Electronic Colloquium on Computational Complexity, Report No. 28 (2007)

Implicit Simulation of FNC Algorithms

Daniel Sawitzki*
University of Dortmund, Computer Science 2
D-44221 Dortmund, Germany
daniel.sawitzki@cs.uni-dortmund.de
http://ls2-www.cs.uni-dortmund.de/~sawitzki/

February 12, 2007

Abstract

Implicit algorithms work on their input's characteristic functions and should solve problems heuristically by as few and as efficient functional operations as possible. Together with an appropriate data structure to represent the characteristic functions they yield heuristics which are successfully applied in numerous areas. It is known that implicit algorithms which execute $t(N)$ functional operations while using at most $k \log_2 N$ Boolean variables can be simulated by CREW-PRAMs with $\mathcal{O}(N^k)$ processors and parallel runtime $\mathcal{O}((t(N))^2)$. In this paper we consider the opposite case and present a simulation of FNC algorithms by implicit OBDD-based algorithms with polylog(N) functional operations and $\mathcal{O}(\log N)$ Boolean variables.

1 Introduction

For $\mathbb{B} := \{0, 1\}$, let us denote the ith character of a binary string $x \in \mathbb{B}^n$ by x_i and let $|x|$ identify its length n. The class of Boolean functions $f\colon \{0,1\}^n \to \{0,1\}$ will be denoted by B_n. We define the *characteristic Boolean function* $\chi_I \in B_n$ of some $I \in \mathbb{B}^N$ by $\chi_I(x) := I_x$ for $n := \lceil \log_2 N \rceil$, $x \in \mathbb{B}^n$, and $I_N, \ldots, I_{2^n-1} := 0$.

The input $I \in \mathbb{B}^N$ of *Implicit Algorithms* is given by the characteristic function $\chi_I \in B_n$ of I. We then want so solve problems on I without extracting too much explicit information from it. Instead, most work should be done by functional operations on χ_I and further intermediate results. Finally, also the problem's solution $O \in \mathbb{B}^M$ should be presented as $\chi_O \in B^m$ for $m := \lceil \log_2 M \rceil$.

*Supported by DFG grant We 1066/10-3.

ISSN 1433-8092

The purpose of this approach is to create practically successful heuristics on inputs which are large and structured at the same time. Therefore, an implicit algorithm should execute a small number $t(N)$ of functional operations on inputs of size N. The remaining computation steps should be of the same magnitude as $t(N)$. To obtain an overall efficient heuristic, each single functional operation has to be efficient. Hence we need an appropriate data structure for the hopefully succinct representation of all handled characteristic functions.

Ordered Binary Decision Diagrams (*OBDDs*) [2, 15] are a data structure for Boolean functions which is proven as succinct representation for structured and regular data in many practical applications. They are often used to represent characteristic functions in implicit algorithms (e. g. for special problems in CAD and Model Checking, see [8, 15]), because they have efficient algorithms for all usual functional operations like binary operators, quantifications or satisfiability tests. This efficiency is related to the corresponding operand OBDD sizes, while the latter may grow exponentially during a short sequence of only $\log_2 N$ OBDD operations. So the analysis of OBDD-based implicit algorithms is not trivial. In fact, their theoretical analysis has mostly been restricted to the number of functional operations up to the present, which at last is no more than a trivial lower bound on the real overall runtime.

Recent research tries to develop theoretical foundations on OBDD-based graph algorithms. On the one hand, this includes the development of implicit methods for fundamental problems like topological sorting [16] and the computation of connected components [5, 6], maximum flows [9, 12], and shortest paths [11, 11]. On the other hand, we need more sophisticated analysis techniques to explain the practical success of implicit algorithms.

Problems typically get harder when their input is represented implicitly. For circuit representations, this is shown in [1, 4, 10]. Because OBDDs may be exponentially larger than circuits, these results do not directly carry over to problems on OBDD-represented inputs. Feigenbaum et al. [3] prove that the *Graph Accessibility Problem* is PSPACE-complete on OBDD-represented graphs. First efficient upper bounds on time and space of implicit graph algorithms on special inputs have been presented by Sawitzki [11, 12] and Woelfel [16]. These results rely on restrictions on the complete OBDD width of occurring OBDDs.

