Polynomial-size binary decision diagrams for the Exactly half-d-hyperclique problem reading each input bit twice

Daniel Král'*

Abstract

Binary decision diagrams (BDDs) are graph-based data structures representing Boolean functions; ℓ-BDDs are BDDs with an additional restriction that each input bit can be tested at most ℓ times. A d-uniform hypergraph H on N vertices is an exactly half-d-hyperclique if $N/2$ of its vertices form a hyperclique and the remaining vertices are isolated. Wegener [J. ACM 35(2) (1988), 461–471] conjectured that there is no polynomial-size $(d-1)$-BDD for the Exactly half-d-hyperclique problem. We disprove this conjecture by constructing polynomial-size 2-BDDs for the Exactly half-d-hyperclique problem for every $d \geq 2$. Our construction is based on a new idea involving log-space algorithms with faulty inputs.

1 Introduction

Binary decision diagrams are important data structure for representation of Boolean functions. In particular, they are significant data structures in circuit verification algorithms [12]. Several modifications of binary decision diagrams have been proposed in hope that they can provide small compact representations of important Boolean functions such as integer multiplication. This leads to an enormous interest in upper and lower bounds on the sizes of binary decision diagrams and their variants for various naturally

*Institute for Theoretical Computer Science, Charles University, Malostranské náměstí 25, 118 00 Prague 1, Czech Republic. E-mail: kral@kam.mff.cuni.cz. Institute for Theoretical Computer Science (ITI) is supported by Ministry of Education of Czech Republic as project LN00A056.

defined Boolean functions. However, since the sizes of binary decision diagrams are closely related to non-uniform space complexity, the problems are also interesting from the theoretical point of view. The reader is referred to the monographs [10, 12] by Wegener for a more detailed exposition.

A *binary decision diagram (BDD)* is an acyclic oriented graph with three special nodes: the *source*, the 0-*sink* and the 1-*sink*. Each node v of the diagram, except for the 0-sink and 1-sink, is assigned one of the input variables x_i and its out-degree is two. One of the arcs leaving v is labeled with 0, the other with 1. The out-degrees of both the 0-sink and the 1-sink are zero. The *computation path* for the input $x_1, \ldots, x_n$ is the path $v_1, \ldots, v_k$ where the node v_1 is the source, v_k is one of the sinks and the arc between v_i and v_{i+1} is labeled by the value of the variable x_{j_i} assigned to the node v_i. The variable x_{j_i} is said to be *tested* at the node v_i. The input is accepted if the final node of the path is the 1-sink, and the input is rejected, otherwise. The *size* of the diagram is the number of its nodes. The sizes of BDDs for naturally defined sequences of Boolean functions $f_n : \{0,1\}^n \to \{0,1\}$ are often expressed as a function of the input length n.

In this paper, we study binary decision diagrams where the number of tests of input bits is restricted. As noticed by Borodin et al. [2], there are actually two possible types of restrictions: if each path from the source to a sink contains at most ℓ nodes labeled by each of the variables, the BDD is said to be a *syntactic ℓ-BDD*. On the other hand, if only on each computation path from the source to one of the sinks, i.e., the path corresponding to some input, each variable is tested at most ℓ times, the BDD is said to be a *semantic ℓ-BDD*. The notions of syntactic ℓ-BDDs and semantic ℓ-BDDs coincide for $\ell = 1$. A 1-BDD is also called a *free binary decision diagram (FBDD)*.

Clearly, a syntactic ℓ-BDD is also a semantic ℓ-BDD for every $\ell \geq 1$. On the other hand, there is not known a single example of a Boolean function which has polynomial-size semantic ℓ-BDDs but which does not have polynomial-size syntactic ℓ-BDDs. In fact, there is not even known an exponential lower bound on the size of semantic ℓ-BDDs for any explicit function with polynomial-size BDDs (for $\ell \geq 2$). This contrasts to the case of well-understood lower bound techniques for free binary decision diagrams (1-BDDs) [6, 7, 8].

