounded above by $n - k$. Hence, each $N(k, s) \leq 2^{n-k}$ and

$$\sum_{n/2<k\leq n} \sum_{0\leq s\leq n-k} 2^{n-k} \leq n \cdot \sum_{n/2<k\leq n} 2^{n-k} \leq 2n2^{n/2}$$

is an upper bound on the size of the last $n/2$ levels of the OBDD.

Exercise 4.20. We apply Lemma 4.10.1 for $k = n/2$ and $s = n/4$. The window has length $n + 1 - k = n/2 + 1$. All variables already tested are in the window. Hence, $w = n/2$. This leads to the lower bound

$$N(n/2, n/4) = \binom{n/2}{n/4}.$$

It is well-known that $\binom{n}{n/2} = \Theta(n^{-1/2}2^n)$ and, therefore, $\binom{n/2}{n/4} = \Theta(n^{-1/2}2^{n/2})$.

Exercise 4.21. For a lower bound, we consider $s_0 + s_1 + \cdots + s_n$ where s_i is the number of different subfunctions after the test of the variables of the ith row. For the inputs which have not reached the 0-sink we have to distinguish the $\binom{n}{i}$ different subsets of the set of columns which may already contain a 1. We may take the sum of all s_i, since the subfunctions considered for s_i essentially depend on $x_{i+1,1}$ while the subfunctions considered for s_{i+1} do not depend on this variable. This leads to the lower bound $\binom{n}{0} + \binom{n}{1} + \cdots + \binom{n}{n} = 2^n$. If we have tested the variables of the ith row and some variables of the next row, we have to distinguish at most $\binom{n}{i} + \binom{n}{i+1}$ subfunctions (besides the constant 0). Either we have seen i ones, one in each of the first i rows and the ones are located in different columns or we have seen $i + 1$ ones, one in each of the first $i + 1$ rows and the ones are located in different columns. This leads to an upper bound of $n \cdot \sum_{0\leq i\leq n-1} \left(\binom{n}{i} + \binom{n}{i+1}\right) = O(n2^n)$.

Chapter 5

Exercise 5.1. The proof that the inner product function is almost ugly is similar to the proof for the disjoint quadratic function in Theorem 5.3.3. The variable ordering, $x_1, y_1, \ldots, x_n, y_n$ leads to linear size. Now we consider an arbitrary variable ordering and a cut after n levels. For the pairs, we fix the assignment $(0,0)$. If the number of singletons is s, we may assume w.l.o.g. that we still have to compute $x_1y_1 \oplus \cdots \oplus x_sy_s$ and that we first test $x_1, \ldots, x_s$. Then we obtain 2^s different subfunctions, for each $S \subseteq \{1, \ldots, s\}$ the $\oplus$-sum of all y_i, $i \in S$. The probabilistic part of the proof is identical to the proof for DQF.

Exercise 5.2. We know from Theorem 4.4.4 that the OBDD size of multiple addition is polynomial.

Now we investigate a random variable ordering and consider the choice of the first $n/2$ variables. The probability that a chosen variable belongs to a column such that no previously chosen variable belongs to this column is at least $1/2$. Hence, by Chernoff's bound, with a probability exponentially close to 1 we have among the first $n/2$ variables variables from at least $n/5$ different columns. This leads to 2^c different subfunctions if we have chosen variables from c different columns. The reason is the following. We may fix the variables in columns without chosen variables in such a way that these columns contain exactly one 1-entry. For the other columns we only investigate assignments with at most one 1-entry. Then we have reduced the problem to the addition of two c-bit binary numbers where the bits of the first number are tested first. Hence, the OBDD size, even for the output at position n, is with a probability exponentially close to 1 at least $2^{n/5}$.

We remark that we can obtain a lower bound of 2^n using a cut after $\Theta(n \log n)$ tested variables and applying the coupon collector's theorem (Motwani and Raghavan (1995)).

Exercise 5.3. The function ROW is almost ugly. It has linear OBDD size for a rowwise variable ordering. The investigation of random variable orderings follows the lines of the solution of Exercise 5.2 with the exception that we consider rows instead of columns. Let r be the number of rows such that at least one variable of the row is among the first $n/2$ variables. We consider the 2^r assignments to the first $n/2$ variables where all variables of each row get the same value. Let two assignments differ in the value of row i. Then we obtain different subfunctions. We fix the remaining variables in such a way that exactly the variables of row i get the value 1. Then we obtain different output values. Hence, the OBDD size is with a probability exponentially close to 1 at least $2^{n/5}$.

Exercise 5.4. Yes, e.g., the functions $x_1y_1 + \cdots + x_ny_n$ and $x_1y_1 \oplus \cdots \oplus x_ny_n$ are almost ugly and are defined by read-once formulas.

Exercise 5.7. The sensitivity of symmetric functions is equal to 1, since the variable ordering has no influence on the OBDD size.

For each polynomial p we find a threshold function with polynomial weights whose sensitivity is at least $\Omega(p(n))$. As an example let $p(n) = n$. Let $n = 3k$. Then we set $w_1 = \cdots = w_k = n^2$, $w_{k+1} = \cdots = w_{2k} = n$, $w_{2k+1} = \cdots = w_{3k} = 1$, and $t = (n^2+n+1)k/2$. The OBDD size for the variable ordering $x_1, \ldots, x_n$ is bounded above by $O(n^2)$. First, we count the number of ones among $x_1, \ldots, x_k$. If this number is less than $k/2$, we reach the 0-sink, and if the number is larger than $k/2$, we reach the 1-sink. Only if the number is equal to $k/2$, we count the number of ones among $x_{k+1}, \ldots, x_{2k}$ and, by the same arguments, reach a sink if the number of ones is not equal to $k/2$. Only if also that number is equal to $k/2$ we have to count the number of ones among $x_{2k+1}, \ldots, x_{3k}$. The three sub-OBDDs for counting have size $O(n^2)$. Now we investigate the variable ordering $x_1, x_{k+1}, x_{2k+1}, x_2, x_{k+2}, x_{2k+2}, \ldots, x_k, x_{2k}, x_{3k}$. After the test of the first $k/2$ triples of variables we have to distinguish all $(k/2+1)^3$ different triples of the number of ones in the blocks $B_1 = (x_1, \ldots, x_k)$, $B_2 = (x_{k+1}, \ldots, x_{2k})$, $B_3 = (x_{2k+1}, \ldots, x_{3k})$. Hence, the OBDD size equals $\Omega(n^3)$. We may generalize this to m groups of variables with the weights $n^{m-1}, \ldots, n, 1$ resp. for the members of the different groups. The upper bound remains $O(n^2)$ and the lower bound is $\Omega(n^m)$ by the same arguments as above (as long as m is a constant). This leads to a sensitivity of $\Omega(n^{m-2})$.

Before we consider functions $f = (f_n)$ where $|f_n^{-1}(1)|$ is polynomially bounded, we investigate the multiplexer function and prove that its sensitivity is $\Omega(2^n/n)$. It has linear size if the address variables are tested first. Now consider variable orderings where the data variables are tested first. Let us consider two different assignments a and b where $a_i \neq b_i$. The corresponding subfunctions are different, since we obtain different outputs if the address equals i. This leads to the lower bound 2^n. If we consider the multiplexer on approximately $c \log n$ variables, the sensitivity equals $\Omega(n^c/\log n)$. The same holds for the function f_n on n variables which is equal to the multiplexer on the last approximately $c \log n$ variables if all other variables are equal to 1 and which outputs 0 otherwise. This function outputs 1 for $O(n^c)$ inputs. Hence, the sensitivity can be larger than any given polynomial.

For the fourth class of functions, we obtain the same type of result as for the last two classes. Let X_i contain n/k variables, let h_i be the exact counting function checking whether exactly half of the variables of X_i are equal to 1, and let g be the conjunction. Similarly to the consideration of the general threshold function, we obtain an $O(n^2)$ bound for blockwise variable orderings and a lower bound of $\Omega(n^k)$ for variable orderings starting with k variables from different blocks, followed by k variables from different blocks, and so on.

It is difficult to estimate the sensitivity of the most significant bit of the squaring function. We need properties of $c = (c_{n-1}, \ldots, c_0)$ where $|c| = \lceil 2^{n-1/2} \rceil$. We know that c consists of the first n bits of the binary representation of $\sqrt{2}$ which follow the binary point where we add 1 at the last position. Let i be the smallest index where $c_i = 1$. Then the function essentially depends on $x_{n-1}, \ldots, x_i$ leading to a lower bound of $\Omega(n-i)$. Using the variable ordering $x_{n-1}, \ldots, x_0$ the OBDD contains exactly one x_j-node for $j \geq i$. If $x_{n-1}, \ldots, x_{j+1}$ have been tested, we either have reached a sink or $x_{n-1} =$

$c_{n-1}, \ldots, x_{j+1} = c_{j+1}$. Only in this situation we test x_j. Hence, the optimal OBDD size equals $n - i + 2$. We conjecture that $n - i = \Theta(n)$.

Let $b = b(n)$ be the number of blocks $B_1, \ldots, B_b$ of c where a block is a largest constant subvector. Since $1 < \sqrt{2} < 3/2$, B_1 is a 0-block if n is large enough. Either B_b or B_{b-1} is a 1-block. To simplify the notation we assume that B_b is a 1-block.

We claim an upper bound of $O(nb)$ independently of the variable ordering. The crucial idea is that it is sufficient to store the crucial index namely the index of the first block where some bit of the input differs from the corresponding c-bit (this index is $b+1$ if no bit of the tested variables differs from the corresponding c-bit). Hence, the OBDD width is bounded above by $b + 1$.

Now we design a bad variable ordering. We only consider the test of the variables in the 1-blocks $B_2, B_4, \ldots, B_{b-2}, B_b$. If the crucial index equals i, the subfunction essentially depends exactly on all variables from $B_1, \ldots, B_{i-1}$ not yet tested. The reason is that we have to decide whether the crucial index i^* of the whole input is even or odd. We already know that $i^* \leq i$ and each variable of $B_1, \ldots, B_{i-1}$ not yet tested can be the only one to change the situation. We start with the test of one variable of each block $B_b, B_{b-2}, \ldots, B_4, B_2$. By the above considerations, these levels have size $1 + \cdots + b/2 = \Theta(b^2)$. We conjecture that $b = \Theta(n)$.

We do not try to prove the two number theoretical conjectures. If both are true, the sensitivity of the most significant bit of the squaring function equals $\Theta(n)$.

Exercise 5.8. We guess a sequence of s BDD nodes and an ordering π of the variables tested in the OBDD representing f. Then we check whether the BDD nodes with node 1 as source lead to a π-OBDD. In the negative case we reject. Otherwise, we have given the OBDD G_f representing f and the π-OBDD. Let g be the function represented by this π-OBDD called G_g. We compute a variable ordering π' such that G_f is a π'-OBDD. The OBDD defines a partial ordering on the set of variables. If an edge leads from an x_i-node to an x_j-node, $x_i \leq x_j$. Hence, we may apply well-known techniques for topological sorting. Afterwards, we can apply the equivalence check for OBDDs with different variable orderings (Corollary 5.7.11).

Exercise 5.9. We design a Turing reduction from the classical SAT problem to the considered problem. Let $c_1, \ldots, c_m$ be a set of clauses defined on $x_1, \ldots, x_n$. Then we compute a circuit C realizing $f = c_1 \wedge \cdots \wedge c_m \wedge (y_1 y_2 + y_3 y_4)$. Let π be the variable ordering $y_1, y_3, y_2, y_4, x_1, \ldots, x_n$. If $c_1 \wedge c_2 \wedge \cdots \wedge c_m$ is not satisfiable, $f \equiv 0$ and π is like each other variable ordering optimal. Otherwise, the π-OBDD starts with an OBDD for $y_1 y_2 + y_3 y_4$ and the 1-sink is replaced by an *id*-OBDD for $c_1 \wedge \cdots \wedge c_m$. The variable ordering π is not optimal, since we may save one node by using the variable ordering $y_1, y_2, y_3, y_4, x_1, \ldots, x_n$. Hence, we can decide the satisfiability of $c_1 \wedge \cdots \wedge c_m$ by checking whether π is an optimal variable ordering for the function realized by C.

Exercise 5.12. For each choice of $I \subseteq \{1, \ldots, n\}$ we like to compute MIN(I), the minimal number of x_i-nodes, $i \in I$, of an OBDD representing f and using a variable ordering which starts with all x_i, $i \in I$, and we like to compute $\pi(I)$, an optimal ordering of these variables. We are interested in $\pi(\{1, \ldots, n\})$. Let everything be done for sets I where $|I| = k$. Now we consider a set I where $|I| = k+1$. For each $i \in I$ we compute the number of subfunctions of f which can be obtained by replacing the variables x_j, $j \in I - \{i\}$, by constants and which essentially depend on x_i. We work with the function table of size 2^n. We copy that table in such a way that we obtain 2^k subtables of size 2^{n-k} which describe the subfunctions we are interested in. Each table can be checked in linear time whether the subfunction essentially depends on x_i. In order to determine the number of different subfunctions essentially depending on x_i, we sort the function tables for the subfunctions essentially depending on x_i. The sorting can be performed with $O(k2^k)$ comparisons of vectors of length 2^{n-k}. Hence, $O(k2^n)$ operations are enough. Let $s(i, I)$ be the resulting number. Then

$$\text{MIN}(I) = \min\{\text{MIN}(I - \{i\}) + s(i, I) \mid i \in I\}.$$

If i leads to the minimal value, $\pi(I)$ is the concatenation of $\pi(I - \{i\})$ and x_i. We store the function table and one copy (size 2^n) and for some k for all $\binom{n}{k}$ sets I where $|I| = k$ the numbers MIN(I) and the partial orderings $\pi(I)$ of length k. Hence, the storage space is bounded by $\max\{k\binom{n}{k} + 2^n \mid 1 \leq k \leq n\} = O(n^{1/2}2^n)$. For the analysis of the run time we consider $\binom{n}{k}$ sets of size k, for each set k indices i and for each i we have a run time of $O(k2^n)$. Hence, the run time is of order

$$\sum_{1 \leq k \leq n} k^2 \binom{n}{k} 2^n \leq n^2 4^n.$$

Exercise 5.13. We have to prove that each for the four criteria is sufficient to prove the asymmetry of f with respect to x_i and x_j.

1. If f is symmetric with respect to x_i and x_j, where w.l.o.g. $i < j$, $f(a_1, \ldots, a_{i-1}, a^*, a_{i+1}, \ldots, a_{j-1}, 1, a_{j+1}, \ldots, a_n) =$ $f(a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_{j-1}, a^*, a_{j+1}, \ldots, a_n)$ implying that $|f^{-1}_{|x_i=1}(1)| = |f^{-1}_{|x_j=1}(1)|$.

2. The node v of a reduced OBDD represents a subfunction of f essentially depending on x_i, since it is labeled by x_i. This subfunction f^* does not essentially depend on x_j, since it is represented by an OBDD without x_j-node. Hence, $f^*_{|x_i=0,x_j=1} = f^*_{|x_i=0} \neq f^*_{|x_i=1} = f^*_{|x_i=1,x_j=0}$ implying that $f_{|x_i=0,x_j=1} \neq f_{|x_i=1,x_j=0}$.

3. Let f^* be the subfunction obtained by replacing the variables on the path from the source to the considered x_i-node by the corresponding constants which activate this path. Then f^* essentially depends on x_i but not on x_j and the same arguments as for Criterion 2 can be applied.

4. Let $v^* \in V_{01} - V_{10}$ (the other case can be handled similarly). Let a be the partial assignment defined by a path from the source to v^* via the 0-edge leaving v and a 1-edge leaving an x_j-node. Let f^* be the corresponding subfunction and let f^{**} be the subfunction for the partial assignment b which fixes the same variables as a and differs in the assignments to x_i and x_j. If f is symmetric with respect to x_i and x_j, we can conclude that $f^* = f^{**}$ and both functions are represented in a reduced OBDD by the same node. Hence, f^{**} is represented at v^* and $v^* \in V_{10}$ in contradiction to the assumption.

Exercise 5.14. The function $s_{n|x_i=1}$ syntactically depends on x_i. This function takes the value 1 if there is some $j > i$ such that $x_j = y_j = 1$ and $x_k \oplus y_k = 1$ for $k > j$ or $y_i = 1$ and $x_k \oplus y_k = 1$ for $k > i$ or there is some $j < i$ such that $x_j = y_j = 1$, $y_i = 0$, and $x_k \oplus y_k$ for $k > j$ such that $k \neq i$. Hence,

$$\begin{aligned}
|s^{-1}_{n|x_i=1}(1)| &= \sum_{i+1 \leq j \leq n-1} 2^{n-j-1} \cdot 4^j + 2^{n-i-1} \cdot 2 \cdot 4^i + \sum_{0 \leq j \leq n-1} 2^{n-j-1} \cdot 4^j \\
&= \sum_{0 \leq j \leq n-1} 2^{n-j-1} \cdot 4^j + 2^{n-i-1} \cdot 4^i \\
&= \sum_{0 \leq j \leq n-1} 2^{n-j-1} \cdot 4^j + 2^{n-1} \cdot 2^i
\end{aligned}$$

which obviously is strictly increasing with i.

Exercise 5.15. The number is equal to $2^{3 \cdot 2^{n-2}}$. For all $(a_1, a_2, a_3, \ldots, a_n)$ where $(a_1, a_2) \in \{(0,0), (0,1), (1,1)\}$ we may choose arbitrary function values but $f(1, 0, a_3, \ldots, a_n)$ has to be equal to $f(0, 1, a_3, \ldots, a_n)$.

Exercise 5.16. Sometimes we refer to the criteria of Theorem 5.5.4.

1. DSA_n is not symmetric with respect to any pair of variables. For two data variables we apply Criterion 2 for the usual optimal variable ordering. For one address variable and one data variable we apply Criterion 2 and a variable ordering starting with all address variables but not with the considered address variable. For two address variables, we consider a data vector with a single 1-entry at a position such that the two considered address variables take different values. If we switch these address bits, the output switches from 1 to 0.

2. HWB_n is not symmetric with respect to any pair of variables x_i and x_j. W.l.o.g. $i < n$. We can construct an input a with $a_i = 0$ and $a_j = 1$ and altogether i ones leading to the output 0. If we switch the values of x_i and x_j, the output equals 1.

3. The maximal sets of symmetric variables of ADD_n are $\{x_i, y_i\}$, $0 \leq i \leq n-1$. It is obvious that ADD_n is symmetric with respect to x_i and y_i. For all other pairs we can apply Criterion 1 and the solution of Exercise 5.14.

4. MUL_n, $n \geq 2$, is not symmetric with respect to any pair of variables. For x_i and x_j, we set $y_{n-1} = \cdots = y_1 = 0$ and $y_0 = 1$ and consider the output p_j which is equal to x_j. Similar arguments work for y_i and y_j. For x_i and y_i we fix all other variables where only y_0 (if $j \neq 0$) gets the value 1. The output p_j is equal to x_j. If $j = 0$, we only set y_1 to 1 and consider p_{j+1}. If $n = 1$, we obtain the symmetric function $x_0 y_0$.

5. The maximal sets of symmetric variables for ROW_n are the sets containing the variables of one row. Nothing changes if we switch the values of two entries of the same row. If two variables belong to different rows, w.l.o.g. row 1 and row 2, we can fix all other variables in such a way that only the variables of row 1 get the value 1. The resulting subfunction is equal to the chosen variable of row 1.

6. $\mathrm{ROW}_n + \mathrm{COL}_n$, $n \geq 3$, is not symmetric with respect to any pair of variables. We consider two different variables. They do not belong to the same row or not to the same column. In the first case, we use the same arguments as in the proof for ROW_n and in the second case we argue with respect to the columns. In the first case, it is not possible to obtain a column of 1-entries, since for $n \geq 3$ we have replaced at least one row by 0-entries. If $n = 2$, the function is symmetric with respect to x_{11} and x_{22} and with respect to x_{12} and x_{21}.

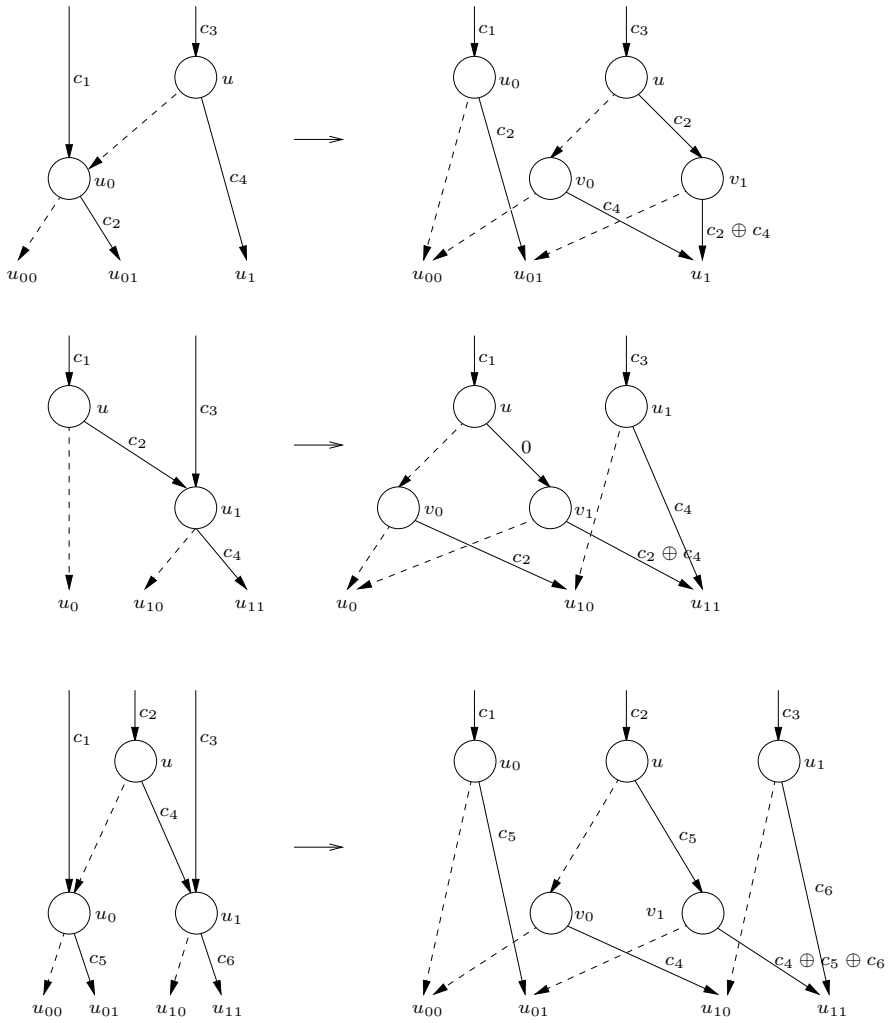
7. DQF_n is symmetric with respect to the sets $\{x_i, y_i\}$. Obviously, we can interchange x_i and y_i. Now we consider one variable z_i of the ith pair and the variable z_j of the jth pair. If we replace all variables besides the partner of z_i by 0 and the partner of z_i by 1, we obtain the subfunction z_i.

Exercise 5.17. Let $a = (a_0, \ldots, a_{k-1})$, $x = (x_0, \ldots, x_{n-1})$, and $y = (y_1, \ldots, y_n)$ where $n = 2^k$. Let $f_n(a, x, y) = \mathrm{MUX}_n(a, x) \oplus \mathrm{MAJ}_n(y)$. Let $\pi_1 = (a, x, y)$. The π_1-OBDD for f_n starts with an OBDD for MUX_n with $2n - 1$ inner nodes. Afterwards, we have to represent MAJ_n and $\overline{\mathrm{MAJ}}_n$. Since MAJ_n is monotone, these sub-OBDDs can only share the sinks (see the solution of Exercise 3.4). The OBDD size of MAJ_n equals $\frac{1}{4}n^2 + \frac{1}{2}n$ (see Theorem 4.7.2.i). Hence, the π_1-OBDD size of f_n equals $\frac{1}{2}n^2 + 3n + 1$. The function f_n is symmetric with respect to the y-variables. Group sifting leads to $\pi_2 = (a, y, x)$. Then we start with a complete tree on the a-variables. This tree has $n - 1$ inner nodes. We have to "store" the value of $|a|$ and need n disjoint copies of an OBDD for MAJ_n, altogether $\frac{1}{4}n^3 + \frac{1}{2}n^2$ inner nodes. Finally, we need $2x_i$-nodes representing x_i and $\overline{x}_i$ and 2 sinks. Hence, the π_2-OBDD size of f_n equals $\frac{1}{4}n^3 + \frac{1}{2}n^2 + 3n + 1$. A typical sifting process will be aborted. Let $\pi_3 = (y, a, x)$. Then we start with an OBDD for MAJ_n with $\frac{1}{4}n^2 + \frac{1}{2}n$ inner nodes. Afterwards, $4n$ nodes are sufficient to represent MUX_n and $\overline{\mathrm{MUX}}_n$ simultaneously. Hence, the π_3-OBDD size of f_n equals $\frac{1}{4}n^2 + \frac{9}{2}n$ and π_3 is better than π_1.