In [13, 14] it is shown that implicit algorithms with a polylogarithmic number of functional operations using $\mathcal{O}(\log N)$ Boolean can be simulated by CREW-PRAMS with polylogarithmic parallel time and a polynomial number of activated processors.

In this work we consider the opposite situation and show that every problem in the class FNC has also an according implicit algorithm. In Section 2 we introduce OBDDs, before addressing further preliminaries in Section 3. Then we finally present the simulation result in Section 4. Section 5 gives conclusions on the work.

2 Ordered Binary Decision Diagrams

A Boolean function $f \in B_n$ defined on variables $x_0, \ldots, x_{n-1}$ can be represented by an *Ordered Binary Decision Diagram (OBDD)* [2]. An OBDD $\mathcal{G}$ is a directed acyclic graph consisting of *internal nodes* and *sink nodes*. Each internal node is labeled with a Boolean variable x_i, while each sink node is labeled with a Boolean constant. Each internal node is left by two edges one labeled 0 and the other 1. A *function pointer* p marks a special node that represents f. Moreover, a permutation $\pi \in \Sigma_n$ called *variable order* must be respected by the internal nodes' labels on every path from p to a sink. For a given variable assignment $a \in \mathbb{B}^n$, we compute the function value $f(a)$ by traversing $\mathcal{G}$ from p to a sink labeled with $f(a)$ while leaving each node labeled with x_i via its a_i-edge.

An OBDD with variable order π is called π-OBDD. The minimal-size π-OBDD for a function $f \in B_n$ is known to be canonical and will be denoted by $\pi\text{-OBDD}[f]$. Its size $\text{size}(\pi\text{-OBDD}[f])$ is measured by the number of its nodes. Its width $\text{width}(\pi\text{-OBDD}[f])$ is the maximum number of its nodes labeled with the same variable. We adopt the usual assumption that all OBDDs occurring in implicit algorithms have minimal size, since all essential OBDD operations produce minimized diagrams. On the other hand, finding an optimal variable order leading to the minimum size OBDD for a given function is known to be NP-hard. Independent of π it is $\text{size}(\pi\text{-OBDD}[f]) \leq \big(2 + o(1)\big)2^n/n$ for any $f \in B_n$.

OBDDs offer algorithms (called *OBDD operations* in the following) for all the essential functional operations on Boolean functions, which are efficient w. r. t. the size of involved OBDDs. The satisfiability of f can be decided in time $\mathcal{O}(1)$. The negation $\overline{f}$, the replacement of a variable x_i by some constant c (i. e., $f_{|x_i=c}$), and computing $|f^{-1}(1)|$ are possible in time $\mathcal{O}\big(\text{size}(\pi\text{-OBDD}[f])\big)$. The set $f^{-1}(1)$ of f's minterms can be obtained in time $\mathcal{O}\big(n \cdot |f^{-1}(1)|\big)$. Whether two functions f and g are equivalent (i. e., $f = g$) can be decided in time $\mathcal{O}\big(\text{size}(\pi\text{-OBDD}[f]) + \text{size}(\pi\text{-OBDD}[g])\big)$. The most important OBDD operation is the *binary synthesis* $f \otimes g$ for $f, g \in B_n$, $\otimes \in B_2$ (e. g., $\wedge$, $\vee$); in general, it produces the result $\pi\text{-OBDD}[f \otimes g]$ in time and space $\mathcal{O}\big(\text{size}(\pi\text{-OBDD}[f]) \cdot \text{size}(\pi\text{-OBDD}[g])\big)$. The synthesis is also used to implement *quantifications* $(\mathcal{Q}x_i)f$ for $\mathcal{Q} \in \{\exists, \forall\}$. Hence, computing $\pi\text{-OBDD}\big[(\mathcal{Q}x_i)f\big]$ takes time $\mathcal{O}\big(\text{size}^2(\pi\text{-OBDD}[f])\big)$ in general.

Nevertheless, a sequence of only n synthesis operations may cause an exponential blow-up on OBDD sizes, in general. The book of Wegener [15] gives a comprehensive survey on different types of Binary Decision Diagrams.

3 Preliminaries

In this work we consider a *search problem* Π to be a map $\Pi\colon \mathbb{B}^* \to \mathcal{P}(\mathbb{B}^*)$ mapping each input instance to a set of valid solutions. We formally define the problem class with the desired kind of implicit algorithms:

Definition 1 P-Ops *is the class of search problems* $\Pi\colon \mathbb{B}^* \to \mathcal{P}(\mathbb{B}^*)$*, for which an implicit OBDD-based algorithm* $\mathcal{A}$ *exists that on input* χ_I *with* $I \in \mathbb{B}^N$ *computes an output* χ_O *with* $O \in \Pi(I)$ *using* $\mathrm{polylog}(N)$ *OBDD operations on functions defined on* $\mathcal{O}(\log N)$ *Boolean variables. Furthermore* $\mathcal{A}$ *may execute* $\mathrm{polylog}(N)$ *regular computation steps besides the OBDD operations.*

Because we are only interested in the *number* of OBDD operations, this work's simulation result is mainly independent of the data structure for characteristic functions, as long as they support the same set of functional operations. There is one exception: The simulation will rely on some supporting functions which have to be generated first before processed by functional operations. This step will of course depend on the OBDD size of these functions.

The following definitions are mainly taken from [7].

Definition 2 FNC *is the class of search problems* $\Pi\colon \mathbb{B}^* \to \mathcal{P}(\mathbb{B}^*)$ *solved by CREW-PRAMs in parallel time* $\mathrm{polylog}(N)$ *using* $\mathrm{poly}(N)$ *processors.*

Definition 3 *Boolean circuits.*

(a) A Boolean circuit C *is a directed acyclic graph. Each node* v *has a* type $g(v) \in \{\mathrm{I}, \mathrm{O}\} \cup B_0 \cup B_1 \cup B_2$*. Nodes* v *with* $g(v) = \mathrm{I}$ *have indegree 0 and are called* input*. Nodes* v *with* $g(v) = \mathrm{O}$ *have indegree 1, outdegree 0 and are called* output*. Nodes* v *with* $g(v) \in B_i$ *must have indegree* i *and are called* gate*.*

(b) A Boolean circuit C *with inputs* $x_0, \ldots, x_{N-1}$ *and outputs* $y_0, \ldots, y_{M-1}$ *computes a function* $f\colon \mathbb{B}^N \to \mathbb{B}^M$ *in the following way: Each input* x_i *gets a* value $b(x_i) \in \mathbb{B}$ *corresponding to the* i*th argument of* f*. The value* $b(v) \in \mathbb{B}$ *of each other node* v *is obtained by recursively applying* $g(v)$ *on the values of nodes adjacent to* v*. The value of* f *is finally obtained by the values of* C*'s outputs.*

(c) The size $\mathrm{size}(C)$ *of a Boolean circuit* C *is the number of its nodes. The* depth $\mathrm{depth}(C)$ *of* C *is the maximum path length in* C*.*

(d) The standard encoding *of a Boolean circuit* C *with inputs* $x_0, \ldots, x_{N-1}$ *and outputs* $y_0, \ldots, y_{M-1}$ *is a binary string* $\bar{C} \in \mathbb{B}^*$*. This string* $\bar{C}$ *consists of* $\mathrm{size}(C)$ *4-tuples* (j, g, ℓ, r) *followed by both sequences of node number of* $x_0, \ldots, x_{N-1}$ *and* $y_0, \ldots, y_{M-1}$*. Each tuple* (j, g, ℓ, r) *represents a node* v*:* j *is an arbitrary unique number in* $\{0, \ldots, \mathrm{size}(C) - 1\}$ *for* v*,* g *is* $g(v)$*, and* ℓ *as well as* r *are the numbers of the up to two predecessors of* v *in* C*.*

(e) A sequence $C := (C_N)_N$ *of Boolean circuits* C_N *with inputs* $x_0, \ldots, x_{N-1}$ *and outputs* $y_0, \ldots, y_{M-1}$ *is called* logarithmic space-uniform*, if each* $\bar{C}_N$ *can be computed by a deterministic Turing machine* $\mathcal{M}_C$ *with input* N *on space* $\mathcal{O}(\log |\bar{C}_N|)$*.*

We now introduce an alternative characterization of parallel FNC algorithms, which we will use for our simulation.