Exercise 5.19. Let $x_1, \ldots, x_m$, $m \leq n$, be the variables which each label one leaf of the formula and let $y_1, \ldots, y_k$, $k = O(\log n)$, be the other variables. We

construct a read-once formula G_n on the x-variables. For each gate v where one subtree contains only y-leaves we eliminate the y-subtree and the gate. The output of the other subtree is directly connected with the successor of the eliminated gate. Theorem 4.11.2 ensures the existence of a variable ordering π of the x-variables (which indeed can be computed efficiently) such that the π-OBDD size of the function g_n represented by G_n is polynomially bounded with respect to $m \leq n$. We use a variable ordering starting with the y-variables in an arbitrary ordering followed by the x-variables ordered according to π. An OBDD for the given formula F_n may start with a complete binary on the y-variables. This tree has polynomially many leaves. It is sufficient to prove that each subfunction for an assignment to the y-variables has polynomial π-OBDD size. The x-leaves of F_n are the same as in G_n. Let us consider the influence of the y-variables in G_n and F_n. Let v be a gate as considered in the beginning of the proof. First we investigate G_n. If the second subformula represents h, the function h is also input of the successor of v. If the second subformula represents h^* as subformula of F_n, the gate v represents one on the functions 0,1,h^* and $\overline{h}^*$, since the y-variables are replaced by constants. The proof of Theorem 2.1.4 shows that it makes no difference whether we have to represent h^* or $\overline{h}^*$. If instead of this we only have to represent a constant, this makes the resulting OBDD only smaller. Altogether, each subfunction obtained by replacing the y-variables by constants has a π-OBDD size which is not larger than the OBDD size of g_n and we have constructed a π-OBDD of polynomial size.

Exercise 5.20. We consider Figure 5.7.1. The first two cases do not change if edges can be complemented. Here we illustrate the new cases with the compl-bits.



Exercise 5.21. The solution of Exercise 5.20 shows that we obtain the same results for the swap operation on OBDDs with complemented edges as described in Theorem 5.7.2 for OBDDs without complemented edges. The first bound of Theorem 5.7.4 for jump-up and the second bound of Theorem 5.7.5 for jump-down are based on the investigation of a sequence of swap operations. These results also hold for OBDDs with complemented edges. It is easy to transfer the second bound of Theorem 5.7.4. We use the same notation but now for OBDDs with complemented edges. An x_k-node, $j \leq k \leq i-1$, can represent $g = f_{|x_1=a_1,\ldots,x_{k-1}=a_{k-1}}$ and its negation $\overline{g}$. We obtain at most four subfunctions $g_{|x_k=0}$, $g_{|x_k=1}$,$\overline{g}_{|x_k=0}$, $\overline{g}_{|x_k=1}$ which can be represented by at most two nodes. The result on s_i^* holds, since it is based on graph-theoretical arguments. Now we investigate the first bound of Theorem 5.7.5. Before the jump-operation we

represent all functions $f_{|x_1=a_1,\ldots,x_{k-1}=a_{k-1}}$ essentially depending on x_k where a function and its negation can be represented at the same node. Let g be a subfunction of f where the variables $x_1,\ldots,x_{i-1},x_{i+1},\ldots,x_{k-1}$ are replaced by constants. By Shannon's decomposition rule, $g = x_i g_{|x_i=1} + \overline{x}_i g_{|x_i=0}$ and $g_{|x_i=1}$ and $g_{|x_i=0}$ are represented in the given OBDD on one of the levels $x_k,\ldots,x_n$ or the sink level. At least one of the subfunctions has to essentially depend on x_k. If we replace in $x_i g_{|x_i=1} + \overline{x}_i g_{|x_i=0}$ the function $g_{|x_i=1}$ with $\overline{g}_{|x_i=1}$, we do not obtain $\overline{g}$. Hence, we only obtain the bound

$$s_1 + \cdots + s_i - 1 + (2s_{i+1} + \cdots + 2s_n + 2)^2$$

which follows by eliminating the complemented edges on the last levels before applying the old arguments.

Exercise 5.22. Let us consider the multiplexer with the following variable orderings:

- $y_0,\ldots,y_{n-1},x_0,\ldots,x_{k-1}$ (π_1)
- $y_{n-1},y_0,\ldots,y_{n-2},x_0,\ldots,x_{k-1}$ (π_2)
- $y_0,\ldots,y_{n-2},x_0,\ldots,x_{k-1},y_{n-1}$ (π_3)

It is easy to see that π_1 is the result of jump-down $(1,n)$ applied to π_2 and π_1 also is the result of jump-up $(n+k,n)$ applied to π_3. The y_{n-1}-level of a π_2-OBDD or π_3-OBDD has size 1, while the y_{n-1}-level of a π_1-OBDD has size 2^{n-1}. The last bound follows, since the OBDD starts with a complete binary tree as long as only data variables are tested (see the solution of Exercise 5.7).

Exercise 5.23. The functions ROW_n and COL_n are defined on $N = n^2$ variables. Their OBDD size is $O(N)$ (see Theorem 4.12.1) for a rowwise resp. columnwise variable ordering. The OBDD size of $\mathrm{ROW}_n + \mathrm{COL}_n$ grows exponentially (see Theorem 4.12.2 and Theorem 6.2.13).

Exercise 5.24. The solution of Exercise 4.5 shows that $3n \pm O(1)$ OBDD nodes are sufficient for the representation according to the variable ordering $x_0,y_0,\ldots,x_{n-1},y_{n-1}$. On the x_j-level we distinguish whether $c_{j-1} = 0$ or $c_{j-1} = 1$. On the y_j-level we have a node for the situation that $x_j \oplus c_{j-1} = 1$. If we now test y_{n-1} at the top, the number on these levels gets doubled (with the exception of the x_{n-1}-level), since the four different values for (y_{n-1},c_{j-1}) lead to different subfunctions essentially depending on x_j and, if $x_j \oplus c_{j-1} = 1$, the different values for y_{n-1} lead to different subfunctions essentially depending on y_j. Hence, the size for the variable ordering $y_{n-1},x_0,y_0,\ldots,x_{n-2},y_{n-2},x_{n-1}$ equals $6n \pm O(1)$. This proves the optimality of the second bound of Theorem 5.7.4.

Exercise 5.25. Let the given variable ordering be $\ldots,y_1,\ldots,y_m,z_1,\ldots,z_k,\ldots$ where the y-variables as well as the z-variables are sets of symmetric

variables. The pointers to one of the y-levels (z-levels) from the upper part always point to the y_1-level (z_1-level). We replace the y_1-nodes by nodes for a new variable y which may take any value in $\{0, \ldots, m\}$ and have $m+1$ outgoing edges labeled by $0, \ldots, m$ resp. The edge with label i leads to the node which is reached by inputs where $y_1 + \cdots + y_m = i$ (this node is uniquely determined for reduced OBDDs). The same is done for the z_1-nodes. Now we generalize the swap operation described in Figure 5.7.1. Nodes on the z-level reached also from nodes above the y-level only lose their pointers from the y-level. Nodes on the y-level not reaching nodes on the z-level do not have to be changed. Nodes on the z-level only reached from nodes on the y-level will be eliminated. We consider a y-node with at least one outgoing edge leading to a z-node. This y-node is relabeled by z with now $k+1$ outgoing edges reaching new y-nodes with new outgoing edges. The path for $z = i$ and $y = j$ reaches the same node as the path for $y = j$ and $z = i$ before. Finally the y-nodes and z-nodes are replaced by sub-OBDDs. As in the binary case, it is possible that reduction rules can be applied.

Exercise 5.26. We investigate the OBDD size for the three variable orderings.

1. $x_1, \ldots, x_n, y$. We have to decide whether $||x||_n := x_1 + \cdots + x_n$ is equal to k (this leads to the 1-sink), $||x||_n = 2k+1$ (this leads to a y-node), or $||x||_n \notin \{k, 2k+1\}$ (this leads to the 0-sink). Obviously, we only have to distinguish different partial sums. Before having tested x_n, we only may save nodes if we can replace them by 0-sinks. Hence, we start with i x_i-nodes, altogether $1 + \cdots + n = \frac{1}{2}n^2 + \frac{1}{2}n$ nodes. We cannot save nodes on the levels $x_1, \ldots, x_{2k+2}$. Even if $x_1 = \cdots = x_{2k+1} = 0$, we may have k ones in the input. If $||x||_{2k+2} = 0$, $||x||_n < k$. If $||x||_{2k+2} = 2k+2$, also $||x||_n > 2k+1$. If $||x||_{2k+2} = k+1$, $k < ||x||_n < 2k+1$. We save 3 nodes. The number of nodes which are saved increases on each level by 3. We have two intervals of partial sums which are represented by nodes namely $k-i, \ldots, k$, and $2k+1-i, \ldots, 2k+1$ if there are still i x-variables which have to be tested. The next level has one more node (before the replacement by 0-sinks) but two partial sums less that have to be represented namely $k-i$ and $2k+1-i$. Hence, the number of nodes replaced by 0-sinks equals

   $$3(1 + \cdots + (k-1)) = \frac{3}{2}k(k-1) = \frac{3}{2} \cdot \frac{n-1}{3} \cdot \frac{n-4}{3} = \frac{1}{6}n^2 - \frac{5}{6}n + \frac{4}{6}.$$

   We have to add 3 for the y-node and the two sinks. Altogether, the number of nodes equals

   $$\frac{1}{2}n^2 + \frac{1}{2}n - (\frac{1}{6}n^2 - \frac{5}{6}n + \frac{4}{6}) + 3 = \frac{1}{3}n^2 + \frac{4}{3}n + \frac{7}{3}.$$

2. $y, x_1, \ldots, x_n$ We start with one y-node whose edges point to an SBDD on the x-variables where we have to represent f_0 checking whether $||x_n|| = k$ at the 0-successor and f_1 checking whether $||x||_n \in \{k, 2k+1\}$ at the

1-successor. After k tests it is still possible to have k or $2k+1$ ones in the input. The first $k+1$ levels contain $2(1+\cdots+(k+1)) = (k+1)(k+2)$ nodes, since the OBDDs for the subfunctions with respect to y cannot share nodes. On level $k+2$, we have the partial sums $0,\ldots,k+1$. The last sum can be replaced for f_0 by the 0-sink, since the number of ones is already too large. The partial sum 0 for f_1 can be merged with the partial sum 0 for f_0, since the value $2k+1$ is no longer reachable. Moreover, we can merge the partial sum $k+1$ for f_1 with the partial sum 0 for f_0, since we are only looking for k further ones. On each of the levels $k+2,\ldots,2k+1$ we have the $k+1$ nodes for the sub-OBDD for f_0. They check whether the number of ones in the remaining input is equal to $0,\ldots,k$. For f_1 we only need additional nodes for those partial sums such that $||x||_n = k$ and $||x||_n = 2k+1$ are still possible. These are at first the partial sums $1,\ldots,k$ and at the end the partial sum k. The number of nodes on these levels equals $k(k+1)+k+\cdots+1 = \frac{3}{2}(k^2+k)$. On the levels $2k+2,\ldots,3k+1$ we have $(k+1)+\cdots+2 = \frac{1}{2}k^2+\frac{3}{2}k$ nodes, since all nodes for f_1 can be merged with nodes for f_0. Additionally, we have to count the sinks. This leads to the following number of nodes:

$$1+(k^2+3k+2)+(\frac{3}{2}k^2+\frac{3}{2}k)+(\frac{1}{2}k^2+\frac{3}{2}k)+2 =$$

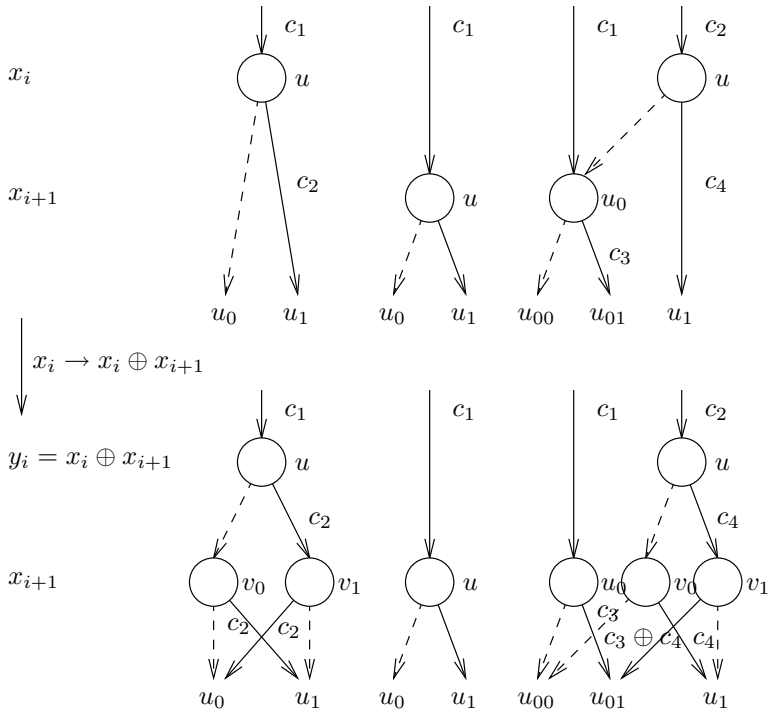
$$3k^2+6k+5 = 3(\frac{n-1}{3})^2+6\frac{n-1}{3}+5 = \frac{1}{3}n^2+\frac{4}{3}n+\frac{10}{3}.$$

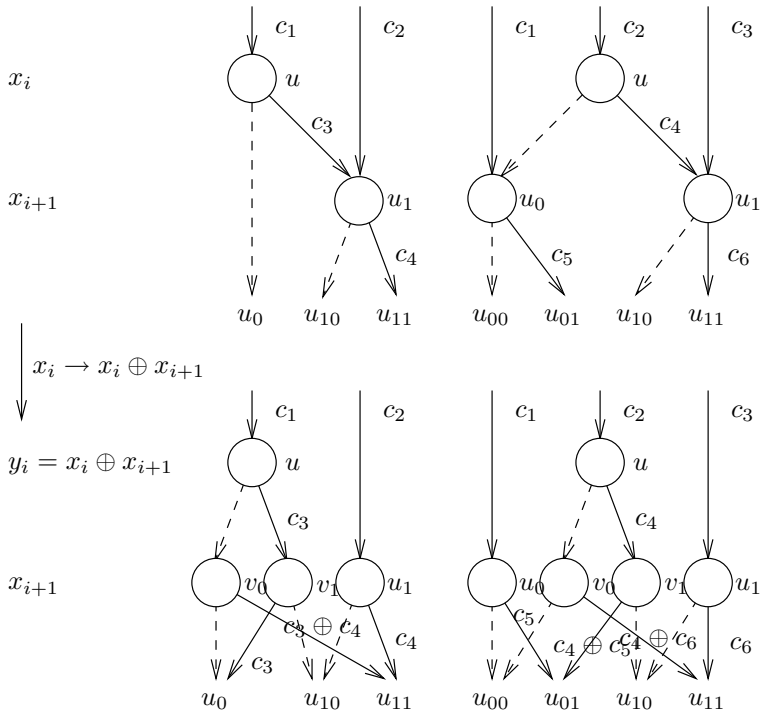
3. $x_1,\ldots,x_{n/2},y,x_{n/2+1},\ldots,x_n$. The first $n/2$ levels have the same size as for the first variable ordering namely $1+\cdots+n/2 = \frac{1}{8}n^2+\frac{1}{4}n$. Only if it is still possible that $||x||_n = 2k+1$ we have to test y. If the partial sum equals s, the largest possible value is $s+n/2$ and $s+n/2 = 2k+1 = 2\frac{n-1}{3}+1$ iff $s = \frac{2}{3}n+\frac{1}{3}-\frac{1}{2}n = \frac{1}{6}n+\frac{1}{3}$. Hence, we have $\frac{n}{2}-(\frac{1}{6}n+\frac{1}{3})+1 = \frac{1}{3}n+\frac{2}{3}$ y-nodes. Afterwards, the number of nodes is equal to the case of the second variable ordering namely $2+\cdots+(\frac{n}{2}+1) = \frac{1}{8}n^2+\frac{3}{4}n$. We have to add the number of the sinks. The resulting OBDD size equals

$$\frac{1}{8}n^2+\frac{1}{4}n+\frac{1}{3}n+\frac{2}{3}+\frac{1}{8}n^2+\frac{3}{4}n+2 = \frac{1}{4}n^2+\frac{4}{3}n+\frac{8}{3}.$$

.

Exercise 5.28. This exercise can be solved by a generalization of Figure 5.9.1.


x_i
x_{i+1}
c_1
u
c_2
u_0
u_1
c_1
u
u_0
u_1
c_1
c_2
u
c_4
u_0
c_3
u_{00}
u_{01}
u_1
$x_i \to x_i \oplus x_{i+1}$
$y_i = x_i \oplus x_{i+1}$
x_{i+1}
c_1
u
c_2
v_0
v_1
c_2
c_2
u_0
u_1
c_1
u
u_0
u_1
c_1
c_2
u
c_4
u_0
v_0
v_1
c_3
$c_3 \oplus c_4$
c_4
u_{00}
u_{01}
u_1


x_i
x_{i+1}
c_1
c_2
u
c_3
u_1
c_4
u_0
u_{10}
u_{11}
c_1
c_2
c_3
u
c_4
u_0
u_1
c_5
c_6
u_{00}
u_{01}
u_{10}
u_{11}
$x_i \to x_i \oplus x_{i+1}$
$y_i = x_i \oplus x_{i+1}$
x_{i+1}
c_1
c_2
u
c_3
v_0
v_1
u_1
$c_3 \oplus c_4$
c_4
c_3
u_0
u_{10}
u_{11}
c_1
c_2
c_3
u
c_4
u_0
v_0
v_1
u_1
c_5
$c_4 \oplus c_5$
$c_4 \oplus c_6$
c_6
u_{00}
u_{01}
u_{10}
u_{11}

Chapter 6

Exercise 6.1. We consider an arbitrary path starting at the source and stopping at some inner node. Then there exist some $i \geq 0$ and $j \geq 0$ such that the variables $x_1, \ldots, x_i$ and $x_{n-j}, \ldots, x_n$ have been tested. If there are only two variables left, i.e., $n - j - i = 3$, we have reached a sink, since the last two levels are eliminated by the reduction rules. Otherwise, the 0-edge leads to an x_{n-j-i}-node and the 1-edge leads to an x_{i+1}-node. We always start with a test of x_n. Hence, the number of different variable orderings equals 2^{n-3}.

Exercise 6.2. Let G_n be a graph ordering on n variables describing at most $p(n)$ (polynomially many) different variable orderings. We describe G_n by a tree-like structure. More precisely, each node except the sink is reached only by edges from one node. A node v is called branching node if v has two different direct successors. Such a representation is possible with at most $p(n)$ branching nodes. We follow the path from the source to the sink which always chooses the successor which has the smaller number of successors which are branching nodes (ties can be broken arbitrarily). This path has at most $\lceil \log p(n) \rceil$ branching nodes. Hence, there is an assignment a to at most $\lceil \log p(n) \rceil$ variables such that the graph ordering G_n checks all extensions of this partial assignment according to the same variable ordering.

Let us start with a G_n-FBDD representing HWB_n and let us replace variables according to the partial assignment a. Then we obtain an OBDD representing the corresponding subfunction HWB'_n. It is sufficient to prove an exponential lower bound on the OBDD size of HWB'_n. There are at least $n - 3\lceil \log p(n) \rceil$ free variables among the variables x_i, $\lceil \log p(n) \rceil \leq i \leq n - \lceil \log p(n) \rceil$. Hence, there is an interval $x_i, \ldots, x_{i+l}$ of $l = \Omega(n/\log n)$ free variables. We extend the partial assignment a by replacing all variables except $x_{i+1}, \ldots, x_{i+l}$ by constants. Let $x_i = 0$ and let the total number of variables replaced by 1 be equal to i. The resulting function is the hidden weighted bit function on $\Omega(n/\log n)$ variables and its OBDD size is bounded below by $2^{\Omega(n/\log n)}$ (see Theorem 4.10.2).

Exercise 6.3. If f is not k-mixed, two different replacements of the same k variables lead to the same subfunction. After renumbering we obtain

$$f_{|x_1=a_1,\ldots,x_k=a_k} = f_{|x_1=b_1,\ldots,x_k=b_k}$$

and $f(a_1, \ldots, a_k, c_{k+1}, \ldots, c_n) = f(b_1, \ldots, b_k, c_{k+1}, \ldots, c_n)$ for all $c_{k+1}, \ldots, c_n \in \{0, 1\}$ and $a_1 \neq b_1$. Let $V = \{x_1, \ldots, x_k\}$ and $i = 1$. If f is k-stable, there are constants $d_{k+1}, \ldots, d_n$ such that $f_{|x_{k+1}=d_{k+1},\ldots,x_n=d_n}$ is one of the functions x_1 or $\overline{x}_1$. Since $a_1 \neq b_1$,

$$f_{|x_{k+1}=d_{k+1},\ldots,x_n=d_n}(a_1, \ldots, a_k, 0, \ldots, 0) \neq f_{|x_{k+1}=d_{k+1},\ldots,x_n=d_n}(b_1, \ldots, b_k, 0, \ldots, 0)$$

or

$$f(a_1, \ldots, a_k, d_{k+1}, \ldots, d_n) \neq f(b_1, \ldots, b_k, d_{k+1}, \ldots, d_n)$$

in contradiction to the conclusions implied by the assumption that f is not k-mixed.

Exercise 6.4. W.l.o.g. n is even. We prove that DET_n is $n/2$-stable and, therefore, $n/2$-mixed (Exercise 6.3). This leads by Lemma 6.2.4 to a lower bound of $2^{n/2}-1$. Let V be a set of at most $n/2$ variables and $x^* \in V$. We may interchange the rows and columns such that $x_{ij} \in V$ implies $1 \leq i,j \leq n/2$ and $x^* = x_{11}$. We choose the following assignment to the variables outside V. Let $x_{2,n/2+1} = x_{3,n/2+2} = \cdots = x_{n/2,n-1} = x_{n/2+1,2} = x_{n/2+2,3} = \cdots = x_{n-1,n/2} = x_{n,n} = 1$. All other variables are set to 0. We claim that the resulting subfunction is equal to x_{11}. In order that $x_{1,\pi(1)}x_{2,\pi(2)}\cdots x_{n,\pi(n)} = 1$, it is necessary to choose $\pi(n/2+1) = 2, \ldots, \pi(n-1) = n/2, \pi(n) = n$, since these are the only ones in these rows. Moreover, we have to choose $\pi(2) = n/2+1, \ldots, \pi(n/2) = n-1$, since these are the only ones in these columns. Hence, only one of the $n!$ permutations π can lead to a 1 and this depends only on the value of x_{11}. It is even possible to prove that DET_n is $(n-1)$-stable.

Exercise 6.5. The function UHC_n is defined on the variables x_{ij}, $1 \leq i < j \leq n$, and decides whether the undirected graph $G(x)$ defined by x contains a Hamiltonian circuit. We claim that UHC_n is $\lfloor n/4 \rfloor$-stable.

W.l.o.g. n is odd. Let V be a set of $\lfloor n/4 \rfloor$ variables and let w.l.o.g. $x_{1,n} \in V$ be the considered special variable. We assign constants to the variables outside V such that the resulting subfunction equals $x_{1,n}$. Since $|V| = \lfloor n/4 \rfloor$, there are at least $\lfloor n/2 \rfloor$ vertices (called fixed vertices) such that we have to replace all variables describing edges adjacent to these vertices by constants. The vertices 1 and n are not fixed. After renumbering we may assume that the vertices $2, 4, \ldots, n-1$ (n is odd) are fixed. We replace the variables $x_{1,2}, x_{2,3}, x_{3,4}, x_{4,5}, \ldots, x_{n-2,n-1}$, $x_{n-1,n}$ by ones and all other variables outside V by zeros. The degree of each vertex $2i$ is known to be 2. Hence, a Hamiltonian circuit has to contain the edges $(2i-1, 2i)$ and $(2i, 2i+1)$ implying that $(1, 2, \ldots, n-1, n, 1)$ is the only possible Hamiltonian circuit. Hence, the resulting subfunction outputs 1 iff $x_{1,n} = 1$.