Lemma 1 (see Lemma 2.4.2 in [7]) *A search problem* $\Pi\colon \mathbb{B}^* \to \mathcal{P}(\mathbb{B}^*)$ *is computed by a sequence* $C := (C_N)_N$ *of logarithmic space-uniform circuits of size* $\mathrm{poly}(N)$ *and depth* $\mathrm{polylog}(N)$ *if and only if* $\Pi \in \mathrm{FNC}$.

Finally, we take a look at a powerful class of Boolean functions, for which good upper bounds on their OBDD size are known.

Definition 4 *Let* $f \in B_{kn}$ *be defined on variable vectors* $x^{(1)}, \ldots, x^{(k)} \in \mathbb{B}^n$. *Function* f *is called* k-variate comparison function *if there are* $W \in \mathbb{N}$, $t \in \mathbb{Z}$, *and* $\delta_1, \ldots, \delta_k \in \{-W, \ldots, W\}$, *such that*

$$f\left(x^{(1)}, \ldots, x^{(k)}\right) = \left(\sum_{i=1}^{k} \delta_i \cdot \sum_{j=0}^{n-1} 2^j x_j^{(i)} \bowtie t\right)$$

for $\bowtie \in \{\geq, >, \leq, <, =\}$. W *is called* maximum absolute weight *of* f. *The corresponding class of functions is denoted by* $\mathbb{V}_{k,n}^{W}$.

Lemma 2 (see [14] or Corollary 3.7.6 in [13]) *Functions* $f \in \mathbb{V}_{k,n}^{W}$ *have OBDDs of width* $\mathrm{poly}(kW)$ *which can be generated in linear time.*

4 The Simulation Algorithm

Theorem 1 $\mathrm{FNC} \subseteq \text{P-Ops}$.

Proof. We give an implicit OBDD-based algorithm $\mathcal{A}$ which computes and simulated logarithmic space-uniform circuits by $\mathrm{polylog}(N)$ functional operations using $\mathcal{O}(\log(N))$ Boolean variables for input length N. First we simulate the logarithmic space-uniform Turing machine $\mathcal{T}$ which computes $\bar{C}_N$ from N. $\mathcal{T}$ exists due to Lemma 1. Then we use this implicit representation of $\bar{C}_N$ to simulate C_N.

$\mathcal{A}$ receives the input $I \in \mathbb{B}^N$ for the FNC problem Π in Form of the characteristic function $\chi_I \in B_n$ with $\chi_I(x) := I_x$ and $x \in \mathbb{B}^n$. In the same way, $\mathcal{A}$ presents the output $O \in \mathbb{B}^M$ with $O \in \Pi(I)$ in form of $\chi_O \in B_n$. Due to $M = \mathrm{poly}(N)$ we may assume w. l. o. g. that $n = \mathcal{O}(\log N) = \mathcal{O}(\log M)$ is a sufficient number of Boolean variables for encoding positions in both input and output.

First we define an appropriate implicit representation χ_{C_N} for C_N. This characteristic function should map exactly those binary encoded 4-tuples (j, g, ℓ, r) of length $4t = \mathcal{O}(\log |\bar{C}_N|) = \mathcal{O}(\log N)$ to 1 which are contained in $\bar{C}_N$ in terms of Definition 3(d).

In order to simplify the latter extraction of χ_{C_N} from $\mathcal{T}$'s output later on, we modify $\mathcal{T}$: After each circuit node encoding tuple (j, g, ℓ, r) we output $2^{\lceil \log_2(4t) \rceil} - 4t$ more padding zeros. So we guaranty that each tuple takes

$2^{\lceil \log_2(4t) \rceil} =: T$ positions in the output of $\mathcal{T}$, which still has polynomial length in N. Also the logarithmic space bound of $\mathcal{T}$ is not violated by this.

Finally it is easy to see that $\mathcal{T}$ can enumerate the 4-tuples of both input nodes and output nodes, such that $v_0, \ldots, v_{N-1}$ corresponds to input bits $I_0, \ldots, I_{N-1}$ and $v_{\text{size}(C_N)-M-1}, \ldots, v_{\text{size}(C_N)-1}$ correspond to output bits $O_0, \ldots, O_{M-1}$. So $\mathcal{T}$ may elide to output the input and output node numbers of C_N subsequent to the tuples – their position in the vector $(v_0, \ldots, v_{\text{size}(C_N)} - 1)$ is uniquely defined.

Now we consider the parallel simulation of $\mathcal{T}$ by $\mathcal{A}$. Due to the logarithmic space bound, configurations of $\mathcal{T}$ can be encoded by $m = \mathcal{O}(\log N)$ bits each. We assume w. l. o. g. that the first two bits K_0 and K_1 of such an encoding K are designated to $\mathcal{T}$'s output behavior: Let K_0 be 1 iff $\mathcal{T}$ produces an output in the current computation step. Then let K_1 be this output while leaving it arbitrarily defined else.

Moreover, let $\chi_{\mathcal{T}} \in \mathbb{B}_{2m}$ be the implicit configuration transition relation of $\mathcal{T}$, which maps two configuration encodings K and K' to 1 if and only if they represent a computation step of $\mathcal{T}$. It is easy to see that the OBDD for $\chi_{\mathcal{T}}$ can be generated analogously to [3, 14] or Theorem 7.2.8 in [13] in time $\mathcal{O}(m)$. (Reading K and K' interleaved, the correct transition can be verified locally for any sequence of three tape positions, which directly implies a constant OBDD width.)

Concatenating configuration transitions of a deterministic Turing machine obviously is an associative operation. So we can use a simple divide-and-conquer approach to solve the according prefix problem and, therewith, obtain $\mathcal{T}$'s configuration sequence of length at most $2^m = \text{poly}(N)$ executing only few OBDD operations.

Hence we introduce the function $\chi'_{\mathcal{T}} \in B_{3m}$, which maps binary encoded triplets (i, K, K') to 1 if and only if $\mathcal{T}$ transforms configuration K to K' in exactly i computation steps. We obtain $\chi'_{\mathcal{T}}$ by iteratively computing the functions $\chi^{(k)}_{\mathcal{T}} \in B_{3m}$ for $k := 0, \ldots, m$, such that $\chi^{(k)}_{\mathcal{T}}$ covers all triplets $(i, K, K') \in (\chi'_{\mathcal{T}})^{-1}(1)$ with $i \leq 2^k$ (i. e., maps them to 1).

Clearly it is $\chi^{(0)}_{\mathcal{T}} = \chi_{\mathcal{T}}$. The iterative step is then done in the following way:

$$\begin{aligned}\chi^{(k+1)}_{\mathcal{T}}(i, K, K') := (i \leq 2^k) \wedge \chi^{(k)}_{\mathcal{T}}(i, K, K') \\ \vee \big[(2^k < i \leq 2^{k+1}) \\ \wedge (\exists j, L)[(i = 2^k + j) \wedge \chi^{(k)}_{\mathcal{T}}(2^k, K, L) \wedge \chi^{(k)}_{\mathcal{T}}(j, L, K')]\big].\end{aligned}$$

So first we check for $i > 2^k$ if, starting from K, an intermediate configuration L can be reached by 2^k computation steps, from which in turn we can reach K' by further $i - 2^k$ steps. Finally it is $\chi^{(m)}_{\mathcal{T}} = \chi'_{\mathcal{T}}$.

It is $m = \mathcal{O}(\log N)$. The OBDD generation of supporting functions is discussed later. In each iteration there are $\mathcal{O}(1)$ quantification blocks over $\mathcal{O}(m)$ variables each. This number dominates the overall amount of OBDD operations and remaining runtime in each iteration. So $\chi'_{\mathcal{T}}$ is known after $\mathcal{O}(\log^2 N)$ OBDD operations.

Now we replace the K-argument in $\chi'_{\mathcal{T}}$ by the encoding of $\mathcal{T}$'s starting configuration on input N by $\mathcal{O}(m) = \mathcal{O}(\log N)$ OBDD operations. Moreover, using the same number of OBDD operations we apply existential quantifications on the variables $K'_2, \ldots, K'_{m-1}$ and call the new function $\chi''_{\mathcal{T}}$. It only depends on the state index i and the output encoding variables K'_0 and K'_1.