Exercise 6.6. We describe the subfunctions represented in FBDDs or OBDDs representing f. First, we consider a π-OBDD. Let f_{ij} be the subfunction obtained from f by replacing i of the first $i+j$ variables by zeros and j of these variables by ones. The reduced π-OBDD for $\pi = id$ contains besides the sinks nodes for the non-constant subfunctions $f_{ij}, 0 \leq i+j \leq n-1$. Let G be an FBDD representing f. We follow the path starting at the source choosing at first i 0-edges and then j 1-edges. We reach a node representing f_{ij} on some subset of $n-i-j$ variables. Hence, each FBDD contains at least as many nodes as a reduced OBDD.

Exercise 6.7. The number of cliques of the considered special kind is $O(n^3)$. There are n choices for i and afterwards $\binom{n-k(n)+2}{2}$ possibilities to choose the two further nodes (some special cliques are counted more than once). For each special clique we choose the monomial consisting of those positive literals describing the edges of the clique. The function $f_{k(n),n}$ is the disjunction of these $O(n^3)$ monomials.

Exercise 6.8. Let $m(n) = \binom{\lfloor n^{1/3} \rfloor}{2} - 1$. Similarly to the proof of Theorem 6.2.7 we prove that $f_{k(n),n}$ is $m(n)$-stable. Let V be a set of at most $m(n)$ variables and let $x_{ij} \in V$. Let A be the set of vertices $z \notin \{i,j\}$ such that some variable of V describes an edge adjacent to z. Then $|A| \leq 2m(n) - 2$. Thus $B := \{1,\ldots,n\} - A$ contains at least $n - 2m(n) + 2$ vertices. The set B consists of at most $n^{2/3}$ successive subsequences mod n. The length of the longest subsequence is at least $(n - 2m(n) + 2)/n^{2/3} \geq n^{1/3} - 1$. Hence, we can choose some r such that $r, r+1, \ldots, r+k(n)-3 \bmod n \in B$. We choose a set D of $k(n)$ vertices from B among them i, j, and $r, r+1, \ldots, r+(k(n)-3) (\bmod\ n)$. We replace the variables outside V in the following way by constants. Exactly the variables describing edges between vertices of D are replaced by ones. We claim that the resulting subfunction equals x_{ij}. If $x_{ij} = 1$, we obtain the special clique on D. Let $x_{ij} = 0$. The graph does not contain the clique on D. All nodes in $B - D$ are isolated. Hence, a $k(n)$-clique has to contain a vertex v from A. Since the number of edges on A is bounded above by $m(n)$ (only the edges described by the variables in V can exist), a $k(n)$-clique also has to contain a vertex w from B. But the edge between v and w does not exist. Hence, $x_{ij} = 0$ implies that the graph does not contain a $k(n)$-clique.

Exercise 6.9. We apply Corollary 6.2.8. W.l.o.g n is even. We prove that $cl_{n/2,k(n/2)}$ is a read-once projection of $cl_{n,n/2}$. We choose $k(n/2)$ nodes among the nodes $n/2+1,\ldots,n$ and replace all variables describing edges adjacent to these nodes by zeros. These nodes are isolated. We replace all variables describing edges between a node $i \leq n/2$ and a remaining node $j > n/2$ by ones. Hence, a $k(n)$-clique on $1,\ldots,n/2$ can be extended to an $n/2$-clique on $1,\ldots,n$ and each $n/2$-clique on $1,\ldots,n$ contains a $k(n)$-clique on $1,\ldots,n/2$.

Exercise 6.10. The solution of this exercise is presented before Theorem 7.2.6.

Exercise 6.11. We define a general threshold function T_n^{**} which is based on the general threshold function T_n^* from Definition 4.8.2. Let $n = k^2$ for an even k. Let $m := \lceil \log k \rceil + 1$. Then let $w_{ij} := 2^{(i-1)m} + 2^{(k+j-1)m}$ and let the threshold value t be half the sum of all w_{ij}. Now we may consider the weight as $2km$-bit numbers with $2k$ special positions where the weights may have a 1-entry. Only k weights have a 1-entry at a fixed special position. Hence, there can be no carry from one special position to another special position. We may use the proof of Theorem 4.8.3 with some obvious changes to obtain in the same way a $2^{\sqrt{n}/2}$ lower bound on the OBDD size of T_n^{**}. We claim that at most $2^{k/2-t}$, $t := \lceil (k/2)^{1/2} \rceil$, different partial assignments to some set of $k/2$ variables lead to the same subfunction.

First, we argue how this claim implies an exponential bound on the FBDD size of T_n^{**}. We consider the $2^{k/2}$ paths of length $k/2$ starting at the source of a complete FBDD representing T_n^{**}. Only paths where the same variables are tested can lead to the same node, since the FBDD is complete. Hence, the claim proves a lower bound of 2^t on the size of the complete FBDD. This implies a lower bound of $2^t/n = 2^{\Omega(n^{1/4})}$ on the size of arbitrary FBDDs representing T_n^{**}.

Finally, we prove the claim. Let a set of $k/2$ variables be fixed. The variables x_{ij} are considered as members of a $k \times k$-matrix. Let s be the maximal number of rows or columns containing at least one of the chosen variables. Then $s \geq t$. W.l.o.g. we have s rows containing chosen variables. We investigate how many partial assignments lead to the same partial sum. It follows from the proof of Theorem 4.8.3, more precisely its variant for T_n^{**}, that partial assignments leading to different partial sums also lead to different subfunctions. Let the ith row be one of the rows containing at least one of the chosen variables. At most half of the partial assignments lead to the same partial sum at the specific position belonging to row i. Since the assignment of constants to the variables of different rows is independent of each other and since the carries are stopped before the next specific position, at most $2^{k/2-s} \leq 2^{k/2-t}$ partial assignments lead to the same partial sum.

A proof of an exponential lower bound for T_n^* is technically much more involved. The case that we can consider s rows is not too difficult. Since the assignments to the variables of different rows are independent, it is possible to show that at most $2^{n/2-s}$ partial assignments lead to the same bits at the specific positions. If we have to consider s columns, we may have a carry from the rightmost half of the binary representation of the weights. This carry is not independent from the partial sums at the specific positions for the columns belonging to chosen variables. We omit the details.

Exercise 6.12. This proof follows the general lower bound technique discussed at the beginning of Section 6.2. The crucial idea is the construction of a suitable prefix free set of paths. Then the proof follows standard techniques. Nevertheless, it is technically involved. A detailed proof is published in the paper "Complexity Theoretical Results on Partitioned (Nondeterministic) Binary Decision Diagrams" (B.Bollig and I.Wegener), Theory of Computing Systems 32, 487-503, 1999 as proof of Theorem 12 (pages 497-499).

Exercise 6.13. The function f_n^{**} is the $\oplus$-sum of all $x_i y_j$ such that $x_i y_j$ is a prime implicant of f_n^*. This representation is the unique ring-sum-expansion (RSE) of f_n^{**}. If a variable, w.l.o.g x_i, is replaced by 0, this eliminates all terms of the RSE containing x_i. If x_i is replaced by 1, a term $x_i y_j$ is replaced by y_j and a term x_i is replaced by 1. Afterwards, we have to eliminate terms which occur twice (in general for an even number).

It follows from these considerations that each subfunction obtained by the assignment of constants to at most $n^{1/2} - 1$ variables essentially depends on all variables not replaced by constants. We consider the $2^{n^{1/2}-1}$ paths of length $n^{1/2} - 1$ starting at the source and claim that they lead to different subfunctions. We only have to consider partial assignments of the same set of variables, since otherwise the resulting subfunctions essentially depend on different sets of variables. Let us consider two partial assignments, where w.l.o.g. $x_i = 0$ for one assignment and $x_i = 1$ for the other one. The variable x_i has $n^{1/2}$ partners y_j. Each variable $x_k, k \neq i$, has only one partner in common with x_i. Since only $n^{1/2} - 2$ variables besides x_i are replaced by constants, there exists a partner

y_j of x_i such that y_j is not replaced by a constant and no variable $x_{i'}, i' \neq i$, which also is a partner of y_j is replaced by a constant. The assignment where $x_i = 0$ destroys $x_i y_j$ and all other terms $x_{i'} y_j, i' \neq i$, which belong to f_n^{**} are not influenced by this partial assignment. Hence, y_j is not a term of the RSE of the corresponding subfunction. The assignment where $x_i = 1$ replaces $x_i y_j$ by y_j and has no influence on the other terms $x_{i'} y_j, i' \neq i$, belonging to f_n^{**}. Hence, y_j is a term belonging to the RSE of the corresponding subfunction. This proves that the resulting subfunctions are not equal and f_n^{**} is $(n^{1/2} - 1)$-mixed.

Exercise 6.14. W.l.o.g. $n = 3m$. We claim that HWB_n^* is k-mixed for $k = m-3 = n/3-3$. First we prove a simple number theoretical fact. Let $w_i = 2$, if $i \leq 2m$, and $w_i = 1$ otherwise. If $J \subseteq \{1, \ldots, n\}$ has a size of at least $2m+1$, we can define for each $s \in \{0, \ldots, n-1\}$ the constants $u_j^* \in \{0, 1\}$, $j \in J$, in such a way that the sum of all $w_j u_j^*$, $j \in J$, equals s. If s is odd, we choose some $j \in J$ such that $w_j = 1$ and set $u_j^* = 1$. Since $|J| \geq 2m + 1$, such an index j exists. We are left with the case of even s and an index set J such that $|J| \geq 2m$. The sum of all w_j, $j \in J$, is at least $2m + m = n$. If we assign successively 1 to all u_j^*, $j \in J$, such that $w_j = 2$ and afterwards to all other u_j^*, $j \in J$, we obtain at least all even numbers in $\{0, \ldots, n-1\}$ as partial sums.

In order to prove that HWB_n^* is k-mixed, let I be some set of k variable indices and let u and v be different assignments to the corresponding variables. The goal is to prove that $f_u \neq f_v$. Let $J = \{1, \ldots, n\} - I$ and $\Delta = \sum_{i \in I} w_i v_i - \sum_{i \in I} w_i u_i$:

Case 1: $\Delta \equiv 0 \bmod n$. Let $i^* \in I$ be chosen such that $u_{i^*} \neq v_{i^*}$. By the claim above, we find an input u^* with $u_i^* = u_i$ for $i \in I$ such that $s(u^*) := \sum_i w_i u_i^* \equiv i^* \bmod n$. Let v^* be the corresponding extension of v. Since $\Delta \equiv 0 \bmod n$, we conclude that $s(v^*) \equiv i^* \bmod n$. Hence, $\mathrm{HWB}_n^*(a^*) = u_{i^*}^* \neq v_{i^*}^* = \mathrm{HWB}_n^*(v^*)$.

Case 2: $\Delta \not\equiv 0 \bmod n$. We fix some $j \in J$. Let $l \equiv j + \Delta \bmod n$. Obviously, $l \neq j$. If $l \in J$, we assign the value 0 to x_j and the value 1 to x_l. If $l \notin J$, we assign the value $1 - v_l$ to x_j. Afterwards, we have at least $2m + 1$ free variables. Hence, we can construct an input u^* with $u_i^* = u_i$ for $i \in I$, $u_j^* = 0$, and $u_l^* = 1$, if $l \in J$, and $u_j^* = 1 - v_l$, if $l \in J$, such that $s(u^*) \equiv j \bmod n$. This implies $\mathrm{HWB}_n^*(u^*) = u_j^*$. Let v^* be the corresponding extension of v. Then $s(v^*) \equiv j + \Delta \equiv l \bmod n$. This implies $\mathrm{HWB}_n^*(v^*) = v_l^*$. If $l \in J$, $u_j^* = 0$ and $v_l^* = 1$. If $l \notin J$, $u_j^* = 1 - v_l = 1 - v_l^*$. Hence, in both cases $\mathrm{HWB}_n^*(u^*) \neq \mathrm{HWB}_n^*(v^*)$.

Exercise 6.16. Let $X = (x_{ij})$ be the $n \times n$-matrix of Boolean variables and let V be an arbitrary subset of $n - 1$ variables. We investigate two different assignments a and b to the variables in V. W.l.o.g. $x_{11} \in V, a_{11} = 0$ and $b_{11} = 1$. Let s_i be the number of variables in V belonging to the ith row. After renumbering we can assume that $s_2 \geq \cdots \geq s_n$. In particular, $s_i \leq n - i$ implying that $s_n = 0$. We define a partial assignment c to the variables outside

V such that the input (a, c) leads to the output 1 and the input (b, c) leads to the output 0. Free variables of the first row are replaced by ones. The variables of the last row are replaced by the corresponding entries of the first row. This implies that (a, c) leads to the output 1. We fix the remaining variables rowwise in increasing order in such a way that for (b, c) the ith row, $2 \leq i \leq n-1$, is different from the rows $1, \ldots, i-1, n$. In the row i we have at least i free positions leading to 2^i possible vectors and we have to ensure there the row is different from i given vectors. Since $2^i > i$ for $i \geq 2$, this is possible.

Exercise 6.17. (Solution from the Diploma Thesis of M. Sauerhoff, Univ. Dortmund) We consider the function EAR_n (equal adjacent rows) defined on a Boolean $n \times n$ matrix $X = (x_{i,j})_{1 \leq i,j \leq n}$ by

$$\mathrm{EAR}_n(X) = \bigvee_{1 \leq i \leq n-1} (x_{i,1} = x_{i+1,1}) \wedge \cdots \wedge (x_{i,n} = x_{i+1,n}).$$

We claim that EAR_n has exponential size even for k-OBDDs. Using the technique described in Chapter 7 of the book, we prove a lower bound of order $2^{\Omega(\sqrt{n}/k)}$.

Let G be an arbitrary k-OBDD for EAR_n with variable ordering π. Our aim is to show that G can be used to construct a $(2k-1)$-round communication protocol either for the equality function EQ_m or for the nondisjointness function NDISJ_m for some parameter $m = \Omega(\sqrt{n})$. For input vectors $u = (u_1, \ldots, u_m)$, $v = (v_1, \ldots, v_m) \in \{0,1\}^m$, these functions are defined by $\mathrm{EQ}_m(u, v) = (u_1 = v_1) \wedge \cdots \wedge (u_m = v_m)$ and $\mathrm{NDISJ}_m(u, v) = u_1 v_1 \vee \cdots \vee u_m v_m$.

Let X denote the input matrix as well as the set of all variables of EAR_n. Let A be the set of the first $\lfloor n^2/2 \rfloor$ variables of EAR_n according to the given variable ordering π, and let $B := X - A$. For each choice of the input vectors u and v for EQ_m, we are going to define input assignments a_u to A and b_v to B such that EAR_n applied to the joint assignment (a_u, b_v) to all variables yields the output $\mathrm{EQ}_m(u, v)$. For this, we will choose *special variables* from A and B which will be used to encode the input vectors u and v, resp., of EQ_m. Furthermore, the remaining variables will be set to constants in such a way that the function EAR_n does not become a constant function and still depends on all values of the special variables.

If we have defined the input assignments a_u and b_v in the described way, we can apply the construction from Chapter 7 to obtain a $(2k-1)$-round communication protocol for EQ_m with complexity $(2k-1) \cdot \lceil \log |G| \rceil$. Since the deterministic communication complexity of EQ_m, $C(\mathrm{EQ}_m)$, is equal to m, we get

$$\log |G| \geq C(\mathrm{EQ}_m)/(2k-1) - 1 = m/(2k-1) - 1 = \Omega\left(\sqrt{n}/k\right).$$

This gives the desired lower bound for EAR_n. For certain variable orderings, we alternatively choose NDISJ_m instead of EQ_m in the above construction. The proof works in the same way, since also $C(\mathrm{NDISJ}_m) = m$.

We first describe how the special variables from A and B for the construction of the assignments a_u and b_v are chosen. For this, we have to introduce some notation. Let $C = (c_{ij})_{1 \le i,j \le n}$ be the Boolean matrix defined by $c_{ij} = 1$ if $x_{ij} \in A$, and $c_{ij} = 0$ if $x_{ij} \in B$. For $i = 1, \ldots, n$, define $c_i = (c_{i,1}, \ldots, c_{i,n})$, the ith row of C. Notice that C contains $\lfloor n^2/2 \rfloor$ 1-entries and $\lceil n^2/2 \rceil$ 0-entries. For arbitrary Boolean vectors x, y, let $d(x, y)$ denote the Hamming distance of x and y. Finally, define $\|x\| := d(x, 0)$, i. e., $\|x\|$ is the number of 1-entries in x.

Call a row of C with index i *split* if there are indices j, k such that $c_{i,j} \neq c_{i,k}$. The following lemma is crucial for the choice of the special variables:

Lemma:

(1) There is an index $i_0 \in \{1, \ldots, n-1\}$ such that $d(c_{i_0}, c_{i_0+1}) \ge \lfloor \sqrt{n} \rfloor$, or

(2) there are at least $\lfloor \sqrt{n} \rfloor$ split rows.

Proof of the Lemma: Define $I := \{i \mid 1 \le i \le n, \|c_i\| = 0 \vee \|c_i\| = n\}$. Notice that a row is split exactly if its index is not contained in I. Hence, if $|I| \le n - \lfloor \sqrt{n} \rfloor$, (2) is fulfilled and we are finished.

Let us assume now that $|I| > n - \lfloor \sqrt{n} \rfloor$. Since C contains only around $n^2/2$ 0- and 1-entries each, there are two rows with indices ℓ and h such that $\|c_\ell\| = 0$ and $\|c_h\| = n$ if n is large enough. We assume that $\ell < h$ for the following, the case $\ell > h$ is handled analogously. Furthermore, we assume that ℓ is the largest row index such that $\|c_\ell\| = 0$, and that h is the smallest row index with $\|c_h\| = n$. Then $\ell+1, \ell+2 \ldots, h-1 \notin I$, and thus $|\ell - h| \le n - |I| + 1 < \lfloor \sqrt{n} \rfloor$. We now can conclude that there are two adjacent rows with index between ℓ and h whose Hamming distance is not too small. We have

$$\begin{aligned} \big|\|c_\ell\| - \|c_h\|\big| &= |d(c_\ell, 0) - d(c_h, 0)| \\ &\le d(c_\ell, c_h) \\ &\le d(c_\ell, c_{\ell+1}) + \cdots + d(c_{h-1}, c_h) \\ &\le (h - \ell) \cdot \max_{\ell \le i \le h-1} d(c_i, c_{i+1}). \end{aligned}$$

Thus, there is a row index i_0 with $\ell \le i_0 \le h - 1$ such that $d(c_{i_0}, c_{i_0+1}) \ge n/\lfloor \sqrt{n} \rfloor \ge \lfloor \sqrt{n} \rfloor$, i. e., (1) is fulfilled. □

Now we are ready to construct the desired assignments to the variables of EAR_n.

First, assume that Case (1) in the above lemma occurs, i. e., $d(c_{i_0}, c_{i_0+1}) \ge \lfloor \sqrt{n} \rfloor$ for some row with index i_0. Then we can choose $m := \lfloor \sqrt{n} \rfloor$ columns with index j such that $x_{i_0,j} \in A$ and $x_{i_0+1,j} \in B$ or vice versa. Let $u, v \in \{0,1\}^m$ be an arbitrary pair of input vectors for EQ_m. Assign the value of u_j to the A-variable in the jth of these columns, and the value of v_j to the respective B-variable. For all other columns with index j, assign the same constant to both variables $x_{i_0,j}$ and $x_{i_0+1,j}$ (independent of u and v). Finally, fix all remaining variables such that no adjacent rows are equal. For the assignments a_u to A and b_v to B obtained by this construction, we obviously have $\mathrm{EAR}_n(a_u, b_v) = \mathrm{EQ}_m(u, v)$.

The Case (2) is a little bit more difficult. First, we observe that there are either at least $\lfloor \lfloor \sqrt{n} \rfloor /2 \rfloor$ split rows with even index, or at least as many split rows with odd index. W. l. o. g., we only consider the latter case here. We choose $m := \lfloor \lfloor \sqrt{n} \rfloor /2 \rfloor - 1$ of the split rows whose index i is odd and for which $i+1 \leq n$. Let $R = \{r_1, \ldots, r_m\}$ be the set of indices of these rows.

We now construct assignments a_u to A and b_v to B for given vectors $u, v \in \{0,1\}^m$ such that $\mathrm{EAR}_m(a_u, b_v) = \mathrm{NDISJ}_m(u, v)$. For each row $r_i \in R$, which is a split row, there are columns c_i and d_i such that $x_{r_i,c_i} \in A$ and $x_{r_i,d_i} \in B$. Assign the value u_i to x_{r_i,c_i} and the value v_i to x_{r_i,d_i}. The rest of the variables are fixed independently of u and v. For each row $r_i \in R$, we assign 1 to the variables x_{r_i+1,c_i} and x_{r_i+1,d_i}. Fill the two adjacent rows r_i and r_i+1 with the constant 0, if $r_i \equiv 1 \bmod 4$, and with the constant 1 if $r_i \equiv 3 \bmod 4$. The last remaining rows of unfixed variables not considered so far are alternatingly filled with constants such that no two adjacent rows except for the rows with indices r_i and r_i+1, $i = 1, \ldots, m$, can be equal.

This construction has the following effect. The obtained matrix has a pair of equal adjacent rows if and only there is an $r_i \in R$ such that $u_i = v_i = 1$. Hence, we have $\mathrm{EAR}_n(a_u, b_v) = \mathrm{NDISJ}(u, v)$ for the assignments a_u to A and b_v and B obtained by the construction. □

Exercise 6.18. The claimed space bound is $O(|G|+n|H|)$. The representations of the functions f_v where v is a node of the FBDD G are no longer computed bottom-up but in the order of the last visit of a DFS traversal. For each point of time we distinguish three types of nodes of G. A node is unvisited if it has not been reached by the DFS traversal. It is called finished if the DFS traversal starting at the node is finished. All other nodes are called visited. We have computed π-OBDDs for all f_v where v is finished but we only store those π-OBDDs representing a function f_v such that v has no finished predecessor. For all other finished nodes we only store a pointer to one the finished predecessors. Each finished node without a finished predeccessor has a visited predecessor. The number of visited nodes during a DFS traversal of an FBDD is bounded by n. Each one of them has at most two finished successors (in fact, only the last one can really have two finished successors). Hence, we only store $O(n)$ π-OBDDs each of size bounded above by $|H|$. Whenever we need the π-OBDD for f_v which is no longer stored, we find in $O(n)$ steps a predecessor w such that the π-OBDD for f_w is stored. Applying the appropriate replacements by constants to this π-OBDD we obtain a π-OBDD for f_v. This extra time does not increase the asymptotic run time.

Exercise 6.20. It is easy to guess an input and to verify whether $f(a) = 1$ and $g(a) = 0$. The NP-hardness is proved by the following reduction from SAT. An input $c_1, \ldots, c_m$ of clauses on $x_1, \ldots, x_n$ corresponds to the formula $c_1 \wedge \cdots \wedge c_m$ which efficiently can be transformed into a BP (see Theorem 2.1.3.ii). The jth node labeled by x_i is replaced by an $x_{i,j}$-node. This leads to an OBDD G_1 for some ordering of the $x_{i,j}$-variables, since each variable is the label of only one node. We also can efficiently construct a second OBDD G_2 testing whether all

variables $x_{i,\cdot}$ have the same value. Let f be the function represented by G_1 and let g be the negation of the function represented by G_2. If $f \leq g$, all inputs satisfying f have some pair $(x_{i,j}, x_{i,j'})$ of variables such that $x_{i,j}$ and $x_{i,j'}$ have different values. Hence, there is no satisfying input for $c_1, \ldots, c_m$. If $f \leq g$ is not true, there is an input satisfying f and not satisfying g. Hence, all $x_{i,\cdot}$-variables have the same value and the input for $c_1, \ldots, c_m$ where x_i takes the common value of all $x_{i,\cdot}$-variables satisfies $c_1, \ldots, c_m$.

Exercise 6.21. A pair of nodes $(v, w) \in V_G \times V_H$ is called compatible if the BP with source w is a $G(v)$-FBDD where $G(v)$ is the graph ordering with source v. We like to decide whether (v^*, w^*) for the sources v^* of G and w^* of H is compatible. If v' is the sink of G and w' a sink of H, (v', w') is compatible. Let $(v, w) \in V_G \times V_H$. If label$(v)$ = label(w), (v, w) is compatible iff (v_0, w_0) and (v_1, w_1) are compatible where v_0, v_1, w_0, w_1 are the successors of v and w resp. If label$(v) \neq$ label(w), (v, w) is compatible iff (v_0, w) and (v_1, w) are compatible. This follows, since we never omit a test in G. Hence, the simultaneous DFS traversal through G and H with the above rules for the definition of the successor decides whether H is a G-FBDD. This is the case iff all reachable pairs of nodes are shown to be compatible.