This intermediate result has now to be compressed to a function $\chi_{\mathrm{R}}(i,a) \in B_{m+1}$ with $i \in \mathbb{B}^m$ and $a \in \mathbb{B}$, which maps (i,a) to 1 if and only if a is the bit at position i of $\mathcal{T}$'s output.

Therefore, we first compute functions $\chi_{\mathrm{S}}(i,j) \in B_{2m}$, which map (i,j) to 1 if and only if the jth output is produced in the ith computation step of $\mathcal{T}$. We iteratively compute functions $\chi_{\mathrm{S}}^{(k)}(i,j) \in B_{2m}$ for $k := 0, \ldots, m$, which map (i,j) to 1 if and only if exactly j outputs are generated by $\mathcal{T}$ during its computation steps $\lfloor i/2^k \rfloor \cdot 2^k, \ldots, i$. So the ranks obtained in iteration k are related to an interval of length 2^k.

If and only if there happens an output in step i (i. e., $K'_0 = 1$), this is the first and only output in $[i,i]$ (i. e. $j = 1$) and we initialize the computation by

$$\chi_{\mathrm{S}}^{(0)}(i,j) \;:=\; (\exists K'_1)\big[\chi''_{\mathcal{T}}(i,1,K'_1) \wedge (j = 1) \vee \chi''_{\mathcal{T}}(i,0,K'_1) \wedge (j = 0)\big].$$

We consider the iteration. Let be $\alpha(i,k) := \lfloor i/2^k \rfloor \cdot 2^k$.

$$\begin{aligned}\chi_{\mathrm{S}}^{(k+1)}(i,j) := \big[(j \le \alpha(i,k+1) + 2^k) \wedge \chi_{\mathrm{S}}^{(k)}(i,j)\big] \\ \vee (j > \alpha(i,k+1) + 2^k) \wedge (\exists i', j', j'')\big[(i' = \alpha(i,k+1) + 2^{k+1}) \\ \wedge (j = j' + j'') \wedge \chi_{\mathrm{S}}^{(k)}(i',j') \wedge \chi_{\mathrm{S}}^{(k)}(i,j'')\big]. \quad (1)\end{aligned}$$

Let us analyze the correctness: If i lies in the "left" part $[\alpha(i,k+1), \alpha(i,k+1) + 2^k]$ of the next larger interval $[\alpha(i,k+1), \alpha(i,k+1) + 2^{k+1}]$, it clearly is $\chi_{\mathrm{S}}^{(k+1)}(i,j) = \chi_{\mathrm{S}}^{(k)}(i,j)$. Else the number j' of outputs in the left part has to be added to the rank j'' of i in the right part in order to obtain the correct j.

In the same way as for $\chi'_{\mathcal{T}}$ it follows, that $\mathcal{O}(\log^2 N)$ OBDD operations is sufficient for computing $\chi_{\mathrm{S}} = \chi_{\mathrm{S}}^{(m)}$. Again the OBDD generation is discussed later.

Finally we obtain χ_{R} by

$$\chi_{\mathrm{R}}(i,a) := (\exists j)\big[\chi_{\mathrm{S}}(j,i) \wedge \chi''_{\mathcal{T}}(j,1,a)\big].$$

So the output of $\mathcal{T}$ has an a at position i iff an a is outputted in computation step j and this step has rank i in the output order. Due to the dominating number of quantifications, $\mathcal{O}(\log N)$ overall OBDD operations are executed here.

Now we have to use $\chi_{\mathrm{R}}(i,a)$ in order to obtain $\chi_{C_N}(j,h,\ell,r)$, which finally will enable us to simulate C_N. We now use the property that $\mathcal{T}$ outputs each 4-tuple (j,g,ℓ,r) as block of length T, which is a power of two. This implies:

$$\chi_{C_N}(j,g,\ell,r) = (\exists j^*) \bigwedge_{k=0}^{t-1} (\exists j^{(1)}, j^{(2)}, j^{(3)}, j^{(4)}) \big[(j^{(1)} = j^* + k) \wedge \chi_{\mathrm{R}}(j^{(1)}, j_k)$$
$$\wedge (j^{(2)} = j^* + t + k) \wedge \chi_{\mathrm{R}}(j^{(2)}, g_k) \wedge (j^{(3)} = j^* + 2t + k) \wedge \chi_{\mathrm{R}}(j^{(3)}, \ell_k)$$
$$\wedge (j^{(4)} = j^* + 3t + k) \wedge \chi_{\mathrm{R}}(j^{(4)}, r_k) \big]. \quad (2)$$

We consider the correctness: The variables $i_0, \ldots, i_{m-\log_2 T-1}$ of position argument i of $\chi_{\mathrm{R}}(i,a)$ corresponds to the index j^* of a tuple (j,g,ℓ,r), and the variables $i_{m-\log_2 T}, \ldots, i_{m-1}$ correspond to the bit index within this tuple. So function χ_{C_N} maps (j,g,ℓ,r) to 1 if and only if such an index j^* exists, and at the corresponding output positions j^*+k, j^*+t+k, j^*+2t+k and j^*+3t+k of $\mathcal{T}$ the bits j_k, g_k, ℓ_k and r_k are contained.

Computing χ_{C_N} causes $\mathcal{O}(\log N)$ OBDD operations and remaining runtime: The number OBDD operations is dominated by $\mathcal{O}(t) = \mathcal{O}(m) = \mathcal{O}(\log N)$ existential quantifications and conjunctions. The OBDD generation is discussed later.

Remark: Without filling up the tuple blocks, the position of a tuple within the overall output of $\mathcal{T}$ had to be calculated by a multiplication of a tuple index with the tuple length $4t$. For this multiplication no small OBDD size could be guaranteed.

It remains the simulation of C_N on I. Therefore we introduce a function $\chi_b \in B_{t+1}$, which maps a pair (j,a) consisting of a circuit node number $j \in \mathbb{B}^t$ and a value bit $a \in \mathbb{B}$ to 1 if and only if $b(v_j) = a$ for input I. We compute χ_b iteratively with the help of intermediate functions $\chi_b^{(k)}$ for $k := 0, \ldots, D$ and $D := \mathrm{depth}(C_N) = \mathrm{polylog}(N)$, such that $\chi_b^{(k)}$ correctly represents the values up to level k of C_N.

Initially, only input nodes have a value:

$$\chi_b^{(0)}(j,a) := (j < N) \wedge \big[(a=1) \wedge \chi_I(j) \vee (a=0) \wedge \overline{\chi_I(j)} \big].$$

For the iterative step, we moreover assume that the function $\chi_{\mathrm{X}}(g,x,y,a)$ is given, which maps a binary encoded gate type $g \in B_0 \cup B_1 \cup B_2 =: H$, two input bits x and y, and a result bit a to 1 if and only if a is the corresponding output of a g-gate on x and y. For $g \in B_1$ the function χ_{X} depends only on x, for $g \in B_0$ it depends neither on x nor on y. Due to $|H| = \mathcal{O}(1)$ the OBDD for χ_{X} can obviously be generated in time $\mathcal{O}(\log N)$.

$$\chi_b^{(k+1)}(j,a) := \overline{(\exists a')\chi_b^{(k)}(j,a')} \wedge (\exists g, j', j'', a', a'') \big[\chi_{C_N}(j,g,j',j'')$$
$$\wedge \chi_b^{(k)}(j',a') \wedge \chi_b^{(k)}(j'',a'') \wedge \chi_{\mathrm{X}}(g,a',a'',a) \big]$$

So if $b(v_j)$ is unknown yet, while the values of its predecessor nodes $v_{j'}$ and $v_{j''}$ in the circuit graph have been computed already, then the value input a has to be equal to the output of a g-gate on the values a' and a'' of $v_{j'}$ and $v_{j''}$.

Finally it is $\chi_b = \chi_b^{(\mathrm{depth}(C_N))}$. By the same arguments as for the computation of $\chi'_{\mathcal{T}}$, a number of $\mathcal{O}(\log^{\mathrm{depth}(C_N)+1} N)$ OBDD operations and remaining computation steps follows.

We still have to extract the output χ_O:

$$\chi_O(x) := (x < M) \wedge (\exists j)\big[(j = \mathrm{size}(C_N) - M + x) \wedge \chi_b(j, 1)\big].$$

This is correct since the xth output bit of C_N corresponds to the circuit node with number $\mathrm{size}(C_N) - M + x$. The circuit size in turn can be obtained by counting the number of satisfying inputs of the function $(\exists g, \ell, r)\chi_{C_N}(j, g, \ell, r)$ by executing $\mathcal{O}(\log N)$ OBDD operations.