Exercise 6.22. It is obvious that all $*$-sinks can be merged and that one $*$-sink is necessary. Hence, level $n+1$ has a canonical representation obtained by the application of the merging rule. Let us assume that the levels $i+1, \ldots, n+1$ have a canonical representation such that each node on level j describes a graph ordering on $n-j+1$ variables which is part of the given graph ordering and is obtained by a certain assignment a to $j-1$ variables which are tested first for the partial input a. Let us consider a representation of the graph ordering where the levels $i+1, \ldots, n+1$ are reduced. An application of the merging rule does not change the variable ordering. If two nodes on level i cannot be merged, they represent different graph orderings. Either they have different labels or their 0-successors represent, because of the canonicity of the lower part, different graph orderings or the property holds for the 1-successors. If two nodes on level i represent the same graph ordering, they have the same label and, because of the canonicity, the same 0-successors and the same 1-successors. Hence, they can be merged. We have proved that the application of the merging rule leads to the minimal size of the ith level representing all different graph orderings on $n-i+1$ variables which are part of the given graph ordering. The labels of the nodes are canonical and so are their successors.

Exercise 6.23. We start with the reduced representation of G (see Exercise 6.22). If we treat two edges from v to w as one edge, there are as many paths from the source to a $*$-sink as there are different variable orderings. Let $\pi_1, \ldots, \pi_r$ be the different variable orderings. Let p_i be the path describing π_i and let m_i be the monomial describing the inputs following p_i. Then $m_1 + \ldots + m_r = 1$. Let h be the function represented by H and let f be the function represented by I. It is sufficient to check whether $h \wedge m_i = f \wedge m_i$ for all i. We replace H and I with equivalent complete FBDDs H' and I'. Then

it is easy to obtain FBDDs H_i' and I_i' representing $h \wedge m_i$ and $f \wedge m_i$ resp. If $x_j \in m_i$, 0-edges leaving x_j-nodes are redirected to the 0-sink. The same is done for 1-edges leaving x_j-nodes if $\overline{x}_j \in m_i$. The crucial fact is that H_i is an OBDD, since G prescribes for all inputs a where $m_i(a) = 1$ the same variable ordering. We can apply Theorem 6.3.6 to check the equivalence of H_i and I_i. The whole algorithm runs in time $O(r(|I'| + n \cdot |H'|))$ and r is polynomially bounded by assumption.

Exercise 6.24. The size is 3. We only have to represent the function y_n, any other information is contained in the transformation . The claim can be proved in the following way. We start with the complete FBDD for HWB_n constructed in the proof of Theorem 6.1.4. We obtain a τ_G-TBDD by replacing the node on the ith level by y_i. The crucial observation is that all nodes on the nth level represent y_n. This implies that the whole τ_G-TBDD represents y_n.

Exercise 6.25. We consider the FBDD and DT constructed in the proof of Theorem 6.1.3. We relabel all sinks by "$*$" and merge them to one $*$-sink. For each edge from an inner node v to the $*$-sink we add an arbitrary variable ordering of the variables not tested on the path to v. This defines a graph ordering G. For this graph ordering, we obtain in the following way a τ_G-TBDD representing ISA_n. We work with the variables $z_1, \ldots, z_{n+k}$. The FBDD for ISA_n has $2k+1$ levels of inner nodes. They are relabeled levelwise by $z_1, \ldots, z_{2k+1}$. The z_{2k+1}-level only contains nodes representing z_{2k+1}. These nodes can be merged. Let us consider the subtrees whose roots are labelled by z_{k+1}. We number these subtrees from left to right by $T_0, \ldots, T_{n-1}$. The tree T_i has $n = 2^k$ nodes $v_{i,0}, \ldots, v_{i,n-1}$ in distance k from the root. The nodes $v_{i,j}$, $i \leq j \leq i+k-1$ are labelled by constants, all other $v_{i,.}$-nodes represent z_{2k+1}. All nodes in distance d from the root of T_i are roots of a subtree containing a subsequence of 2^{k-d} of the nodes $v_{i,0}, \ldots, v_{i,n-1}$. All but $O(k)$ nodes of T_i are roots of subtrees which represent z_{2k+1}. All these nodes can be merged and the τ_G-FBDD size of ISA_n can be bounded above by $O(n \log n)$.

Exercise 6.26. In order to evaluate a τ_G-TBDD G' on the input $a = (a_1, \ldots, a_n)$ we have to follow the path activated in G' by this input. If we reach a y_i-node of G' we have to switch to the $\tau_i(a)$-successor. For this reason we have to follow the path in G activated by a. If the ith node on this path is labelled by x_j, $\tau_i(a) = a_j$.

Exercise 6.27. We start with the graph ordering G and replace the c-successor of x_i-nodes by the c-sink. Then the inner nodes on level j are relabelled by y_j. If we evaluate this τ_G-FBDD (see the solution of Exercise 6.26), we reach as last inner node a node which has replaced an x_i-node in G. If $a_i = c$, we reach the c-sink. Hence, the τ_G-FBDD represents x_i.

Exercise 6.28. We test the rows in the order $1, \ldots, m$. Whenever a row contains no 1-entry, we reach a sink labelled by the corresponding row. Otherwise,

we store the ones ever seen. Whenever we have seen two ones in the same column, we reach a sink labelled by the corresponding column and the two entries with the ones. We reach a sink after having tested at most $n+1$ rows. Hence, the depth is bounded by $n(n+1)$. The width is bounded by $(n+1)^n$, since it is sufficient to distinguish for each of the n columns whether we have not seen a 1-entry or which of the first n rows contains the only seen 1-entry. Hence, we obtain an OBDD whose size is bounded by $n(n+1)(n+1)^n = 2^{O(n \log n)}$.

Chapter 7

Exercise 7.1. For each BP node v let A_v be an array of length n initialized with zeros. We perform a DFS traversal of G. At the end of the traversal starting from the x_i-node v we consider the arrays A_{v_0} and A_{v_1} for the direct successors v_0 and v_1 of v. We set A_v to the bitwise disjunction of A_{v_0} and A_{v_1}. We stop with the result that G is not an FBDD if A_v has a 1 at position i. Otherwise, we replace the 0 at position i of A_v by 1. We stop with the result that G is an FBDD if we finish the DFS traversal without having obtained the result that G is not an FBDD. The run time of the algorithm is obvious. The correctness follows from the claim that A_v finally contains a 1 at position j iff the FBDD with source v contains at least one x_j-node. This can be proved by induction on the nodes according to the point of time at the end of the corresponding DFS traversal. The claim holds for nodes whose successors are sinks. The induction step is obvious.

Exercise 7.2. The BP G is not an OBDD if it contains an edge from an x_i-node to an x_i-node. Otherwise, we interpret an edge from an x_i-node to an x_j-node as $x_i < x_j$. We obtain at most $2|G|$ such conditions. There are well-known algorithms for topological sorting which decide in time $O(|G|)$ whether the relation "$<$" can be embedded into a complete ordering of all variables which occur as node labels of G. The BP G is an OBDD iff this is possible. In the positive case, we obtain a variable ordering.

Exercise 7.3. We start with a complete binary tree T of depth $O(\log n)$ and, therefore, polynomial size. We test at all nodes of each level the same variable. At each leaf we have to represent a subfunction f' of f on $n - O(\log n)$ variables. We obtain a BP $G_{f'}$ representing f' from the polynomial-size BP G_f representing f by replacement by constants. The size of $G_{f'}$ is not larger than the size of G_f. The total size of the new BP representing f is polynomially bounded, since it contains besides the tree of polynomial size polynomially many BPs each of a size not larger than the size of G_f. Moreover, the $O(\log n)$ variables tested in the tree are tested on each computation path only once.

Exercise 7.4. In the top part we use an OBDD on $x_{1,2}, \ldots, x_{1,n}$ to count the number of edges adjacent to vertex 1. This part has size $O(n^2)$. At the sink reached by all inputs where the degree of vertex 1 equals k we go on in the same way as for the test whether the graph is k-regular (see Theorem 7.2.6). For each k, $O(n^3)$ nodes are sufficient, altogether $O(n^4)$ nodes.

Exercise 7.5. We use n blocks and in the ith block we use some ordering of the variables which describe hyperedges containing the vertex i. This leads to an oblivious k-BP, since each hyperedge of a k-uniform hypergraph contains k vertices. Each block contains $\binom{n-1}{k-1}$ variables and we check whether there are exactly $\lceil \binom{n-1}{k-1}/2 \rceil$ ones among these variables. Such a block has a size bounded above by $\binom{n-1}{k-1}^2$ and the whole size of the k-BP is bounded above by $n\binom{n-1}{k-1}^2 + 2$ which obviously is polynomially bounded in the number $\binom{n}{k}$ of variables.

Exercise 7.6. A k-uniform hypergraph consists of a hyperclique on $\lceil n/2 \rceil$ vertices and $\lfloor n/2 \rfloor$ isolated vertices iff the degree of $\lfloor n/2 \rfloor$ vertices is 0 and the degree of $\lceil n/2 \rceil$ vertices equals $\binom{\lceil n/2 \rceil - 1}{k-1}$. Hence, we use the same labelling of the levels as in Exercise 7.5. For the ith block we check whether the degree of vertex i is 0 or $\binom{\lceil n/2 \rceil - 1}{k-1}$. Moreover, we count the number of isolated vertices and accept an input iff the degree of each vertex has one of the two admissible values and the number of isolated vertices equals $\lfloor n/2 \rfloor$. Altogether, we have at most n^2 modules and the size of each module is bounded above by $\binom{n-1}{k-1}^2$. Hence, the size is polynomially bounded.

Exercise 7.8. Let us consider a variable ordering π_d where the variables are ordered according to hyperplanes in direction d. The function $H_d(X) \bmod q$ can be represented in size $O(N)$ (see the proof of Proposition 7.2.9). Let us investigate π_d-OBDDs for the representation of $H_r(X) \bmod q$ where $r \neq d$. The function $H_r(X) \bmod q$ is the mod q sum of n parity functions. It is sufficient to store for each parity function the partial parity sum of the tested variables. Hence, a width of 2^n and a size of $O(N2^n)$ is sufficient. Moreover, size $O(N^2 2^n)$ is sufficient to compute $H_d(X) \bmod q$ and $H_r(X) \bmod q$ in a π_d-OBDD with q^2 sinks.

Altogether, we design a $(k-1)$-IBDD using the variable orderings $\pi_1, \ldots, \pi_{k-1}$. For CHSP_q^k, we check in the first layer whether $H_1(X) \equiv 0 \bmod q$ and we go on in the same way for the layers $2, \ldots, k-2$. In the last layer we check whether $H_{k-1}(X) \equiv 0 \bmod q$ and $H_k(X) \equiv 0 \bmod q$. The size of this $(k-1)$-IBDD is bounded above by $O(kN + N^2 2^n) = 2^{O(N^{1/k})}$. For HSP_q^k, we obtain the same asymptotic bound with some obvious changes.

Exercise 7.9. We use the school method for multiplication and describe a BP for an arbitrary output bit. We obtain an $n \times 2n$-matrix with the entry $x_i y_j$ in the row i, $0 \leq i \leq n-1$, and column $i+j$ (the columns are numbered from left to right). We partition the matrix to blocks of size $n \times (2\lceil \log n \rceil)$. In each of the at most $\lceil n / \log n \rceil$ layers we compute the sum of the carry and the sum of the n $2\lceil \log n \rceil$-bit numbers of the block. The carry and the sum of the number of a block (including the carry from the last block) are $O(\log n)$-bit numbers and we have to distinguish only polynomially many values. For each block, we design an FBDD. The sum is computed by considering the bits rowwise. Then it is sufficient to test each x_i once and to store always only one x-value. A y_j-variable contributes to at most $2\lceil \log n \rceil$ consecutive rows. Since we can test y_j only once, we have to store it for a while. At each point of time it is sufficient to store $2\lceil \log n \rceil$ y-bits. Altogether, we have to store not more than $O(\log n)$ bits leading to polynomial size. The computation can be stopped if we know the value of of the output bit.

Exercise 7.11. The first part follows similarly to the solution of Exercise 7.9. In order to compute $x_1 w_1 + \cdots + x_n w_n$ we may consider an $n \times (n+1)$-matrix containing rowwise the binary representations of $w_1, \ldots, w_n$. The ones in the row containing w_i are replaced by x_i. Then we have to compute the sum of

these numbers and have to compare this sum with the given threshold value t. After having considered the ith block (see the solution of Exercise 7.9) we know the carry and whether the tail of the binary representation of $x_1w_1 + \cdots + x_nw_n$ is larger than, smaller than, or equal to the tail of the binary representation of t. At the end we know the output of the general threshold function.

We can improve the solution of Exercise 7.9 and can obtain a polynomial-size $\lceil n/\log n \rceil/2$-BP for multiplication. It is only necessary to enlarge the blocks to size $n \times (4\lceil \log n \rceil)$. We obtain a representation of the squaring function by replacing y_i by x_i. Since x_i and y_i are tested on each computation path at most $\lceil n/\log n \rceil/2$ times, the variable x_i is after this replacement tested at most $\lceil n/\log n \rceil$ times.

Exercise 7.12. Let $G(k)$ be the graph consisting of k copies of G where the sink of the ith copy, $1 \leq i \leq k-1$, is replaced by the source of the $(i+1)$th copy. Let H be a k-G-FBDD. With a simultaneous DFS traversal through $G(k)$ and H it is possible to partition H to k layers where the ith layer, $1 \leq i \leq k-1$, is a G-FBDD whose outgoing edges lead to nodes of one of the layers $i+1, \ldots, k$. The kth layer is a G-FBDD.

For the synthesis of k-G-FBDDs H_1 and H_2, we replace x_j-nodes of the ith layer of $G(k)$, H_1, and H_2 by $x_{j,i}$-nodes. Then we obtain a graph ordering G^* on the new set of variables and G^*-FBDDs H_1^* and H_2^*. We perform the synthesis of H_1^* and H_2^* as G^*-FBDDs. Let the result be the G^*-FBDD H^*. In H^* we replace $x_{j,i}$-nodes by x_j-nodes in order to obtain the k-G-FBDD representing the result of the synthesis of H_1 and H_2.

The satisfiability test for a k-G-FBDD H can be performed in a way similar to the case of k-OBDDs (see Theorem 7.3.3). We obtain the time bound $O(|G|^k|H|^{2k-1})$ if we apply the result on the run time for G-FBDD synthesis steps.

Exercise 7.13. It is easy to guess an input a and to verify that the computation path for a contains repeated tests for at least two variables.

For the NP-hardness proof we design a polynomial-time reduction of 3-SAT to our problem. Let $c_1, \ldots, c_m$ be 3-SAT clauses over the variables $x_1, \ldots, x_n$. We consider the $(1,+1)$-BP G constructed in the proof of Theorem 7.3.2. We know that $c_1, \ldots, c_m$ are satisfiable iff G is satisfiable. Let G^* be obtained from G by replacing the 1-sink by a constant size BP H on some y-variables which is not a semantic $(1,+1)$-BP. If $c_1, \ldots, c_m$ are not satisfiable, all computation paths of G^* only run through G and reach the 0-sink. Hence, G^* is a semantic $(1,+1)$-BP. If $c_1, \ldots, c_m$ are satisfiable, there is an assignment to the x-variables leading to the source of H and an assignment to the y-variables such that two y-variables are tested at least twice. Hence, in this case, G^* is not a semantic $(1,+1)$-BP.

Exercise 7.14. The solution of this exercise is due to Detlef Sieling.

The inconsistency test is contained in NP, since it suffices to guess a path in the given branching program and to verify that more than k variables are tested more than once.

We provide a polynomial time reduction from 3-SAT to the inconsistency test. Let (X, C) be an instance for 3-SAT, where $X = \{x_1, \ldots, x_n\}$ is the set of variables and $C = \{C_1, \ldots, C_m\}$ is the set of clauses. Let $p(i)$ be the number of occurrences of x_i (negated and unnegated) in all clauses. We introduce new variables $b_{i,1}, \ldots, b_{i,p(i)}, c_{i,1}, \ldots, c_{i,p(i)}$. If the lth occurrence of x_i is unnegated, we replace it by $b_{i,l}$. If the lth occurrence of x_i is negated, we replace it by $c_{i,l}$. We note that afterwards only one of the variables $b_{i,l}$ and $c_{i,l}$ occurs in the clauses, and it occurs exactly once. Furthermore, we introduce new variables $a_1, \ldots, a_n, d_1, \ldots, d_m, e$. For each variable $x_i \in X$ we construct the component described in Figure 1.

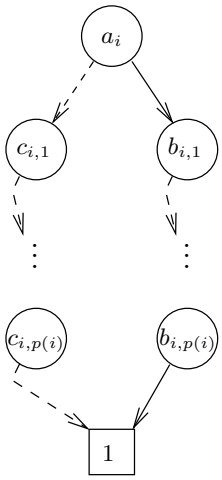


Figure 1

In order to simplify the presentation we use the convention that in the figures omitted edges lead to the 0-sink.

For each clause C_j, e.g., for the clause $C_j = b_{i_1,l_1} \vee b_{i_2,l_2} \vee c_{i_3,l_3}$, we construct the component described in Figure 2.

We note that the 0-sink of this component is in particular reached for those paths on which only variables that are not contained in the clause are tested.

We glue all these components together by replacing the 1-sink of each component by the source of the next component. Finally, we replace the 1-sink of the last component by the branching program described in Figure 3.

We call the constructed branching program P. Let $k = \sum_{i=1}^{n} p(i)$. Then (P, k) is the constructed instance for the inconsistency test. We claim that (X, C) is satisfiable iff there is some path in P on which at least $k+1$ variables are tested repeatedly.

Only-if Part Let $\phi(x_1), \ldots, \phi(x_n)$ be a satisfying assignment for (X, C). We choose a path in P in the following way. In the component for x_i we

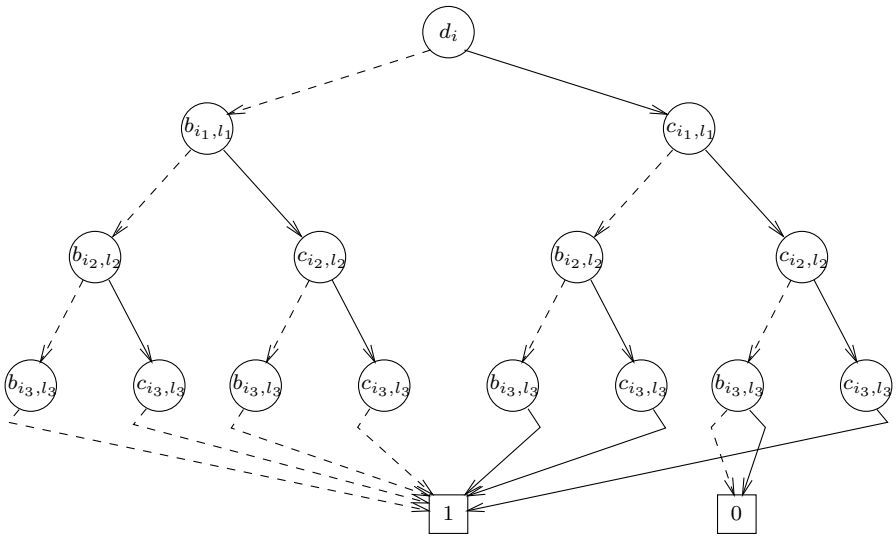


Figure 2

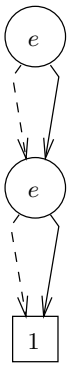


Figure 3

choose the path from the source to the 1-sink via the $b_{i,\cdot}$-tests, if $\phi(x_i) = 1$, and otherwise the path from the source to the 1-sink via the $c_{i,\cdot}$-tests. Now assume that in the clause C_j the literal $b_{i,l}$ occurs. If $\phi(x_i) = 1$ we choose the path through the component for C_j in such a way that a $b_{i,\cdot}$-node is passed. If $\phi(x_i) = 0$, we choose the path in such a way that a $c_{i,\cdot}$-node is passed. Since ϕ is a satisfying assignment, we reach the 1-sink of this component. In the component consisting of the tests of e we choose an arbitrary path to the 1-sink.

On the constructed path the variable e is tested twice. If $\phi(x_i) = 0$, each variable $c_{i,l}$ is tested twice. If $\phi(x_i) = 1$, each variable $b_{i,l}$ is tested twice. Altogether, there are $1 + \sum_{i=1}^{n} p(i) = k + 1$ repeated tests.

If Part Assume that there is a path p in P on which at least $k + 1 = 1 + \sum_{i=1}^{n} p(i)$ variables are tested repeatedly. We note that the a- and d-variables

cannot be tested repeatedly since there is only one node labeled by each of these variables. By the construction of the component for the variable x_i the following holds: If some $c_{i,\cdot}$-variable is tested repeatedly, then each $b_{i,\cdot}$-variable is tested at most once and vice versa. Since for each $b_{i,\cdot}$- and each $c_{i,\cdot}$-variable there are at most two tests, each repeated variable is tested at most twice. We conclude that on p for each i either all $b_{i,\cdot}$- or all $c_{i,\cdot}$-variables are tested repeatedly and that, furthermore, the e-variable is tested repeatedly, because otherwise the number of repeated tests would not exceed k.

We choose $\phi(x_i) = 1$, if on the path p the 1-edge leaving the a_i-node is chosen, and otherwise $\phi(x_i) = 0$. We claim that ϕ is a satisfying assignment for (X, C). Let us assume the contrary, i.e., there is a clause C_j that is not satisfied. W.l.o.g. let $C_j = x_{i_1} \vee x_{i_2} \vee \overline{x}_{i_3}$. Then after the replacement of x-variables by b- and c-variables we have $C_j = b_{i_1,l_1} \vee b_{i_2,l_2} \vee c_{i_3,l_3}$. Since C_j is not satisfied, we have $\phi(x_{i_1}) = 0$, $\phi(x_{i_2}) = 0$ and $\phi(x_{i_3}) = 1$. By the choice of ϕ in the component for x_{i_1} the path on which the $c_{i_1,\cdot}$-variables are tested is chosen. As observed above the $c_{i_1,\cdot}$-variables have to be tested repeatedly. Hence, in the component for C_j the path p runs through a node labeled by c_{i_1,l_1} and it does not run through a node labeled by b_{i_1,l_1}. By the same arguments the path p runs through a node labeled by c_{i_2,l_2} and through a node labeled by b_{i_3,l_3}. But then in the component for C_j the 0-sink is reached, i.e., on p the variable e is not tested repeatedly. Hence, on p at most k variables are tested repeatedly. By this contradiction it follows that all clauses are satisfied.

Exercise 7.15. The consistency tests for s-oblivious BDDs with given s, k-OBDDs with given π, and k-IBBDs with given $\pi_1, \ldots, \pi_k$ can be performed as consistency tests for s-oblivious BDDs G with the corresponding sequence $s = (s_1, \ldots, s_l)$. As described in the proofs of Theorem 7.3.5 and Theorem 7.3.3 we try to partition G into $l + 1$ layers according to s with an additional layer for the sinks. This algorithm is successful iff G is an s-oblivious BDD.

For k-BPs we generalize the solution of Exercise 7.1. Here we define $A_v[j] := \delta(\mathrm{ind}(v), j) + \max\{A_{v_0}[j], A_{v_1}[j]\}$ where $\mathrm{ind}(v)$ is the index of the label of v and $\delta(i, j) = 1$, if $i = j$, and $\delta(i, j) = 0$ otherwise. Then $A_v[j]$ is the largest number of x_j-tests on a path starting at v. The BP is a k-BP if no A_v-entry is larger than k.

Exercise 7.16. We use the given variable ordering $x_0, \ldots, x_{k-1}, y_0, \ldots, y_{k-1}$, $z_0, \ldots, z_{k-1}$. In the top part we have k^3 sinks representing the different values of $(||x|| \bmod k, ||y|| \bmod k, ||z|| \bmod k)$. The width of this part is bounded by k^3 and the size is bounded by $O(nk^3) = O(n^4)$. At each sink we know the three variables whose parity has to be computed. The parity of these three variables is computed according to the given variable ordering in constant size. Hence, the total size of the 2-OBDD is $O(n^4)$. The bottom part has depth 3. Therefore, we have also constructed a $(1, +3)$-BP.

Exercise 7.18. W.l.o.g. n is a multiple of 4. The solution of Exercise 4.18 implies for each variable ordering π that we may fix the z-variables and some x- and y-variables in such a way that we have to represent (after renumbering)

$f = x_1y_1 + \cdots + x_{n/4}y_{n/4}$ with a variable ordering where all x-variables are tested before all y-variables. Hence, it is sufficient to prove an exponential lower bound on the k-OBDD size of this subfunction and variable orderings of the considered type. We claim that the following set S of $2^{n/4}$ inputs is a fooling set if Alice gets the x-variables. Let S contain the vectors $(a_1, \ldots, a_{n/4}, b_1, \ldots, b_{n/4})$ where $b_i = \overline{a}_i$ for all $i \in \{1, \ldots, n/4\}$. Obviously, $f(a, b) = 0$, if $(a, b) \in S$. Let (a', b') and (a'', b'') be different elements from S. Then there exists an index i such that $a'_i = 1$ and $a''_i = 0$ (or vice versa). Then $b''_i = 1$ and $f(a', b'') = 1$. Hence, the communication complexity is bounded below by $n/4$ leading to an exponential lower bound on the k-OBDD size (see Section 7.5).