By consideration of each single phase of $\mathcal{A}$ we conclude an overall number of $\mathcal{O}(\log^{\mathrm{depth}(C_N)+1} N) = \mathrm{polylog}(N)$ OBDD operations, while the remaining runtime is of the same order. We still have to discuss the OBDD size of all supporting functions like $(j^{(4)} = j^* + 3t + k)$ (see Equation 2). However, each of these clearly can be expressed as multivariate comparison function with constant maximum absolute weight W and a constant number k of variable vectors (see Definition 4). Due to Lemma 2, these functions have OBDDs of constant width and, therefore, can be constructed in time $\mathcal{O}(\log N)$ by $\mathcal{A}$.

We have to take a closer look at Equation 1: In contrast to the original definition of multivariate comparison functions, the argument i appears in form of $\alpha(i, k) = \lfloor i/2^k \rfloor \cdot 2^k$. But this operation just replaces the k least significant bits by zero. So corresponding OBDD nodes testing these bits resp. variables have just to be replaced by their 0-edge — this does not enlarge the OBDD size. □

5 Conclusions

Together with the previous result P-Ops ⊆ FNC from [13, 14], this work's simulation result finally shows P-Ops = FNC. While for some problems the existence of P-Ops algorithms follows by intuition, for other problems (e. g. planarity test) only involved FNC algorithms are known which do not allow an obvious conversion to implicit algorithms. In these cases, the new simulation result directly implicates the existence of corresponding P-Ops algorithms and can be used to construct them.

Acknowledgments.

Thanks to Martin Sauerhoff and Ingo Wegener for helpful discussions.

References

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

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

[3] J. Feigenbaum, S. Kannan, M. Y. Vardi, and M. Viswanathan. Complexity of problems on graphs represented as OBDDs. In *Symposium on Theoretical Aspects of Computer Science 1998*, volume 1373 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1998.

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

[5] R. Gentilini, C. Piazza, and A. Policriti. Computing strongly connected components in a linear number of symbolic steps. In *Symposium on Discrete Algorithms 2003*, pages 573–582. ACM Press, 2003.

[6] R. Gentilini and A. Policriti. Biconnectivity on symbolically represented graphs: A linear solution. In *International Symposium on Algorithms and Computation 2003*, volume 2906 of *Lecture Notes in Computer Science*, pages 554–564. Springer, 2003.

[7] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation.* Oxford University Press, 1995.

[8] G. D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms.* Kluwer Academic Publishers, 1996.

[9] G. D. Hachtel and F. Somenzi. A symbolic algorithm for maximum flow in 0–1 networks. *Formal Methods in System Design*, 10:207–219, 1997.

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

[11] D. Sawitzki. A symbolic approach to the all-pairs shortest-paths problem. In *Graph-Theoretic Concepts in Computer Science 2004*, volume 3353 of *Lecture Notes in Computer Science*, pages 154–167. Springer, 2004.

[12] D. Sawitzki. Implicit flow maximization by iterative squaring. In *SOFSEM 2004: Theory and Practice of Computer Science*, volume 2932 of *Lecture Notes in Computer Science*, pages 301–313. Springer, 2004.

[13] D. Sawitzki. *Algorithmik und Komplexität OBDD-repräsentierter Graphen*. PhD thesis, Lehrstuhl Informatik 2, Universität Dortmund, 2006.

[14] D. Sawitzki. The complexity of problems on implicitly represented inputs. In *SOFSEM 2006: Theory and Practice of Computer Science*, volume 3831 of *Lecture Notes in Computer Science*, pages 471–482. Springer, 2006.

[15] I. Wegener. *Branching Programs and Binary Decision Diagrams*. Monographs on Discrete Mathematics and Applications. SIAM Press, 2000.

[16] P. Woelfel. Symbolic topological sorting with OBDDs. In *Mathematical Foundations of Computer Science 2003*, volume 2747 of *Lecture Notes in Computer Science*, pages 671–680. Springer, 2003.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de/