Exercise 7.19. Theorem 5 of the paper "Hierarchy theorems for k-OBDDs and k-IBDDs" by Bollig, Sauerhoff, Sieling, and Wegener published in Theoretical Computer Science 205, 45-60, 1998, contains the statement that in this situation the size of $(k-1)$-OBDDs is bounded below by $2^{\Omega(n/k)}$ which is non-polynomial if $k = o(n/\log n)$. The proof is presented on the pages 52-53 of that paper.

Exercise 7.20. Here we refer to Theorem 6 (and its proof) of the paper mentioned in the solution of Exercise 7.19 (pages 54-55). We obtain a lower bound of $2^{\Omega(n/k2^k)}$ for $(k-1)$-IBDDs which is non-polynomial if $k \leq (1-\epsilon)\log n$ for some $\epsilon > 0$.

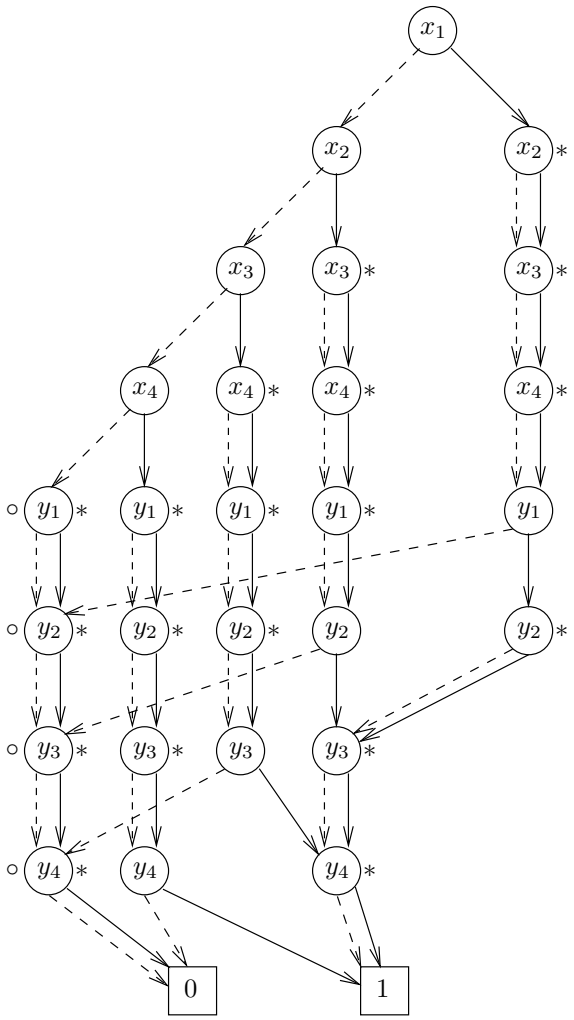
Chapter 8

Exercise 8.1. We have no choice how to define $f(1, a_2, \ldots, a_n)$ but $f(0, a_2, \ldots, a_n)$ may take an arbitrary value independently from all the other values. Hence, the number of functions $f \in B_n$ which are 1-simple with respect to x_1 is equal to $2^{2^{n-1}}$.

Exercise 8.2. It follows from Proposition 8.1.2 that, if $\pi = id$, the partial input $(a_1, \ldots, a_{i-1})$ has to lead in a complete π-ZBDD representing f to an x_i-node representing $\overline{x}_1 \cdots \overline{x}_{i-1} f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$. All these different functions have to be represented at different nodes. However, it is also sufficient to have nodes for these different functions. Then there is no choice how to direct the edges. Hence, complete π-ZBDDs are a canonical representation. If all nodes are reachable from the source, each complete π-ZBDD representing f does not represent other functions than described above. We apply the merging rule levelwise bottom-up. This leads to a unique sink level. If the x_j-levels, $j > i$, are reduced, two x_i-nodes representing the same function can be merged.

Now we consider non-complete π-ZBDDs representing f. If the computation path for a does not contain an x_i-node, the same holds for all inputs b where $b_1 = a_1, \ldots, b_{i-1} = a_{i-1}$. By the evaluation rule, $f(b) = 0$ for all these inputs where additionally $b_i = 1$. Hence, $f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$ and also $\overline{x}_1 \cdots \overline{x}_{i-1} f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$ are 1-simple with respect to x_i. We conclude that all those functions which are not 1-simple with respect to x_i have to be represented at x_i-nodes. Again, it is sufficient to have nodes for these functions and then we have no choice how to direct the edges. A π-ZBDD representing f and containing no nodes not reachable from the source can represent only those functions representable in a complete π-ZBDD of the same kind. The reduction is also performed levelwise bottom-up. If the x_j-levels, $j > i$, are reduced, x_i-nodes representing the same functions can be merged and an x_i-node representing a function which is 1-simple with respect to x_i has the 0-sink as 1-successor and can be eliminated.

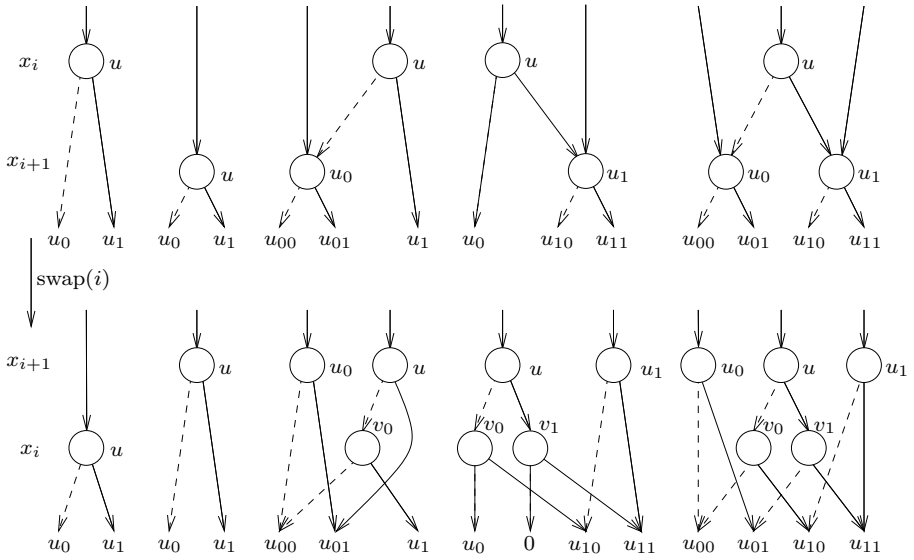
Exercise 8.3. The following figure shows a complete π-OBDD and π-ZBDD representing f_n for $n = 4$. It should be clear how this figure looks for general n. All nodes marked with a "$*$" can be eliminated by the OBDD elimination rule. Hence, the π-OBDD size equals $2n + 2$. Only the nodes marked with a "$\circ$" can be eliminated by the ZBDD elimination rule. The reduced π-ZBDD contains $1 + \cdots + n = \frac{1}{2}n^2 + \frac{1}{2}n$ x-nodes and 2 sinks. There are $n - 1$ y-nodes representing the constant 1 and there are $n + \cdots + 1 = \frac{1}{2}n^2 + \frac{1}{2}n$ further y-nodes. Hence, the π-ZBDD size equals $n^2 + 2n + 1$. For this example, π-ZBDD$(f) \geq n \cdot \pi$-OBDD$(f)/2$ and we get a better trade-off than for MUX_n in Example 8.1.7.



Exercise 8.4. We consider a reduced π-OBDD (w.l.o.g. $\pi = id$) for a symmetric function f. Each edge leaving an x_i-node leads to an x_{i+1}-node or to a sink. Hence, it is sufficient to add at most one x_i-node representing 0 and one x_i-node representing 1 in order to obtain a complete π-OBDD representing f. This implies that the reduced π-OBDD representing f is at most by $2n$ nodes smaller than the quasi-reduced π-OBDD representing f. This quasi-reduced π-OBDD representing f is also the quasi-reduced π-ZBDD representing f. We can eliminate the nodes representing 0 also by the ZBDD elimination rule. If the 1-edge leaving an x_i-node v leads to a node representing 0, the represented function is 1-simple with respect to x_i. Because of the symmetry of f, the 0-edge leads to a sink or to an x_j-node representing a function which is 1-simple with respect to x_j. Hence, the node v can be replaced by the constant

0 or by the constant 1. This implies that the reduced π-ZBDD representing f is at most by $2n$ nodes smaller than the quasi-reduced OBDD or ZBDD representing f. Since the nodes representing 0 can be eliminated in both cases, $|\mathrm{OBDD}(f) - \mathrm{ZBDD}(f)| \leq n$ for symmetric functions $f \in B_n$.

Exercise 8.6. We solve this exercise with a figure similar to Figure 5.7.1.



Exercise 8.7. We obtain the same upper bounds for the swap operation as in the OBDD case (see the solution of Exercise 8.6). Indeed there is one case (the third one) where we save one node in comparison to the OBDD case. Hence, the two bounds in Theorem 5.7.4 and Theorem 5.7.5 which are based on the implementation of a jump operation as sequence of swap operations also hold for the ZBDD case. Also the second bound from Theorem 5.7.4 holds also in the ZBDD case. The arguments can be used word-for-word. Finally, we also can prove the first bound of Theorem 5.7.5 for ZBDDs. We use the same notation. Then $s_k^* = s_k$, if $k \leq i-1$ or $k \geq j+1$. For an x_k-level, $i+1 \leq k \leq j$, we have to consider the functions $\overline{x}_i g_0 + x_i g_1$ which are not 1-simple with respect to x_k, where g_c is a subfunction of $f_{|x_i=c}$ obtained by assigning constants to $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{k-1}$. At least one of the functions g_0 or g_1 has to be not 1-simple with respect to x_k. This leads to the same upper bound on s_k^* as in the proof of Theorem 5.7.5. In order to estimate s_i^* we have to consider the functions $\overline{x}_i g_0 + x_i g_1$ which are not 1-simple with respect to x_i, i.e., g_1 has to be different from the constant 0. This again leads to the same upper bound on s_i^* as in the proof of Theorem 5.7.5.

Exercise 8.8. If G is the quasi-reduced π-ZBDD or π-OBDD representing f, the variable x_i is redundant iff the two edges leaving an arbitrary x_i-node lead to the same node. Here we can restrict ourselves to reduced π-ZBDDs G'. An x_i-node v of G is eliminated iff the function represented at v is 1-simple with respect to v. The only function which is 1-simple with respect to x_i and does not essentially depend on x_i is the constant 0. Hence, f essentially depends on x_i iff some path from the source to a sink reaches an x_i-node with two different successors or crosses the x_i-level and does not reach immediately the 0-sink. This can be checked by a simple DFS traversal.

Exercise 8.9. We consider the function ZMUX_n (zero-suppressed multiplexer), see Example 8.1.7. The size of the reduced π-ZBDD for the variable ordering $x_0, \ldots, x_{k-1}, y_0, \ldots, y_{n-1}$ equals $2n+2$. We like to replace y_0 by 0. We create $n-1$ dummy nodes, one for each y_i-node v_i, $1 \leq i \leq n-1$. The dummy node v_i' for v_i is labelled by y_0, its 0-successor is v_i and its 1-successor is the 0-sink. The only edge leading to v_i is redirected to lead to v_i'. Replacing y_0 by 0 means that the 1-edges leaving y_0-nodes are redirected to lead to the same node as the 0-edge. Only the former y_0-node can be eliminated. Hence, the π-ZBDD size for $\mathrm{ZMUX}_{n|y_0=0}$ and the given variable ordering equals $3n$ which is almost by a factor of $3/2$ larger than the π-ZBDD size for ZMUX_n and the given variable ordering.

Exercise 8.10. We start with π-ZBDDs.
1.) The computation of $f_v(a)$, $a \in \{0,1\}^n$, starts at v and follows the path activated by a. Then $f_v(a) = 1$ iff we reach the 1-sink and $a_i = 1$ for all x_i such that we have not reached an x_i-node.
2.) Each input a activates all a_i-edges leaving x_i-nodes, $1 \leq i \leq n$. Then $f_v(a) = 1$ iff the unique activated path starting at v leads to the 1-sink and $a_i = 1$ for all x_i such that this path does not contain an x_i-node.
3.) The 0-sink represents the constant 0 and the 1-sink represents $\overline{x}_1\overline{x}_2\cdots\overline{x}_n$. An x_i-node v represents $f_0 + x_i f_{1|x_i=0}$ if f_0 and f_1 are represented at the 0-successor resp. 1-successor of v (see Proposition 8.1.2).
Now we consider π-OFDDs.
1.) The computation of $f_v(a)$, $a \in \{0,1\}^n$, starts at v. At x_i-nodes where $a_i = 0$ the 0-edge is chosen. At x_i-nodes where $a_i = 1$ both outgoing edges are chosen. If paths meet again, they are still considered as different paths. Then $f_v(a)$ is equal to the EXOR-sum of the sink labels obtained.
2.) Each input a activates all 0-edges leaving x_i-nodes where $a_i = 0$ and both edges leaving x_i-nodes where $a_i = 1$. Then $f_v(a) = 1$ iff an odd number of activated paths starting at v reaches the 1-sink.
3.) The c-sink represents the constant c. An x_i-node v represents $f_0 \oplus x_i f_1$ if f_0 and f_1 are represented at the 0-successor resp. 1-successor of v.

Exercise 8.11. An FFDD has the same syntax as FBDDs and uses the semantics of OFDDs (see the solution of Exercise 8.10). We claim that a complete FBDD representing f represents τf as FFDD. Since $\tau\tau f = f$ (Lemma 8.2.4), this implies that a complete FFDD representing f represents τf as FBDD. This

implies that the minimal size of complete FBDDs representing f is equal to the minimal size of complete FFDDs representing τf. We still have to prove the claim. Let G be a complete FBDD and let a be an input with j ones. Let G' be the FFDD isomorphic to G. The input a activates 2^j paths in G', since G' is complete. The paths activated by a in G' are the same as the paths activated in G by the 2^j inputs $b \leq a$. Hence, $f_{G'}(a) = \bigoplus_{b \leq a} f_G(b) = \tau f_G(a)$ and the FFDD G' represents τf_G.

Exercise 8.12. The outputs $s_{n-1}, \ldots, s_0$ are of the same type. Hence, we only discuss s_n and s_{n-1}.

The carry s_n is a monotone function. Therefore, $t_n := \tau s_n$ can only take the value 1 if s_n does. Let $(a,b) = (a_{n-1}, \ldots, a_0, b_{n-1}, \ldots, b_0)$ be an input such that $s_n(a,b) = 1$. This implies that there exists some i such that $a_i = b_i = 1$ and $a_j \oplus b_j = 1$ for $j > i$. We count the number of inputs (a',b') such that $(a',b') \leq (a,b)$ and $s_n(a,b) = 1$. If $a_j = 1$ ($b_j = 1$) and $j > i$, it is necessary that $a'_j = 1$ ($b'_j = 1$). If $a'_i = b'_i = 1$, then we may choose $a'_j \leq a_j$ and $b'_j \leq b_j$ for $j < i$ arbitrarily. The number of these inputs is odd iff $a_j = b_j = 0$ for $j < i$. If $a'_i = b'_i = 0$, $s_n(a',b') = 0$. If $a'_i = 1$ and $b'_i = 0$, we obtain the same number of good inputs (a',b') as for the case $a'_i = 0$ and $b'_i = 1$. Hence, the total number for both cases is even. Altogether, $t_n(a,b) = 1$ iff there exists some i such that $a_i = b_i = 1$, $a_j \oplus b_j = 1$ for $j > i$, and $a_j = b_j = 0$ for $j < i$. Let $\pi_1 = (x_{n-1}, y_{n-1}, \ldots, x_0, y_0)$. After having tested y_j we may have reached the 0-sink, we may have seen only pairs with $x_k \oplus y_k = 1$ or we may have seen $x_k \oplus y_k = 1$ for $k > i$, $x_i = y_i = 1$, and $x_k = y_k = 0$ for $j \leq k \leq i-1$ and some i. Hence, we have two x_{j-1}-nodes and three y_{j-1}-nodes if $j - 1 > 0$. On the y_{j-1}-level, we have to distinguish the two situations described above and for the first situation the two possible x_{j-1}-values. If $x_{j-1} = 1$ in the second situation, we reach the 0-sink. Hence, the π_1-OFDD size of s_n equals $5n \pm O(1)$. Let $\pi_2 = (x_0, y_0, \ldots, x_{n-1}, y_{n-1})$. After having tested y_j we may have reached the 0-sink, we may have seen only pairs with $x_k = y_k = 0$ or we may have seen $x_k = y_k = 0$ for $k < i$, $x_i = y_i = 1$, and $x_k \oplus y_k = 1$ for $i + 1 \leq k \leq j$ and some i. Hence, we have two x_{j+1}-nodes and four y_{j+1}-nodes. Here we have to distinguish two situations and on the y_{j+1}-level the possible values of x_{j+1}. The π_2-OFDD size of s_n equals $6n \pm O(1)$.

The sum bit s_{n-1} is not monotone. We distinguish three cases.

Case 1. $a_{n-1} = b_{n-1} = 1$. Let z be the number of inputs (a',b') such that $(a',b') \leq (a,b)$, $a'_{n-1} = b'_{n-1} = 1$, and $s_{n-1}(a',b') = 1$. Then z is also the number of inputs (a',b') such that $(a',b') \leq (a,b)$, $a'_{n-1} = b'_{n-1} = 0$, and $s_{n-1}(a',b') = 1$. Let z' be the number of inputs (a',b') such that $(a',b') \leq (a,b)$, $a'_{n-1} \oplus b'_{n-1} = 1$, and $s_{n-1}(a',b') = 1$. Then z' is even, since we have the cases $(a'_{n-1}, b'_{n-1}) = (1,0)$ and $(a'_{n-1}, b'_{n-1}) = (0,1)$. Hence, the number of inputs $(a',b') \leq (a,b)$ such that $s_{n-1}(a',b') = 1$ is even and $t_{n-1}(a,b) = 0$ for $t_{n-1} := \tau s_{n-1}$.

Case 2. $a_{n-1} = b_{n-1} = 0$. Then s_{n-1} is the carry bit of ADD_{n-1}. We conclude (see above) that $t_{n-1}(a,b) = 1$ iff there exists some i such that $a_i = b_i = 1$, $a_j \oplus b_j = 1$ for $i < j < n-1$ and $a_j = b_j = 0$ for $j < i$.

Case 3. $a_{n-1} \oplus b_{n-1} = 1$. Let r be the number of ones among a_j, b_j, $j < n-1$. Let z be the number of inputs $(a', b') \leq (a, b)$ such that $(a'_{n-1}, b'_{n-1}) = (a_{n-1}, b_{n-1})$ and $s_{n-1}(a', b') = 1$. Then the number of inputs $(a', b') \leq (a, b)$ such that $a'_{n-1} = b'_{n-1} = 0$ and $s_{n-1}(a', b') = 1$ equals $2^r - z$. The reason is that for each of the 2^r choices of assignments to all a_j, b_j, $j \leq n-1$, either $(a'_{n-1}, b'_{n-1}) = (a_{n-1}, b_{n-1})$ or $a'_{n-1} = b'_{n-1} = 0$ leads to the output 1 (but not both). Hence, there are 2^r inputs $(a', b') \leq (a, b)$ such that $s_{n-1}(a', b') = 1$. This implies that $t_{n-1}(a, b) = 1$ iff $a_i = b_i = 0$ for $i < n-1$.

Now we investigate the variable ordering π_1 and π_1-OBDDs for t_{n-1}. If $x_{n-1} = y_{n-1} = 1$, we reach the 0-sink. If $x_{n-1} \oplus y_{n-1} = 1$, we have to check whether $x_i = y_i = 0$ for all $i < n-1$ which leads to one x_i-node and one y_i-node. If $x_{n-1} = y_{n-1} = 0$, we have two x_i-nodes and three y_i-nodes if $i > 0$. Among them there are nodes checking whether the later variables all are 0. Hence, the π_1-OFDD size of s_i, $i < n$ equals $5i \pm O(1)$. Now we investigate the variable ordering π_2. We distinguish the same inputs as for s_n and the last two variables decide about the output. Hence, the π_2-OFDD size of s_i, $i < n$, equals $6i \pm O(1)$.

If we consider shared OFDDs for all outputs, almost nothing can be gained for the variable ordering π_2, since s_i essentially depends on y_i while s_j, $j < i$, does not essentially depend on y_i. Already the description of the subfunctions of t_n and t_{n-1} (and similarly t_i, $i < n-1$) has shown that shared π_1-OFDDs for ADD_n have linear size.

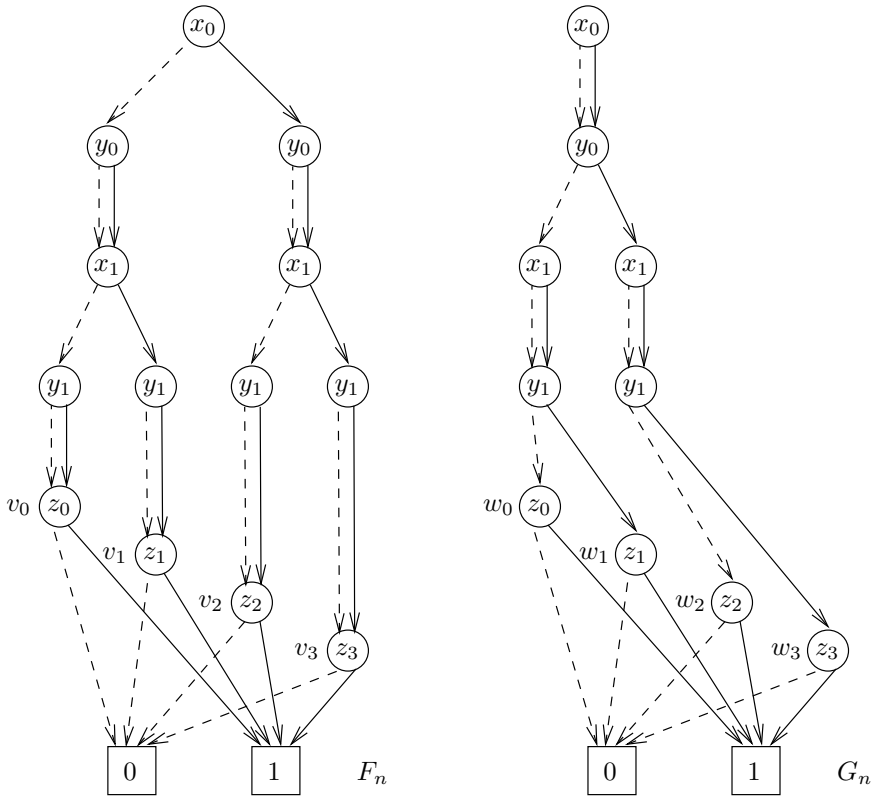
Exercise 8.13. We refer to the construction of the quasi-reduced OBDD G representing $1cl_{n,3}$ which is a quasi-reduced OFDD representing $\oplus cl_{n,3}$. We have shown that the width is bounded by $2N + 3$. It is sufficient to show that only $O(n)$ nodes are left after the application of the OFDD elimination rule. Hence, we investigate G as OBDD and prove that enough nodes have the property that the 1-edge leads to the 0-sink. We claim nothing for the node for the case that we have not found any edge. We have at most N nodes where we have seen one edge. Let us consider the x_{ij}-level. If $x_{ij} = 1$, we reach the 0-sink if the seen edge is not adjacent to i or j. Hence, at most $2n - 4$ of these nodes cannot be eliminated by the OFDD elimination rule. We also have at most N nodes for situations where we have seen two edges. If $x_{ij} = 1$, we reach the 0-sink if the seen edges are not for some vertex k the $\{i, k\}$- and the $\{j, k\}$-edge. Hence, at most $n - 2$ of these nodes cannot be eliminated by the OFDD elimination rule. There are 2 more nodes. Altogether, the width of reduced OFDDs representing $\oplus cl_{n,3}$ is bounded above by $3n - 3$ leading to an $O(nN) = O(N^{3/2}) = O(n^3)$ bound on the OFDD size.

Exercise 8.15. We apply results from Section 10.5. We investigate w.l.o.g. the lexicographical ordering of the inputs $a = (a_1, \ldots, a_n)$ with respect to this ordering of the input bits, i.e., $a \leq_{\mathrm{lex}} b$ iff $a = b$ or $a_i = 0$, $b_i = 1$ and $a_j = b_j$ for $j < i$.

Let G be the given π-OFDD which is w.l.o.g. reduced. Hence, we can decide whether the function f represented by G is the constant 0. In that case there is no satisfying input. Otherwise, we construct an EXOR-π-OBDD G' representing f. This is done by adding for each inner node v an outgoing 1-edge which

reaches the 0-successor of v. We replace x_1 by 1 and check whether $f_{|x_1=1}$ is satisfiable. In the positive case, we look for the lexicographical largest satisfying input for $f_{|x_1=1}$ defined on $x_2, \ldots, x_n$. If the result is $(a_2, \ldots, a_n)$, the final result is $(1, a_2, \ldots, a_n)$. In the negative case, we look for the lexicographical largest satisfying input for $f_{|x_1=0}$ defined on $x_2, \ldots, x_n$. If the result is $(a_2, \ldots, a_n)$, the final result is $(0, a_2, \ldots, a_n)$. Replacement by constants is possible in linear time and does not increase the number of nodes (Theorem 10.5.1.i). The edge size always can be bounded by the square of the node size (if r edges with the same label lead from v to w, they can be replaced by r mod 2 edges). The satisfiability test can be performed by node minimization (Theorem 10.5.5) and the check whether the result is the 0-sink. Altogether, we have to perform n satisfiability tests on EXOR-π-OBDDs with not more nodes than the given OFDD. Hence, the whole algorithm runs in polynomial time.

Exercise 8.16. Let $n = 2^k$ and let the variables be given by their ordering $\pi = (x_0, y_0, \ldots, x_{k-1}, y_{k-1}, z_0, \ldots, z_{n-1})$. We define π-OFDDs F_n and G_n for $n = 4$ by the following figures.



It is obvious how F_n and G_n are defined in the general case. They represent f_n

and g_n resp. The π-OFDD size of f_n and g_n is linear. We apply the synthesis algorithm described in the proof of Theorem 8.2.13 to the π-OFDDs F_n and G_n. There are $n^2/4$ node pairs (v, w) on the y_{k-1}-level which are reachable from the source of the product graph for the $\oplus$-synthesis. They have different node pairs (v_i, w_j) as successors. These successors represent $z_i \oplus z_j$. At most n of these nodes represent the constant 0 leading to mergings or eliminations. Hence, the π-OFDD of $f_n \oplus g_n$ is $\Omega(n^2)$. It is even $\Theta(n^2)$, since the upper bound follows from Theorem 8.2.13.

Exercise 8.17. We know (see the proof of Theorem 8.2.14) that $f_N = \oplus cl_{n,3}$ and $g_N = \overline{T}_{4,N}$ have polynomial π-OFDD size for each variable ordering but $f_N \wedge g_N = 1cl_{n,3}$ has even exponential FFDD size. Let s be a new variable and let $h_N = (s \wedge f_N) + (\overline{s} \wedge g_N)$. If s is the first variable, the 0-successor has to represent g_N and the 1-successor $f_N \oplus g_N$ which both have polynomial π-OFDD size for each ordering of the remaining variables. But $(\forall s)h_N = 1cl_{n,3}$ has exponential FFDD size. Let $h^*_N = s \wedge f_N$. Then $h^*_{N|s=g_N} = 1cl_{n,3}$ and we obtain the claim for the operation replacement by functions.

Exercise 8.18. Yes, such an algorithm exists. Let G and H be the given π-OFDDs. As in the solution of Exercise 8.15 we construct EXOR-π-OBDDs G' and H' of the same node size representing the same functions. The $\wedge$-synthesis of G' and H' leads in polynomial time to an EXOR-π-OBDD F' of size $O(|G'|\cdot|H'|)$ (Theorem 10.5.1.iv). Also the reduction is possible in polynomial time. We still have to transform F' into a π-OFDD F representing the same function f. Let F^* be the quasi-reduced π-OFDD representing f. Then $|F^*| \leq (n+1) \cdot |F|$. It is sufficient if we construct F^* in polynomial time with respect to $|F'|$ and $|F^*|$. W.l.o.g. $\pi = id$ and we construct F^* levelwise top-down. There is one x_1-node. Let us assume that the x_i-level is constructed. Let it contain nodes $v_1, \ldots, v_l$ representing $f_{i,1}, \ldots, f_{i,l}$ and let us assume that we have EXOR-π-OBDDs $F_{i,1}, \ldots, F_{i,l}$ representing $f_{i,1}, \ldots, f_{i,l}$ resp. where $|F_{i,j}| \leq |F'|$. This is in the beginning true for $i = 1$, $l = 1$ and $F_{1,1} = F'$. The successors of v_j have to represent $f_{i,j|x_i=0}$ and $f_{i,j|x_i=0} \oplus f_{i,j|x_i=1}$. The EXOR-$\pi$-OBDDs for $f_{i,j|x_i=0}$ and $f_{i,j|x_i=1}$ can be obtained by replacement by constants. Since x_i is the top level of the considered EXOR-π-OBDDs, this can be done by the operation creation of linear combinations. It is important that we have one x_{i+1}-node representing $f_{i,j|x_i=0}$ and one x_{i+1}-node representing $f_{i,j|x_i=1}$, all other nodes are contained in $F_{i,j}$. Moreover, we can eliminate the x_i-source of $F_{i,j}$. The $\oplus$-synthesis of the two EXOR-π-OBDDs leads to an EXOR-π-OBDD representing $f_{i,j|x_i=0} \oplus f_{i,j|x_i=1}$ whose size is bounded by $|F_{i,j}|$. This operation again is the creation of linear combinations. We obtain EXOR-π-OBDDs representing the $2l$ functions which have to be represented at the successors of $v_1, \ldots, v_l$. We perform pairwise equivalence checks (EXOR synthesis, node minimization (Theorem 10.5.5), and the check whether the result is the 0-sink). Then we know the number of nodes of the x_{i+1}-level of F^* and how to direct the edges between the x_i-level and the x_{i+1}-level. Moreover, we have EXOR-π-OBDDs representing the functions on the x_{i+1}-level, the size of each of them is bounded

by $|F'|$. The number of considered EXOR-π-OBDDs is bounded by $2|F^*|$. Since all operations on EXOR-π-OBDDs are possible in polynomial time, the size of the EXOR-π-OBDDs is bounded by $|F'|$, and the number of operations is bounded by $O(|F^*|^2)$, the whole algorithm runs in polynomial time.

Exercise 8.19. The first step is to add by the inverse elimination rule for the given type of the x_i-level x_i-nodes such that each path from the source to a sink passes through an x_i-node. The set of functions which has to be represented at the x_i-level (and at all levels above the x_i-level) does not change if dt_i is changed. The direct successors of an x_i-node representing g represent

- $g_{|x_i=0}$ and $g_{|x_i=1}$ (if $dt_i = s$),
- $g_{|x_i=0}$ and $g_{|x_i=0} \oplus g_{|x_i=1}$ (if $dt_i = pRM$),
- $g_{|x_i=1}$ and $g_{|x_i=0} \oplus g_{|x_i=1}$ (if $dt_i = nRM$).

Hence, a $\oplus$-synthesis step for each direct successor is sufficient to represent $g_{|x_i=0} \oplus g_{|x_i=1}$, $g_{|x_i=0} = (g_{|x_i=0} \oplus g_{|x_i=1}) \oplus g_{|x_i=1}$, or $g_{|x_i=1} = (g_{|x_i=0} \oplus g_{|x_i=1}) \oplus g_{|x_i=0}$. In all cases the $\oplus$-synthesis on G_1 and G_2 runs in time $O(|G_1| \cdot |G_2|)$. All the resulting nodes are nodes of the product graph of two disjoint copies of the given dt-OKFDD G. Hence, the size of the dt'-OKFDD G' is bounded above by $|G|^2$.

Chapter 9

Exercise 9.1. The proof of Theorem 3.1.4 can be used word-for-word for the generalized case of MTBDDs. The same holds for the proof of Theorem 3.2.2 given as solution of Exercise 3.5.

Exercise 9.2. The terminal cases where v and w are sinks are obvious. We list some additional terminal cases:

1.) Addition. One of the vertices is the 0-sink.

2.) Subtraction. The node w is the 0-sink.

3.) Multiplication. One of the vertices is the 0-sink or the 1-sink.

4.) Min/Max. One of the vertices is the $(+\infty)$-sink or the $(-\infty)$-sink. If we additionally store the smallest and largest possible value, $\max(v) \leq \min(w)$ and $\max(w) \leq \min(v)$ are terminal cases.

Exercise 9.3. Let $n = 2^k$. Let $A_{x,z}$ be a $2^n \times n$-matrix and $B_{z,y}$ be an $n \times 2^n$-matrix and let us use the variable ordering $\pi = (z, x, y)$. The A-matrix contains all 2^n $\{0,1\}$-vectors of length n in the canonical ordering. If we test the z-variables and store the value of $|z|$, we obtain the output as $x_{|z|}$. Hence, $A_{x,z}$ can be realized like a multiplexer in linear size $O(n)$. The B-matrix is the transposed of A and also can be represented in linear size. The C-matrix contains the entries $x_0y_0 + \cdots + x_{n-1}y_{n-1}$. If we test the x-variables before the y-variables, the matrix C has a representation of exponential size.

Exercise 9.4. Let $s_0, \ldots, s_{m-1}$ be the functions describing the bits of f. We construct a π-MTBDD G_i representing $s_i 2^i$ from the π-OBDD representing s_i by replacing the 1-sink by a 2^i-sink. Afterwards, we combine $G_0, \ldots, G_{m-1}$ by synthesis steps realizing integer addition. This causes an exponential blow-up of the size (which cannot be avoided).

Exercise 9.5. An input a activates paths in a BMD in the same way as in OFDDs. The output is the sum of the labels at the sinks reached by the different paths. If we replace each sink by a sink with the negative value, also the output is $-r$ instead of r and we obtain a BMD representation of $-f$ if the given BMD represents f.

Exercise 9.6. We have seen in the proof of Theorem 9.3.2 that π-BMDs representing $c|z| + d$ for constant integers c and d and an n-bit number z only need linear size. Let $|X_i| = x_i 2^i + \cdots + x_0 2^0$. Then it is shown in the proof of Theorem 9.6.2 that

$$|X_{n-1}|^2 = |X_{n-2}|^2 + x_{n-1}(2^{2n-2} + 2^n|X_{n-2}|).$$

Hence, we may start with an x_{n-1}-node whose 0-successor represents $|X_{n-2}|^2$ and whose 1-successor represents with $O(n)$ nodes $2^n|X_{n-2}| + 2^{2n-2}$. Following this approach inductively we obtain a π-BMD of size $O(n^2)$ representing squaring for the ordering $\pi = (x_{n-1}, \ldots, x_0)$.

Exercise 9.7. Each input can activate at most 2^n paths. Hence, each output value is the sum of at most 2^n sink labels (which may be used repeatedly). The input consisting of zeros only activates only one path which has to reach a 1-sink. This sink alone can only lead to outputs up to 2^n. Hence, the second smallest sink label is at most 2^{n+1}. These two sink labels alone can only lead to output values not larger than 2^{2n+1}. Hence, there is a further sink label not larger than 2^{2n+2}, a further one whose label is not larger than $2^{3n+3}, 2^{4n+4}, \ldots$. In order to obtain the output value 2^{2^n-1}, we need a sink label whose size is at least 2^{2^n-1-n}. Hence, we need at least $s+1$ sinks where s is the smallest number such that $sn + s \geq 2^n - 1 - n$. Hence, $s = \Omega(2^n/n)$ and the same lower bound holds for the BMD size.

Exercise 9.8. We describe the six transformation matrices by the following table.

type	$f(0)$	$f(1)$
1	c_0	c_1
2	c_0	$c_1 - c_0$
3	c_0	$c_0 + c_1$
4	c_1	$c_1 - c_0$
5	c_1	$c_0 + c_1$
6	$c_0 + c_1$	$c_1 - c_0$

We do not obtain an essentially different matrix if we interchange the roles of $f(0)$ and $f(1)$ and/or negate some values. For the matrices we may interchange the rows and may negate rows. Hence, each matrix represents 8 matrices which are not essentially different, altogether 48 of the 81 possible matrices. The other 33 matrices are not regular. If the first row is $(0, 0)$, all 9 possibilities for the second row lead to a non-regular matrix. For the other 8 possibilities for the first row, say (a_{11}, a_{12}), we have 3 possibilities to obtain a non-regular matrix namely $(0, 0)$, (a_{11}, a_{12}), and $(-a_{11}, -a_{12})$.

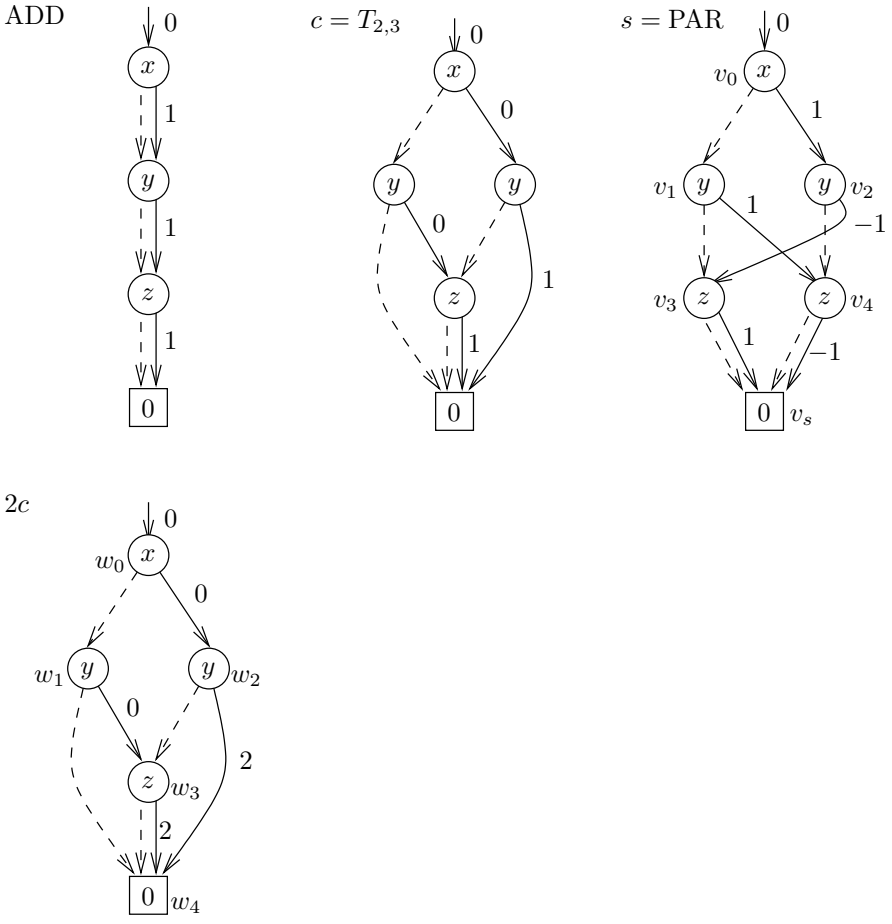
Exercise 9.9. Let f be a function represented at an x_i-node v and let $f_0 = f_{|x_i=0}$ and $f_1 = f_{|x_i=1}$. Depending on the type of the transformation matrix, essentially the following pairs of functions are represented at the direct successors of v. We have chosen computations with integers avoiding rationals (e.g., $f_0 + f_1$ and $f_0 - f_1$ instead of $(f_0 + f_1)/2$ and $(f_0 - f_1)/2$).

type	functions at the successors	
1	f_0	f_1
2	f_0	$f_1 + f_0$
3	f_0	$f_1 - f_0$
4	f_1	$f_0 - f_1$
5	f_1	$f_0 + f_1$
6	$f_0 + f_1$	$f_0 - f_1$

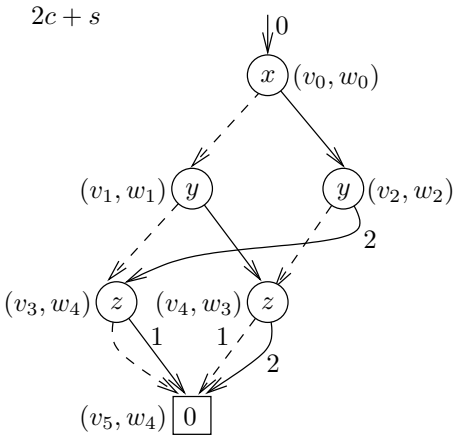
There are simple linear operations to obtain each pair of functions from any other one. These operations have to be performed. Linear operations are supported by each of the representation types.

Exercise 9.10. Let us replace x_i by $c \in \{0,1\}$. We replace all edges leading to x_i-nodes by edges to their c-successors. If $c = 1$, we have to add the weight on the 1-edge leaving the considered x_i-node to the weight of the considered edge to the x_i-node. This procedure is obviously correct, but we may obtain weights on 0-edges. Hence, we have to replace them bottom-up. If w is the weight on the 0-edge leaving v, we replace this weight by 0, add the weight w to all edges leading to v and subtract w from the weight of the 1-edge leaving v.

Exercise 9.11. A simultaneous DFS traversal reaches the following node pairs



with the following "seen weight": $(v_0, w_0, 0), (v_1, w_1, 0), (v_3, w_4, 0), (v_5, w_4, 0)$, $(v_5, w_4, 1), (v_4, w_3, 1), (v_5, w_4, 1), (v_5, w_4, 2), (v_2, w_2, 1), (v_4, w_3, 1)$ already visited, $(v_3, w_4, 2)$ already visited. This leads to the following EVBDD. The 0-edge leav-



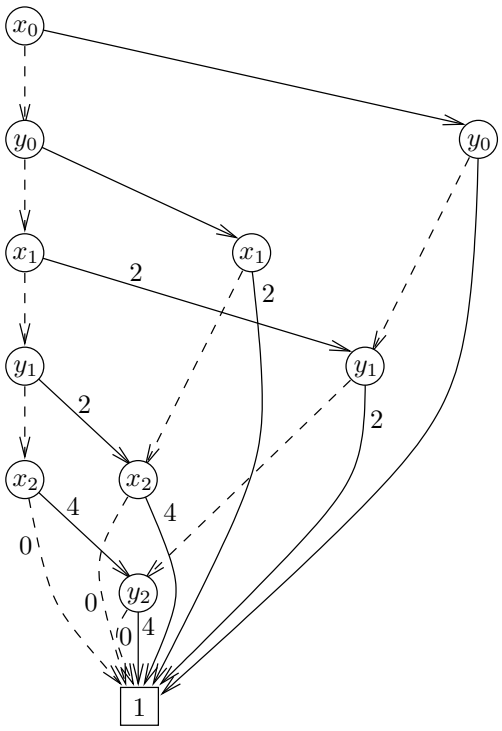
ing (v_4, w_3) has weight 1. This weight is given back to the incoming edges and the weight 1 is subtracted from the weight on the 1-edge leaving (v_4, w_3) leading to the weight 1. Hence, (v_3, w_4) and (v_4, w_3) are merged. The 0-edge leaving (v_2, w_2) has the weight 1. This weight is given back to the incoming edge and the 1-edge leaving (v_2, w_2) gets the new weight 1. Hence, (v_1, w_1) and (v_2, w_2) can be merged leading to the canonical π-EVBDD representing $2c + s$ which is isomorphic to the π-EVBDD representing ADD.

Exercise 9.12. Let π be an arbitrary variable ordering and let x_k be the last variable according to this ordering. The 2^{n-1} assignments to the other $n-1$ variables lead to 2^{n-1} different numbers a. Such an assignment leads to the subfunction $(a + x_k 2^k)^2 = a^2 + x_k(2a2^k + 2^{2k})$. If $a \neq a'$, the difference of the subfunctions equals $2(a - a')2^k x_k + a^2 - a'^2$ and is not a constant. Hence, we obtain a lower bound of 2^{n-1} from Theorem 9.5.3.

Exercise 9.13. We may use the same ideas as for the solution of Exercise 9.12. Here the subfunctions are $2^{(a+x_k 2^k)}$ and the difference for a and a' equals $2^{x_k 2^k}(2^a - 2^{a'})$ which is not a constant if $a \neq a'$. Again we obtain a lower bound of 2^{n-1} from Theorem 9.5.3.

Exercise 9.14. We construct a $*$BMD G_1 representing $|y|$. This $*$BMD has one y_i-node whose 1-edge leads with the weight 2^i to the unique sink which is a 1-sink. The 0-edge has weight 1 and leads to the y_{i+1}-node if $i < n-1$. If $i = n-1$, it leads with weight 0 to the sink. We construct another $*$BMD G_2 representing $|x|$ (we really do not need the x_0-node). Except these nodes we create one x_i-node and one y_i-node. The 0-edge leaving the x_i-node has weight 1 and reaches the y_i-node and the 0-edge leaving the y_i-node has weight 1 and reaches the x_{i+1}-node, if $i < n-1$, and it has weight 0 and reaches the sink, if $i = n-1$. The 1-edge leaving the x_i-node reaches the y_i-node of G_1 and has weight 2^i. The 1-edge leaving the y_i-node reaches the x_{i+1}-node of G_2 and has weight 2^i, if $i < n-1$. If $i = n-1$, it has weight 0 and reaches the

1-sink. This y_{n-1}-node indeed can be eliminated leading to a $*$BMD size of $n + (n-1) + 2n - 1 + 1 = 4n - 1$. The case $n = 3$ is shown in the following figure where the weights 1 are not described explicitly. This $*$BMD realizes the



multiplication in the following way:

$$\begin{aligned} |x| \cdot |y| \quad = \quad & x_0(y_0 2^0 + \cdots + y_{n-1} 2^{n-1}) \\ & + y_0(x_1 2^1 + \cdots + x_{n-1} 2^{n-1}) \\ & + 2x_1(y_1 2^1 + \cdots + y_{n-1} 2^{n-1}) \\ & + 2y_1(x_2 2^2 + \cdots + x_{n-1} 2^{n-1}) \\ & + \cdots \\ & + 2^{n-1} x_{n-1} \cdot 2^{n-1} y_{n-1}. \end{aligned}$$

Exercise 9.15. W.l.o.g. $\pi = id$. We prove the claim by induction on n. For $n = 1$ we have one of the functions $ax_1 + b$, $a, b \in \mathbb{R}$. This function can be represented by an x_1-node whose edges lead to a 1-sink. The weight on the 0-edge equals b and the weight on the 1-edge equals a. If $a = b = 0$, we can eliminate the node and replace it by an edge with weight 0 to the sink. If $a = 0$ and $b \neq 0$, we introduce an edge with weight b to the x_1-node whose outgoing 0-edge gets the weight 1. A similar procedure is possible if $a \neq 0$ and $b = 0$.

If $a \neq 0$ and $b \neq 0$ there is only one way to fulfil the restrictions. The missing common weight is given to an edge leading to the x_1-node.

For the induction step, we also argue with shared π-$*$BMDs. We have a canonical representation of $f_{|x_1=0}$ and $f_{|x_1=1} - f_{|x_1=0}$. We introduce an x_1-node v which is reached by an edge with weight 1. The 0-edge leaving v is identified with the edge representing $f_{|x_1=0}$ and the 1-edge is identified with the edge representing $f_{|x_1=1} - f_{|x_1=0}$. If an edge has weight 0, it points to the sink. If the 1-edge has the weight 0, we can eliminate v and obtain by induction hypothesis a canonical representation. If the 0-edge has the weight 0, there is only one way to replace the weight on the 1-edge by 1. The weight on the edge representing f is multiplied by this weight. There is also a unique way to fulfil the restrictions in the other cases. There is no possibility to change the weights in the sub-π-$*$BMDs. Otherwise, the representation of $f_{|x_1=0}$ and $f_{|x_1=1} - f_{|x_1=0}$ would not be canonical.

Exercise 9.16. The solution of this exercise is due to Enders (1995).

We only discuss complete $*$BMDs which also are canonical. The elimination rule eliminates nodes whose 0-edge has the weight 1 and whose 1-edge has the weight 0 (and, therefore, leads to the sink). This allows the application of the reverse elimination rule. Complete $*$BMDs G on n variables have 2^n paths and, similarly to OFDDs and BMDs, we have to consider the BDD computation paths for all $b \leq a$ in order to evaluate G on input a. More precisely, the contribution of a path is the product of all the edge weights on the path and the value of G for a is the sum of the contributions of the computation paths for all $b \leq a$. We may use another interpretation of a $*$BMD. The Shannon type $*$BMD evaluation rule considers for the input a only the path activated by a, namely the path from the source to the sink containing a_i-edges leaving x_i-nodes and the output is the contribution of this path. Let f be the function represented by G as $*$BMD and let f^* be the function represented by G using the Shannon type $*$BMD evaluation rule. We define the function τ^* on the set of functions $g : \{0,1\}^n \to \mathbb{R}$ by $\tau^*(f^*) = f$. Then we can generalize Lemma 8.2.4.i. By definition

$$\tau^*(g)(a) = \sum_{b \leq a} g(b)$$

and

$$\tau^*(g_1 + g_2)(a) = \sum_{b \leq a} (g_1(b) + g_2(b)) = \sum_{b \leq a} g_1(b) + \sum_{b \leq a} g_2(b) = \tau^*(g_1) + \tau^*(g_2).$$

Moreover, τ^* is one-to-one. Let b be the smallest input (w.r.t. to $\leq$) such that $g_1(b) \neq g_2(b)$. Then $\tau^*(g_1)(b) \neq \tau^*(g_2)(b)$.

Hence, we can solve the problem by investigating $*$BMDs with the Shannon type $*$BMD evaluation rule, S$*$BMDs for short. It is sufficient to prove that addition may lead to an exponential blow-up of the size of S$*$BMDs.

Let $f(a) = 1$ for all $a \in \{0,1\}^n$. The size of complete S$*$BMDs for f obviously is linear. Let $p_1, \ldots, p_n$ be the n smallest primes and let

$g(a) = \prod_{1 \le i \le n} p_i^{a_i}$. Also the size of complete S∗BMDs for g is linear. We need one x_i-node whose outgoing 0-edge has the weight 1 and whose outgoing 1-edge has the weight p_i and both edges lead to the x_{i+1}-node, if $i < n$, and to the sink, if $i = n$. Let $h = f + g$, i.e.,

$$h(a) = \prod_{1 \le i \le n} p_i^{a_i} + 1.$$

Let $a' = (a'_1, \ldots, a'_{n-1})$ and $b' = (b'_1, \ldots, b'_{n-1})$ be two different partial inputs. If the partial computation paths for a' and b' lead to the same x_n-node with weight m_0 on the 0-edge and weight m_1 on the 1-edge, then we can argue as follows. Let $w(a')$ be the contribution of the a'-path, similarly $w(b')$. Then w.l.o.g. $m_0 = 1$ and

$$\begin{aligned}
h(a', 0) = w(a') &= \prod_{1 \le i \le n-1} p_i^{a'_i} + 1, \\
h(a', 1) = w(a')m_1 &= \prod_{1 \le i \le n-1} p_i^{a'_i} p_n + 1, \\
h(b', 0) = w(b') &= \prod_{1 \le i \le n-1} p_i^{b'_i} + 1, \text{and} \\
h(b', 1) = w(b')m_1 &= \prod_{1 \le i \le n-1} p_i^{b'_i} p_n + 1.
\end{aligned}$$

Hence, $h(a', 0)/h(a', 1) = h(b', 0)/h(b', 1)$ and for $c = \prod_{1 \le i \le n-1} p_i^{a'_i}$ and $d = \prod_{1 \le i \le n-1} p_i^{bi'}$ we get $c \neq d$ (since $a' \neq b'$ and $p_1, \ldots, p_{n-1}$ are different primes) and

$$\frac{c+1}{cp_n+1} = \frac{d+1}{dp_n+1}$$

which is equivalent to

$$cdp_n + dp_1 + c + 1 = cdp_n + cp_n + d + 1$$

or

$$d(p_n - 1) = c(p_n - 1)$$

and

$$c = d.$$

This contradiction proves the existence of 2^{n-1} x_n-nodes.

Exercise 9.17. Let v be an x_i-node in a $*$BMD G representing $f = w_0 f_0 + x_i w_1 f_1$ where f_0 and f_1 are the functions represented at the 0-successor resp. 1-successor. If we replace x_i by $wx_i + c$, we have to represent

$$f' = w_0 f_0 + (wx_i + c) w_1 f_1 = w_0 f_0 + c w_1 f_1 + x_i w w_1 f_1.$$

We construct a new node v_1' which has the same label, the same weights on the outgoing edges and the same direct successors as v_1, the 1-successor of v. The weight on the edge from v to v_1 is replaced by ww_1 and we create an edge to the node v_1' with weight cw_1. Then we apply a synthesis algorithm for the addition of the functions represented at the 0-edge leaving v ($w_0 f_0$) and at the edge to v_1' ($cw_1 f_1$). The resulting edge is the new 0-edge leaving v. This is correct, since it represents $w_0 f_0 + cw_1 f_1$. The same procedure is applied to all x_i-nodes. Afterwards, the usual procedure to obtain a canonical representation has to be applied.

Exercise 9.18. The procedure presented as solution of Exercise 9.17 shows that the affine replacement of the variable can be performed by s_i applications of a synthesis algorithm for addition where s_i is the number of x_i-nodes . Hence, the size blow-up can be bounded knowing the size blow-up for addition. Let the source of a $*$BMD be labeled by x_i. Let $w = 0$ and $c = 1$, i.e., x_i is replaced by the constant 1. The new function equals $w_0 f_0 + w_1 f_1$ and is the addition of the functions represented at the edges leaving the source. Hence, each size blow-up caused by an addition can be caused also by the affine replacement of a variable. Hence, the result is contained in the solution of Exercise 9.16.

Chapter 10

Exercise 10.1. The basis $\{\text{NOT}, \text{OR}\}$ is polynomially related to $\{\text{AND}, \text{OR}\}$. An AND-node can be replaced by an OR-node where all incoming and outgoing edges get an additional NOT-gate. By de Morgan laws we obtain the same functions. Since even $\{\text{AND}, \text{OR}\}$-OBDDs are polynomially related to Boolean circuits (Proposition 10.1.2), the basis $\{\text{NOT}, \text{OR}\}$ cannot be more powerful.

The basis $\{\rightarrow\}$ also is polynomially equivalent to $\{\text{AND}, \text{OR}\}$. It is sufficient (see the first part of the solution of this exercise) to show that we can simulate NOT-and OR-nodes. In general $f \rightarrow g = \overline{f} + fg$. Therefore, $f \rightarrow 0 = \overline{f}$. To simulate an OR-node representing $f + g$, we use a NOT-node to represent $\overline{f}$. Then

$$\overline{f} \rightarrow g = f + \overline{f}g = f + g.$$

Exercise 10.2. Let T be an $\{\text{AND,OR}\}$-decision tree with l leaves labeled by constants and r inner nodes labeled by variables. Then we obtain a B_2-formula with $2r + l$ leaves. This follows easily by induction on the number of inner nodes, since an AND- or OR-node with successors representing f_0 and f_1 can be simulated by $f_0 \wedge f_1$ and $f_0 + f_1$ resp. and an x_i-node can be simulated by $(\overline{x}_i \wedge f_0) + (x_i \wedge f_1)$.

A B_2-formula can be simulated by an $\{\text{AND}, \text{OR}, \text{NOT}\}$-formula whose size is only polynomially larger (see Wegener (1987)). Applying de Morgan laws starting at the root of the formula, we obtain a formula with the same number of leaves where $\{\text{NOT}\}$-nodes only have leaves as predecessor. If we allow leaves labeled by literals, we get an $\{\text{AND}, \text{OR}\}$-formula. Now it is sufficient to replace the leaves which only can represent a constant or a positive or a negative literal by a decision tree with one inner node.

Exercise 10.3. The formula size of the majority function MAJ_n is polynomial (see L.G.Valiant (1984). Short monotone formulae for the majority function. Journal of Algorithms 5,363-366). The function MAJ_n has $\binom{n}{\lceil n/2 \rceil}$ prime implicants of length $\lceil n/2 \rceil$ and $\binom{n}{\lfloor n/2 \rfloor + 1}$ prime clauses of length $\lfloor n/2 \rfloor + 1$. Since MAJ_n is monotone, we can conclude that each DNF for MAJ_n contains at least $\binom{n}{\lceil n/2 \rceil}$ monomials and each CNF for MAJ_n contains at least $\binom{n}{\lfloor n/2 \rfloor + 1}$ clauses. Hence, by Theorem 10.1.4 and Theorem 10.1.5, MAJ_n has exponential OR-DT size and AND-DT size. For EXOR-DTs, we also apply Theorem 10.1.5. An EXORNF representing MAJ_n with s monomials leads to a circuit for MAJ_n which has depth 2 if we allow unbounded fan-in AND- and EXOR-gates. The first level contains s AND-gates realizing the monomials (we assume that the literals are given for free). The second level contains one EXOR-gate combining the s monomials. Razborov has shown in 1986, that unbounded fan-in $\{\text{AND}, \text{EXOR}\}$-circuits realizing MAJ_n in constant depth need exponential size (for a proof of the result see R. Smolensky (1987). Algebraic methods in the theory of lower bounds for Boolean circuit complexity. 19.STOC, 77-82).

Exercise 10.6. The function EQ_n^* is defined in Definition 10.3.5. We describe

EQ_n^* as conjunction of the following n^2 functions f_{ij}, $1 \leq i, j \leq n$. Let

$$f_{ij}(a,x,b,y) = 1 \text{ iff } x_i = y_j \text{ or } \neg(a_i = 1, b_j = 1,$$
$$\sum_{1 \leq k \leq i} a_k = \sum_{1 \leq k \leq j} b_k, \sum_{1 \leq k \leq n} a_k = \sum_{1 \leq k \leq n} b_k).$$

If $EQ_n^*(a,x,b,y) = 0$, $a_1 + \cdots + a_n \neq b_1 + \cdots + b_n$ or there is some pair (i,j) such that $a_1 + \cdots + a_i = b_1 + \cdots + b_j$, $a_i = 1$, $b_j = 1$ but $x_i \neq y_j$. Then $f_{ij}(a,x,b,y) = 0$. If $EQ_n^*(a,x,b,y) = 1$, we can conclude that $a_1 + \cdots + a_n = b_1 + \cdots + b_n$ and for each pair (i,j) that $a_1 + \cdots + a_i = b_1 + \cdots + b_j$, $a_i = 1$, $b_j = 1$ implies $x_i = y_j$ and $f_{ij}(a,x,b,y) = 1$. Let π be an arbitrary variable ordering. In order to realize f_{ij} it is sufficient to store the values of x_i, y_j, a_i, b_j, the partial sums of all tested a_k and of all tested a_k where $1 \leq k \leq i$, and the partial sums of all tested b_k and of all tested b_k where $1 \leq k \leq j$. Hence, a width of $16(n+1)^4$ is sufficient which implies a π-OBDD size of $O(n^5)$ for f_{ij} and an AND-π-OBDD size of $O(n^7)$ for EQ_n^*.

Exercise 10.7. The function IP_n^* is defined in Definition 10.3.5. We describe IP_n^* as EXOR sum of the following n^2 functions g_{ij}, $1 \leq i, j \leq n$. Let

$$g_{ij}(a,x,b,y) = 1 \text{ iff } x_i = y_j = 1, a_i = 1, b_j = 1,$$
$$\sum_{1 \leq k \leq i} a_k = \sum_{1 \leq k \leq j} b_k, \text{and } \sum_{1 \leq k \leq n} a_k = \sum_{1 \leq k \leq n} b_k.$$

If $IP_n^*(a,x,b,y) = 1$, the number of pairs (i,j) such that $g_{ij}(a,x,b,y) = 1$ is odd and vice versa. The estimation of the π-OBDD size of g_{ij} is similar to the estimation of the π-OBDD size of f_{ij} in the solution of Exercise 10.6.

Exercise 10.8. First, we investigate f_n. Let $M_0, \ldots, M_{k-1}$ denote the k $s \times s$-matrices and let g_j, $0 \leq j \leq k-1$, be defined on M_j. The function g_j takes the value 1 iff at least $\lceil s/2 \rceil$ rows contain at least $\lceil s/2 \rceil$ ones each.

Our first claim is a lower bound of size $2^{\lceil s/2 \rceil^2 - 1}$ on the FBDD size of f_n. It is sufficient to prove that f_n is $(\lceil s/2 \rceil^2 - 1)$-mixed. After having tested $\lceil s/2 \rceil^2 - 1$ variables, we have not seen $\lceil s/2 \rceil$ variables in each of $\lceil s/2 \rceil$ rows of some matrix. Hence, for two different assignments a and b to $\lceil s/2 \rceil^2 - 1$ variables where $a_i \neq b_i$, and $i = (i_{k-1}, \ldots, i_0)$ we can assign values to all further variables such that $g_i(a^*) = g_i(b^*) = i_j$ for the common extensions of a and b resp. Then $f_n(a^*) = a_i \neq b_i = f_n(b^*)$.

The OR-OBDD representing f_n starts with an OR-node with n outgoing edges. The ith edge, $i = (i_{k-1}, \ldots, i_0)$, corresponds to the guess that $g_j(M_j) = i_j$ and $x_i = 1$. We have to verify that this guess is correct. We test the variables in the order $M_0, \ldots, M_{k-1}$ and the variables in each matrix rowwise. The value of g_j can be computed in size $O(s^4)$. Hence, the OR-OBDD size of f_n is bounded by $O(s^4kn)$. The same holds for the OR-OBDD size of $\overline{f}_n$ and, therefore, for the AND-OBDD size of f_n. The only difference is that we check whether $x_i = 0$.

In order to obtain circuit representations we use the same representation as for the OR-OBDD representation namely

$$f_n(x) = \bigvee_{i=(i_{k-1},\ldots,i_0)\in\{0,1\}^k} x_i \wedge \bigwedge_{0\le j\le k-1} g_j(M_j)^{i_j}$$

Here, we use the notation that $a^1 = a$ and $a^0 = \overline{a}$.

Each function on s variables and, therefore, also MAJ_s and $\overline{\mathrm{MAJ}}_s$ can be represented by DNFs and CNFs of size $2^{O(s)}$. Since g_j is a majority of majorities, we can combine a CNF and a DNF to obtain an OR-AND-AND-OR- and also an OR-AND-OR-circuit of size $2^{O(s)}$ representing $g_j(M_j)^{i_j}$. The crucial fact is that the following conjunction of all $g_j(M_j)^{i_j}$ and x_i only has fan-in $k+1$. Using the law of distributivity we replace the last two levels of type OR-AND by an AND-OR representation whose size can be bounded by $2^{O(sk)}$. Hence, we obtain for f_n an OR-AND-AND-OR-OR representation of size $2^{O(sk)}$ which can be replaced by an OR-AND-OR representation by merging neighbored levels of the same type. The same approach works for $\overline{f}_n$ leading to an AND-OR-AND representation of size $2^{O(sk)}$ for f_n.

Let f_n^* be defined on $N = \lceil 2^{(n\log n)^{1/2}} \rceil$ variables and let f_n^* realize f_n on the first n variables. All the complexity results for f_n also hold for f_n^*. We have to consider the bounds with respect to the number of variables N. Since $s = \lfloor (n/\log n)^{1/2} \rfloor$, $s^2 = \Omega(n/\log n)$, and the FBDD lower bound is $2^{\Omega(n/\log n)} = N^{\Omega(n^{1/2}/\log^{3/2} n)} = N^{\Omega(\log N/(\log\log N)^{O(1)})}$ which grows faster than any polynomial in N. The polynomial upper bounds on the OR-OBDD and AND-OBDD size of f_n lead to polylogarithmic upper bounds for f_n^*. The upper bounds for depth-3 circuits are $2^{O(sk)} = 2^{O((n\log n)^{1/2})}$ and, therefore, polynomial upper bounds with respect to N.

Exercise 10.9. We start with an AND-node with k outgoing edges where the dth edge leads to an OBDD G_d representing $H_d(X)$, $1 \le d \le k$. The OBDD G_d uses the variable ordering π_d where the groups of variables with the same value of i_d are tested blockwise. The parity of all $x_{i_1,\ldots,i_n}$ where $i_d = i$ can be computed easily and the number $(\bmod q)$ of parities with value 1 is stored. The size of the AND-FBDD is $O(kqN) = O(kN)$, since q is a constant.

Exercise 10.10. We use the same approach as in the proof of Theorem 10.3.6 but we do not assume that k is a constant. Then we obtain $m = \lfloor n/2^{2k} \rfloor$ x-variables and m y-variables such that the number of layers with respect to these variables is bounded above by $kn+1$. We give the chosen m x-variables to Alice, the chosen y-variables to Bob, and replace the a- and b-variables in such a way by constants that we have to realize the function EQ_m resp. IP_m on the chosen variables. A nondeterministic oblivious BDD G for EQ_n^* and IP_n^* implies the existance of a nondeterministic protocol of length $(4k-1)\lceil \log|G| \rceil$ for EQ_m or IP_m resp. In those cases where the nondeterministic communication complexity is bounded by $\Omega(m)$ (see Theorem 10.3.9 and the following remarks) we can conclude that

$$(4k+1)\log|G| = \Omega(n/2^{2k})$$

or

$$|G| = 2^{\Omega(n/2^{2k})-2\log k-1}.$$

This is exponential as long as $k \leq (\frac{1}{2} - \varepsilon)\log n$ and grows superpolynomially if $k = \frac{1}{2}\log n - \omega(\log\log n)$. Hence, we obtain non-polynomial lower bounds if $k \leq \frac{1}{2}\log n - \omega(\log\log n)$.

Exercise 10.11. We claim that the OR-FBDD size of excl_n is bounded below by $n^{-2}\binom{n}{\lceil n/2\rceil} = 2^{n-O(\log n)} = 2^{\Omega(N^{1/2})}$. We can almost copy the proof of Theorem 6.2.6, since that proof argues only with inputs $a \in \text{excl}_n^{-1}(1)$ and *the* computation path for a. For OR-OBDDs, we fix for each $a \in \text{excl}_n^{-1}(1)$ one path from the source to the 1-sink activated by a. Then the proof of Theorem 6.2.6 works.

Exercise 10.13. For HWB_n, we define $w_i, 0 \leq i \leq n$, by $w_i(x) = 1$ iff $x_1 + \cdots + x_n = i$. We choose an arbitrary variable ordering. The window functions are symmetric and the OBDD size of w_i is $O(n^2)$. The function $f_i = \text{HWB}_n \wedge w_i = x_i w_i$ computes 1 iff $x_1 + \cdots + x_n = i$ and $x_i = 1$. Hence, the OBDD size of f_i equals $O(n^2)$.

For ISA_n, we define $w_{ij}, 0 \leq i, j \leq n-1$, by $w_{ij}(x, y) = 1$ iff $(y_{k-1}, \ldots, y_0)$ is the binary representation of j and $(x_j, \ldots, x_{j+k-1})$ (the indices are taken mod n) is the binary representation of i. Then w_{ij} is a monomial on $2k$ variables and its π-OBDD size equals $O(k)$ for arbitrary π. Moreover, $f_{ij} = \text{ISA}_n \wedge w_{ij} = x_i \wedge w_{ij}$ and also has π-OBDD size $O(k)$.

For WS_n, we define $w_i, 2 \leq i \leq n$, by $w_i(x) = 1$ iff the sum of all $jx_j \bmod p$ equals i and we define $w_1(x) = 1$ iff the sum of all $jx_j \bmod p \notin \{2, \ldots, n\}$. The width of π-OBDDs representing w_i is bounded by p and, therefore, the size is bounded by $O(np) = O(n^2)$. Moreover, $f_i = \text{WS}_n \wedge w_i = x_i \wedge w_i$ and its π-OBDD size also is bounded by $O(n^2)$.

For $\text{PJ}_{k,n}$, we work with n^{2k+1} window functions, one for each of the possible paths of length $2k+1$. The window functions are monomials (indeed some are the constant 0 and can be omitted) and have a π-OBDD size of $O(k\log n) = O(\log n)$. The corresponding part has to check whether the window function has the value 1 and whether the color of the last vertex of the path equals 1.

Exercise 10.14. Let $w_i, 0 \leq i \leq n-1$, iff $(g_{k-1}(M_{k-1}), \ldots, g_0(M_0))$ is the binary representation of i. We choose a variable ordering π whose first variables of M_{k-1} are tested rowwise, then the variables of M_{k-2}, and so on. We have seen in the solution of Exercise 10.7 that the π-OBDD size of w_i is bounded by $O(s^4 k)$. Moreover, $f_n \wedge w_i = x_i \wedge w_i$ also can be represented in size $O(s^4 k)$.

Exercise 10.15. The (k, w, π)-PBDDs start with an OR-node leading to π_j-OBDDs $G_j, 1 \leq j \leq k$, representing w_j (if we like to represent the constant 1) or $x_i \wedge w_j$ (if we like to represent the variable x_i). If the window functions are quite complicated, the simple functions used in the beginning of a synthesis process based on a circuit representation have already complicated representations. The constant 0 is always represented by OBDDs representing 0.

Exercise 10.16. It is not known how to generalize the technique of the proof of Theorem 6.2.13. A lower bound proof is contained on the pages 497–499 of the paper B.Bollig and I.Wegener (1999). Complexity theoretical results on partitioned (nondeterministic) binary decision diagrams. Theory of Computing Systems 32, 487–503.

Exercise 10.17. In Lemma 10.4.10, it is shown that $P_{k,n}$ can be represented in size $O(2^k k^3 n^k)$ by PBDDs with k parts. The FBDD and the 2-OBDD start with a complete binary tree of the s-variables. The FBDD may continue, if $|s| = i$, as the PBDD in the ith part after the test whether $|s| = i$. Hence, the same upper bound as for the k-PBDDs holds. The 2-OBDD uses the following variable ordering: s-variables, z-variables, $x_{0,\cdot,\cdot}$-variables, $x_{1,\cdot,\cdot}$-variables, ..., $x_{k-1,\cdot,\cdot}$-variables, c-variables. The FBDD constructed above can be interpreted as 2-OBDD. The s-variables are only tested in the beginning. The ith part uses the variable ordering $(z, x_i, x_{i+1}, \ldots, x_{k-1}, x_0, \ldots, x_{i-1}, c)$ where we use the obvious abbreviations. The test of $(z, x_i, x_{i+1}, \ldots, x_{k-1})$ belongs to the top OBDD and the test of $(x_0, \ldots, x_{i-1}, c)$ to the bottom OBDD.

Each 2-OBDD G (and even k^*-OBDD, if $k^* = O(1)$) of polynomial size can be simulated by a polynomial-size EXOR-OBDD. We apply the approach of the satisfiability test for k^*-OBDDs (see Theorem 7.3.3). We obtain at most $|G|^{k^*-1}$ OBDDs with the same variable ordering as G, each of size $O(|G|^{k^*})$ such that the function f represented by G is the disjunction of the $|G|^{k^*-1}$ functions represented by the constructed OBDDs. Morover, at most one of these functions can compute 1 for a given input a. Hence, we obtain an EXOR-OBDD for f starting with an EXOR-node leading to the at most $|G|^{k^*-1}$ OBDDs. The total size is bounded by $O(|G|^{2k^*-1})$. In our case, $k^* = 2$ and we obtain the size bound $O((2^k k^3 n^k)^3) = O(2^{3k} k^9 n^{3k})$. All the bounds are polynomial for constant k.

Exercise 10.18. The solution of this exercise is contained in that part of the solution of Exercise 10.17 where EXOR-OBDDs are considered.

Exercise 10.19. It is mentioned in Section 8.2 that the SAT-COUNT problem is #P-complete for OFDDs. This result has been proved by Werchner, Harich, Drechsler, and Becker (1995). An OFDD G can be transformed in linear time to an EXOR-OBDD representing the same function f as G. Hence, the SAT-COUNT problem for EXOR-OBDDs is #P-hard. The problem is even #P-complete. To prove that it is contained in #P we guess an input a and compute $f(a)$ in polynomial time by evaluating G on a. We accept iff $f(a) = 1$. This nondeterministic Turing machine runs in time $O(|G| + n)$ and has $|f^{-1}(1)|$ accepting computation paths.

Exercise 10.20. W.l.o.g. $\pi = id$. The algorithm applies the node minimization procedure (see Theorem 10.5.5) and constructs the node minimal EXOR-π-OBDD G' which represents the same function f as the given EXOR-π-OBDD G. Then we check whether G' contains an x_i-node and claim that f essentially depends on x_i iff G' contains an x_i-node. The run time of the

algorithm is dominated by the time $O(n \cdot |V(G)|^3)$ for the node minimization. The correctness follows from Theorem 10.5.4. We have to prove that $\dim(V_{f,i}) = \dim(V_{f,i+1})$. We remember that $V_{f,k}$ is the vector space spanned by the subfunctions $f_{|x_1=a_1,\ldots,x_m=a_m}$ where $k-1 \leq m \leq n$. Hence, comparing $V_{f,i}$ and $V_{f,i+1}$ it follows that $V_{f,i+1} \subseteq V_{f,i}$ and that for $V_{f,i}$ we have additionally to consider the functions $f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$. If f does not essentially depend on x_i, $f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1}} = f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1},x_i=0}$ and these functions are also considered for $V_{f,i+1}$. Hence, $V_{f,i+1} = V_{f,i}$. Since $V_{f,i+1} = V_{f,i}$, the node minimal EXOR-π-OBDD G' representing f does not contain an x_i-node implying that f does not essentially depend on x_i.

Exercise 10.21. We consider a random variable ordering and give the first n variables to Alice and the other n variables to Bob. It has been shown in the proof of Theorem 5.3.3 that with a probability exponentially close to 1 there are at least $0.4n$ singletons, i.e., indices i such that Alice gets x_i and Bob gets y_i or vice versa. If j is not a singleton, we fix $x_j = y_j = 0$. Then we are left with a function EQ_k where $k \geq 0.4n$ with high probability. Theorem 10.3.4 implies that the EXOR-communication complexity is in these cases bounded below by k. This leads to a lower bound of 2^k on the EXOR-π-OBDD size. For variable orderings π' like $x_1, y_1, \ldots, x_n, y_n$ even the π'-OBDD size of EQ_n is linear. Hence, EQ_n is almost ugly for EXOR-OBDDs.

Chapter 11

Exercise 11.1. By Lemma 11.1.2, it is sufficient to consider complete $\mathcal{F}$-FBDDs. Then for $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$, $I_c(b) = \{i \mid b_i = c\}$

$$\begin{aligned}(f \wedge g)_{\mathcal{F}}(x,y) &= \sum_{a \in f^{-1}(1), b \in g^{-1}(1)} \prod_{i \in I_0(a)} (1-x_i) \prod_{i \in I_0(b)} (1-y_i) \prod_{i \in I_1(a)} x_i \prod_{i \in I_1(b)} y_i \\ &= \Big(\sum_{a \in f^{-1}(1)} \prod_{i \in I_0(a)} (1-x_i) \prod_{i \in I_1(a)} x_i \Big) \cdot \Big(\sum_{b \in g^{-1}(1)} \prod_{i \in I_0(b)} (1-y_i) \prod_{i \in I_1(b)} y_i \Big) \\ &= f_{\mathcal{F}}(x) \cdot g_{\mathcal{F}}(y).\end{aligned}$$

The signature for random inputs is a random variable. The expected signature for $f \wedge g$ is the product of the expected signatures of f and g, since the choices of x and y are independent and $E(X \cdot Y) = E(X) \cdot E(Y)$ for independent random variables.

Exercise 11.2. W.l.o.g. $\pi = id$. We remember that, by our definition, each probabilistic variable is read on each computation path at most once. Let $z_1, \ldots, z_r$ be the probabilistic variables. For each z_j-node v let $i(v)$ be the largest i such that a path from the source to v passes through an x_i-node. If no path from the source to v contains a decision node, $i(v) = 0$. Let B_i be the set of all probabilistic nodes v where $i(v) = i$. We consider the subgraph of G on the node set B_i. Let $B_{i,1}$ be the set of sources and let $B_{i,j}$ be the set of nodes v such that the longest path from $B_{i,1}$ to v has the length $j-1$. By the read-once property of the probabilistic variables, $B_{i,j} \neq \emptyset$ only if $j \leq r$. The new probabilistic variables are denoted by $z_{i,j}$, $0 \leq i \leq n, 1 \leq j \leq r$. A probabilistic node v where $i = i(v)$ and $v \in B_{i,j}$ is relabeled by $z_{i,j}$. The new randomized OBDD G' is graph theoretically isomorphic to G and G' is ordered with respect to $z_{0,1}, \ldots, z_{0,r}, x_1, z_{1,1}, \ldots, z_{1,r}, x_2, z_{2,1}, \ldots, z_{2,r}, \ldots, x_{n-1}, z_{n-1,1}, \ldots, z_{n-1,r}$, $x_n, z_{n,1}, \ldots, z_{n,r}$. The probability to choose a computation path with k probabilistic nodes equals 2^{-k} for G and G', since the probabilistic variables are read once and, therefore, the random decisions are independent. This leads to the same acceptance and the same rejection probability.

Exercise 11.3. The new model is a generalization of the old one. If not all inner nodes have the same number of outgoing edges, we have to measure the size by the number of edges. A randomized node with 2^k outgoing edges can be simulated by a complete binary randomized tree of depth k with $2^{k+1} - 1$ edges. This does not have any real consequences. But we may have randomized nodes with r outgoing edges where r is not a power of 2, e.g., $r = 3$. Such nodes cannot be simulated exactly in the old model, since there it is impossible to obtain probabilities like $1/3$. Using randomized binary trees of depth k we can guarantee an error probability of 2^{-k} compared with the given node. This procedure needs trees of non-polynomial size if we perform it for each randomized node in order to guarantee a small increase of the error probability. With the same proof we can prove the generalization of Theorem 11.4.2 to the

new model. This leads to a polynomial increase of the size and an increase by an additive constant of the error probability. Hence, the new model has more freedom but, if we allow a small increase of the error probability, we cannot represent more functions in polynomial size. Small increase means $\epsilon(n)$ where $\epsilon(n)^{-1}$ is polynomially bounded. Hence, there may be differences only for the PP-model. Proposition 11.5.11 contains the result MS $\in$ BPP$_{1/3+\epsilon(n)}$-FBDD as long as $\log(\epsilon(n)^{-1})$ is polynomially bounded. A careful analysis shows that MS $\in$ BPP$_{1/3}$-FBDD if we are allowed to start with a randomized node with three outgoing edges.

Exercise 11.4. We claim that Theorem 11.3.5 holds for randomized s-oblivious BDDs G_n and not only for randomized π-OBDDs. If the randomized nodes are labeled by different probabilistic variables and different copies of G_n get different probabilistic variables, we obtain s'-oblivious BDDs where s' also contains all probabilistic variables. The proof of Theorem 11.3.5 is based only on the fact that π-OBDDs and π-MTBDDs allow efficient synthesis algorithms. The synthesis algorithm for s-oblivious BDDs (see Theorem 7.3.5) is a π-OBDD synthesis algorithm and can be generalized to s-oblivious MTBDDs. Hence, the claim follows in the same way as Theorem 11.3.5.

Exercise 11.5. Theorem 11.8.7 contains the result that probability amplification is not possible for FBDDs. In particular, MS $\in$ coRP$_{1/2}$-FBDD and MS $\in$ BPP$_{1/3+\epsilon}$-FBDD but MS $\notin$ coRP$_{1/3-\delta}$-FBDD and MS $\notin$ BPP$_{1/4-\delta}$-FBDD. The FBDD G' proving that MS $\in$ coRP$_{1/2}$-FBDD has a single randomized node at the source and the 0-edge leads to an OBDD with a rowwise variable ordering π_0 while the 1-edge leads to an OBDD with a columnwise variable ordering π_1. It is easy to obtain a graph ordering G_1 for one probabilistic variable y_1 and all variables of MS$_n$ such that the considered randomized FBDD G$'$ is a G_1-FBDD. Let G'' be a copy of G' with the new probabilistic variable y_2 instead of y_1. Then G'' is a G_2-FBDD where G_2 is the copy of G_1 where y_1 is replaced by y_2. We need a graph ordering G on y_1, y_2, and the variables of MS$_n$ such that G' and G'' are G-FBDDs. Then we have to start with y_1 and y_2. The most general case is a complete binary tree. If $y_1 = y_2 = 0$, the variables of MS$_n$ can be ordered according to π_0 and, if $y_1 = y_2 = 1$, according to π_1. If $y_1 = 0$ and $y_2 = 1$ (or vice versa), there is no graph ordering such that G' and G'' are "represented". We need a rowwise ordering for G' and a columnwise ordering for G''. Hence, the two copies cannot be considered as G-FBDDs for the same FBDD. This is different to π-OBDDs or s-oblivious BDDs where the different copies of the same probabilistic variable can be arranged in an arbitrary order. If the copies have different values, this does not cause difficulties, since the remaining variables are always ordered in the same way.

Exercise 11.7. We use the variable ordering $a_{k-1}, \ldots, a_0, x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}$ and start with a complete binary tree on the a-variables. If the vector a is known, we have to perform an equality test of x and $y(a)$. It is shown in Proposition 11.5.1 that the equality test of two vectors is contained in coRP$_{\epsilon(n)}$-π-OBDD for each variable ordering π as long as $\epsilon(n)^{-1}$ is polynomially bounded.

Hence, the chosen variable ordering is appropriate for the equality test of x and $y(a)$. Altogether, $\mathrm{SEQ}_n \in \mathrm{coRP}_{\epsilon(n)}$-OBDD as long as $\epsilon(n)^{-1}$ is polynomially bounded. Here the variable ordering cannot be chosen arbitrarily but it is sufficient that all a-variables are tested in the beginning. The size of the randomized OBDD is $O(n^4\epsilon(n)^{-2}\log n)$ (compare Proposition 11.5.1).

Exercise 11.8. We use the fingerprinting technique and choose s as the smallest power of 2 which is at least $n^2\epsilon(n)^{-1}$. We start with a complete binary tree of depth $\log s$ consisting of randomized nodes only. This is interpreted as the random choice of one of the s smallest primes. Let p be the chosen prime. We use a rowwise variable ordering starting with the first row, followed by the second row and so on. Let the row vectors be denoted by $x_1, \ldots, x_n$. First, we compute $|x_1| \bmod p$ and then $|x_2| \bmod p$. If $|x_1| \equiv |x_2| \bmod p$, we accept. Otherwise, we forget $|x_1| \bmod p$ and compute $|x_3| \bmod p$. If $|x_2| \equiv |x_3| \bmod p$, we accept. Otherwise, we forget $|x_2| \bmod p$ and compute $|x_4| \bmod p$ and so on. If we have not accepted after the computation of $|x_n| \bmod p$, we reject. The width of each sub-OBDD is bounded by $p^2 = O(n^4\epsilon(n)^{-2}\log^2 n)$, the depth is bounded by n^2 and we have $O(n^2\epsilon(n)^{-1})$ sub-OBDDs leading to a total size of $O(n^8\epsilon(n)^{-3}\log^2 n)$ which is polynomially bounded, since $\epsilon(n)^{-1}$ is polynomially bounded. If $\mathrm{EAR}_n(X) = 1$, we have $|x_i| = |x_{i+1}|$ for some i. These inputs are accepted by all sub-OBDDs. If $\mathrm{EAR}_n(X) = 0$, we have $|x_i| \neq |x_{i+1}|$, $1 \leq i \leq n-1$, and for each i there are at most n primes p such that $|x_i| \equiv |x_{i+1}| \bmod p$. Hence, the number of primes p leading to $|x_i| \equiv |x_{i+1}| \bmod p$ for some i is smaller than n^2. Hence, the error probability is bounded above by $n^2/(n^2\epsilon(n)^{-1}) = \epsilon(n)$.

Exercise 11.9. We claim that the matrix storage access function MSA_n is $(s-1)$-mixed leading to a lower bound of $2^{s-1} = 2^{\Omega((n/\log n)^{1/2})}$ for the FBDD size. Let b and c be different assignments to the same set of $s-1$ variables. Let $b_i \neq c_i$. For each matrix M_j, at most $s-1$ variables are tested. Hence, there is no row where all variables are known to be equal to 1 and there is a row without tested variable. Let $a_j = \mathrm{ROW}_s(M_j)$. We can choose an assignment to the remaining $n-(s-1)$ variables which ensures that $|a| = i$. Then $\mathrm{MSA}_n(b^*) = b_i \neq c_i = \mathrm{MSA}_n(c^*)$ for the common extensions b^* and c^* of b and c resp.

For an OR-OBDD we choose a variable ordering where the matrices are tested blockwise and each matrix rowwise. We start with a nondeterministic node with n outgoing edges. The ith edge leads to an OBDD checking whether $|a| = i$ and $x_i = 1$. This is possible with linear size $O(n^2)$. It is obvious that this is a nondeterministic representation of MSA_n. If we check whether $|a| = i$ and $x_i = 0$, we obtain a polynomial-size OR-OBDD representing $\overline{\mathrm{MSA}_n}$. This leads to a polynomial-size AND-OBDD representing MSA_n.

Exercise 11.10. Let $l = \lceil \log k \rceil + 1$. The FBDD starts with a randomized node leading to G_1 and G_2.

In G_1, we want to compute the correct output if $x_{|a|}$ is contained in $M_0, \ldots, M_{k-l-1}$ or belongs to the remaining variables. Otherwise, we like to output "?".

We read the variables of $M_{k-l}, \ldots, M_{k-1}$ blockwise and the variables of each matrix rowwise. Polynomial size is sufficient to compute $(a_{k-1}, \ldots, a_{k-l})$. Now $2^{k-l} = 2^{k-\lceil \log k \rceil - 1} \leq n/k \leq s^2$. Hence, we know an index i such that $x_{|a|}$ is in M_i or in M_{i+1} or $x_{|a|}$ is one of the remaining variables. We read the variables in $M_0, \ldots, M_{k-l-1}$ except M_i and/or M_{i+1} if they are among them. This is done similarly to the reading of $M_{k-l}, \ldots, M_{k-1}$. Then we know all a-bits except at most two. Hence, $|a|$ may take at most four values. We read M_i and/or M_{i+1} in order to obtain the full information on $|a|$. Moreover, we store the value of the at most four variables which describe the output. If $x_{|a|}$ is contained in $M_0, \ldots, M_{k-l-1}$, we can reach the correct sink. If $x_{|a|}$ is a remaining variable, we can test it to reach the correct sink. Otherwise, we reach the ?-sink.

In G_2, we read the variables of $M_0, \ldots, M_{k-l-1}$, blockwise and the variables of each matrix rowwise. Polynomial size is sufficient to compute $(a_{k-l-1}, \ldots, a_0)$. The number of missing a-bits equals $l = \lceil \log k \rceil + 1$. The number of possible $|a|$-values is bounded above by $4 \log n$. We read the matrices $M_{k-l}, \ldots, M_{k-1}$ in order to obtain the value of $|a|$. Moreover, we store the value of the at most $4 \log n$ variables which may determine the output. This is possible in polynomial size. If $x_{|a|}$ belongs to $M_{k-l}, \ldots, M_{k-1}$, we can reach the correct sink. If $x_{|a|}$ is a remaining variable, we may test it to reach the correct sink. Otherwise, we reach the ?-sink.

This FBDD has polynomial size, zero error and the failure probability is bounded by $1/2$.

Exercise 11.11. HWB $\in$ NP-OBDD (Theorem 10.2.1) and NP-OBDD $\subseteq$ PP-OBDD (see the remark before Theorem 11.3.4). We also present a direct proof. We may use an arbitrary variable ordering. Let m be the smallest power of 2 such that $m \geq n$. Then $m < 2n$. We start with a complete binary tree of randomized nodes and depth $\log m$. In the ith OBDD, $1 \leq i \leq n$, we compute $s = x_1 + \cdots + x_n$ and store x_i. If $s \neq i$, we reach a randomized node whose c-successor is the c-sink. If $s = i$, we reach the x_i-sink. Altogether, with probability $\frac{m-1}{m}$ we reach a sub-OBDD where the error probability is bounded by $1/2$. With probability $\frac{1}{m}$ we reach a sub-OBDD where the error probability is 0. Hence, the error probability is bounded by $\frac{m-1}{2m} = \frac{1}{2} - \frac{1}{2m} < \frac{1}{2} - \frac{1}{4n}$.

Exercise 11.12. Let G be a randomized OBDD. We may guess an input $a \in \{0,1\}^n$. Then (see Proposition 11.2.4) we can compute in $O(|G|)$arithmetic steps $\mathrm{acc}_G(a)$ and $\mathrm{rej}_G(a)$ and can verify whether both numbers are smaller than $3/4$.

Let $c_1, \ldots, c_m$ be a set of clauses defined on the variables $x_1, \ldots, x_n$. Since we may add trivial clauses like $x_i + \overline{x}_i$, we can assume w.l.o.g. that $m - 1 = 2^k$ for an integer k. We start with a complete binary tree of depth $k+2$ consisting of randomized nodes only. This tree has $4m-4$ leaves, $3m-4$ are replaced by the 0-sink and the ith remaining leaf is replaced by an id-OBDD G_i realizing c_i. The construction of this randomized OBDD G is possible in linear time. By definition, for each $a \in \{0,1\}^n$

$$1 - \mathrm{rej}_G(a) = \mathrm{acc}_G(a) \leq \frac{m}{4m-4} < \frac{3}{4} \quad (\text{if} \quad m \geq 2).$$

Moreover, $\text{rej}_G(a) < \frac{3}{4}$ iff all G_i accept a and this is equivalent to the property that a is a satisfying input.

Exercise 11.13. Let G_1 be a randomized k-IBDD and let G_2 be a randomized s-oblivious BDD both representing $g \in B_n$ with some given error bound and error type. Let X_n be the set of variables. With the same arguments as in Section 7.5 we obtain the following results:

- For each partition (A, B) of X_n such that $|A|, |B| \geq \lfloor n/2 \rfloor$ we can construct sets A' and B' such that $A' \subseteq A, B' \subseteq B; |A'|, |B'| \geq \lfloor n/2^{k+1} \rfloor$ and the number of layers of G_1 with respect to A' and B' is bounded by $2k$.

- Let the length of s be bounded by kn and let A and B be disjoint sets each containing $\lfloor n/4 \rfloor$ variables which all are labels of at most $2k$ levels. Then there exist sets $A' \subseteq A$ and $B' \subseteq B$ such that $|A'|, |B'| \geq \lfloor n/2^{4k-1} \rfloor$ and the number of layers of G_2 with respect to A' and B' is bounded by $4k + 1$.

Let g' be a subfunction of g on $A' \cup B'$ and let the variables of A' be given to Alice and the variables of B' be given to Bob. Then G_1 leads to a randomized $2k$-round randomized protocol for g' of length $2k\lceil \log |G_1| \rceil$ with the same error bound and error type as G_1. The same holds for G_2 and $(4k+1)$-round protocols of length $(4k + 1)\lceil \log |G_2| \rceil$. Lower bounds on the randomized communication complexity of g' lead to lower bounds on the size of randomized k-IBDDs and s-oblivious BDDs representing g' or g. Moreover, if we can construct a rectangular reduction from f to g (with the given partition of the variables), lower bounds for the randomized communication complexity of f also hold for the randomized communication complexity of g.

Exercise 11.14. Let π be a variable ordering on the variables $x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}$ of ISA_n and let G be a randomized π-OBDD representing ISA_n with two-sided ϵ-bounded error where $\epsilon < 1/2$ is a constant.

We choose s as the largest power of 2 which is smaller than $\lfloor n/k \rfloor = \lfloor n/\log n \rfloor$. Let L be the set of the first s x-variables according to π and let R be the set of the remaining x-variables. Then there exists a block $x_j, \ldots, x_{j+k-1}$ of x-variables which all belong to R. We fix the y-variables such that $|y| = j$. This leads to a randomized π-OBDD G' representing $g = \text{ISA}_{n||y|=j}$ with two sided ϵ-bounded error and it is sufficient to prove a lower bound on G'.

We construct a rectangular reduction from INDEX_s to $g(L, R)$. This leads by Theorem 11.7.1.i and our lower bound technique for randomized OBDDs to a lower bound of $2^{\Omega(n/\log n)}$ for G'. The function $\phi_A : \{0,1\}^s \to \{0,1\}^s$ is the identity function. The function $\phi_B : \{0,1\}^{\log s} \to \{0,1\}^{n-s}$ is defined in the following way. Let $b \in \{0,1\}^{\log s}$. The vector $\phi_B(b)$ contains zeros at all positions not belonging to the block $x_j, \ldots, x_{j+k-1}$. The positions for $x_j, \ldots, x_{j+k-1}$ are replaced by the binary representation of the index of the $|b|$th variable of L. Then $\text{INDEX}_s(a, b) = g(\phi_A(a), \phi_B(b))$ and we have constructed the proposed rectangular reduction.

Exercise 11.15. We use similar arguments as in the proof of Theorem 7.5.15. Here G is a randomized $(k-1)$-OBDD of size s representing $\mathrm{PJ}_{k,n}$ with two-sided 1/3-bounded error. We use the same approach to obtain a pointer jumping scenario as a subfunction of $\mathrm{PJ}_{k,n}$ such that the randomized $(k-1)$-OBDD G' obtained from G by the corresponding replacements by constants has at most $2k-2$ layers with respect to the variables given to Alice and Bob. Hence, we obtain a randomized protocol of a length bounded by $(2k-2)\lceil \log s \rceil$ for a pointer jumping scenario for the parameter $n^{1/2}/3$ (instead of n). Alice and Bob "evaluate" the randomized nodes in their layers by flipping coins. By Theorem 11.7.1.ii,

$$(2k-2)\lceil \log s \rceil \geq c(n^{1/2}/(3k^2) - k\log(n^{1/2}/3))$$

and $s = 2^{\Omega(n^{1/2}/k^2)}$.
This leads to an exponential lower bound for constant k and to a non-polynomial lower bound for $k = o(n^{1/4}/\log^{1/2} n)$.

Exercise 11.16. For $\epsilon \leq 1/3$ we obtain the same lower bounds as proved in the solution of Exercise 11.15. Let $\epsilon > 1/3$ and let G be a randomized $(k-1)$-OBDD representing $\mathrm{PJ}_{k,n}$ with two-sided ϵ-bounded error. Let s be the size of G. We have seen in Exercise 11.4 that Theorem 11.3.5 also holds for $(k-1)$-OBDDs. Hence, we obtain a randomized $(k-1)$-OBDD G' representing $\mathrm{PJ}_{k,n}$ with two-sided 1/3-bounded error whose size is bounded above by s^m where $m = O((\frac{1}{2}-\frac{1}{3})^{-2}\log(\epsilon^{-1}))$ is in our case a constant. Hence, by the solution of Exercise 11.15 we get the lower bound $2^{\Omega(n^{1/2}/k^2 m)} = 2^{\Omega(n^{1/2}/k^2)}$ which only differs by a constant in the exponent compared to the case of an error probability of 1/3.

Exercise 11.17. We may use the same approach as in the solutions of Exercise 11.15 and Exercise 11.6. Again, the solution of Exercise 11.4 ensures that we may apply Theorem 11.3.5 for $(k-1)$-IBDDs. Here, we obtain a randomized protocol with two-sided 1/3-bounded error probability and length bounded by $(2k-2)\lceil \log s \rceil$ for a pointer jumping scenario for a parameter $2^{\Omega(\log^\delta n)}$ instead of n, if $k \leq (1-\delta)\log\log n$. By Theorem 11.7.1.ii,

$$(2k-2)\lceil \log s \rceil = \Omega(2^{\Omega(\log^\delta n)}/k^2 - k\log n) = 2^{\Omega(\log^\delta n)},$$

if $k \leq (1-\delta)\log\log n$. For these values of k, the size of randomized $(k-1)$-IBDDs representing $\mathrm{PJ}_{k,n}$ with two-sided 1/3-bounded error grows superpolynomially. The same holds for each constant $\epsilon < 1/2$ instead of 1/3. For constant k, $2^{\Omega(\log^\delta n)}$ can be replaced by $\Omega(n^\alpha)$ for some $\alpha > 0$ (see the proof of Theorem 7.5.17) and this leads to lower bounds of exponential size.

Exercise 11.18. It is obvious that ZPP-FBDD $\subseteq$ RP-FBDD $\cap$ coRP-FBDD. Let $f = (f_n) \in$ RP-FBDD $\cap$ coRP-FBDD. Let $G_{n,1}$ be a randomized FBDD of polynomial size $p_1(n)$ representing f_n with one-sided ϵ_1-bounded error for some $\epsilon_1 < 1$. Let $G_{n,2}$ be a randomized FBDD of polynomial size $p_2(n)$ representing

$\overline{f}_n$ with one-sided ϵ_2-bounded error for some $\epsilon_2 < 1$. We may assume that $G_{n,1}$ and $G_{n,2}$ do not have a ?-sink. Let G_n be the following randomized FBDD. It starts with a randomized node. The 0-edge leads to the source of $G_{n,1}$ where we replace the 0-sink by a ?-sink. The 1-edge leads to the source of $G_{n,2}$ where we replace the 0-sink by a ?-sink and the 1-sink by a 0-sink. Let $f_n(a) = 1$. If we leave the first node by its 0-edge, we reach the 1-sink with a probability at least $1 - \epsilon_1$ and we reach the ?-sink otherwise. In $G_{n,2}$, we reach the 0-sink for $a \in f_n^{-1}(1)$. Hence, if we leave the first node by its 1-edge, we reach with probability 1 the ?-sink. We never reach the 0-sink and the probability of reaching the correct sink, namely the 1-sink, is at least $(1 - \epsilon_1)/2$. Hence, the failure probability is bounded above $1 - (1 - \epsilon_1)/2 \leq (1 + \epsilon_1)/2 < 1$. The same analysis leads for inputs $b \in f_n^{-1}(0)$ to a failure probability bounded above by $(1 + \epsilon_2)/2 < 1$. Also in this case the wrong sink, here the 1-sink, is never reached. Hence, G_n is a randomized FBDD of polynomial size proving that $f = (f_n) \in$ ZPP-FBDD.

Exercise 11.20. HWB is a separating example . It is contained in NP-OBDD$\cap$ coNP-OBDD (Theorem 10.2.1) but not contained in BPP-OBDD (Theorem 11.7.8) and, therefore, not in RP-OBDD (Theorem 11.3.4).