Wegener [11] raised the question whether the class of Boolean functions which can be represented by polynomial-size ℓ-BDDs is a proper subset of the class of Boolean functions which can be represented by polynomial-size $(\ell+1)$-BDDs for every $\ell \geq 1$. The the first function suggested as an example to separate the two classes [12] was the function excl_N^d. The function excl_N^d is a function of $\binom{N}{d}$ Boolean variables which represent possible edges of a d-uniform hypergraph H on N vertices and the function excl_N^d is true if there is a set A of $N/2$ vertices of H such that the edges of H are precisely the sets of $\binom{A}{d}$, i.e., H is formed by a hyperclique of order $N/2$ and its remaining vertices are isolated. Wegener [11] conjectured the following:

Conjecture 1 *For every $d \geq 1$, there is no polynomial-size $(d-1)$-BDD representing the function excl_N^d.*

The conjecture also appears as Problem 7.7 in [12]. The question whether the function excl_N^d has a polynomial-size $(d-1)$-BDDs is still of particular interest (see comments after Theorem 7.2.6 in [12]), though the separation of the classes of syntactic ℓ-BDDs, which motivated Conjecture 1, has recently been settled by Thathachar [9] who constructed two families of explicit functions, which he called the *hyperplanar sum-of-products predicates* and the *conjunction of hyperplanar sum-of-products predicates*, with polynomial-size ℓ-BDDs and with no polynomial-size $(\ell+1)$-BDDs. In the present paper, we disprove Conjecture 1 by constructing polynomial-size syntactic 2-BDDs for the function excl_N^d for every $d \geq 3$.

If $d = 2$ (the case of ordinary graphs), an exponential lower bound on the size of 1-BDDs (FBDDs) for excl_N^d was proved by Žák [13]. He showed that the size of each 1-BDD which represents the function excl_N^d is at least $2^{\Omega(N)}$ (note that the bound is actually of order $2^{\Omega(n^{1/2})}$ where $n = \binom{N}{2}$ is the length of the input). Let us remark that the first lower bound of order $2^{\Omega(n)}$ where n is the length of the input on the size of 1-BDDs for a function with polynomial-size BDDs was proved by Ajtai et al. [1].

As mentioned before, we disprove Conjecture 1 by showing that the function excl_N^d has syntactic 2-BDDs of size $O(N^{\frac{d^2+3d-4}{2}})$ for every $d \geq 2$ (Theorem 2), i.e., of size $O(n^{\frac{d^2+3d-4}{2d}})$ where $n = \binom{N}{d}$ is the length of the input. We design a log-space algorithm for the function excl_N^d which accesses each input bit at most twice. Since it is well-known that the logarithm of the size of BDDs is essentially the space needed by non-uniform Turing machines [3, 5], this already proves the existence of polynomial-size semantic 2-BDDs for excl_N^d. In order, to construct syntactic 2-BDDs for excl_N^d, we show that our algorithm has the log-space complexity and the restriction on the number of tests of input bits is preserved even for "faulty" inputs (see Section 2 for a formal definition). To our best knowledge, the upper bound technique for syntactic ℓ-BDDs based on algorithms with faulty inputs has not been used before. For the sake of completeness, we show that the function excl_N^d has no polynomial-size 1-BDDs in Section 4.

2 Relation between space complexity and BDD size

Our upper bound construction is based on the close relation between the space complexity of non-uniform Turing machines and the sizes of binary decision diagrams computing the same Boolean function. For ordinary BDDs, this relation were already observed by Cobham [3] and by Pudlák and Žák [5]. However in our case, we deal with semantic and syntactic ℓ-BDDs instead of ordinary BDDs, so an additional assumption that the algorithm reads each input bit limited number of times has to be added.

The computational model considered throughout the paper is the following: the algorithm can be in one of finitely many *states*. The number of states cannot depend on the input size. The *configuration* of the algorithm is determined by the pair consisting of its state and the current content of the memory. The allocated memory may depend on the input size. The configuration uniquely determines the next step of the algorithm which results in either a change of the state and the content of the memory or a change of the state which depends on the read input bit (i.e., the new state depends on whether the read input bit is 0 or 1). The step in which the new state depends on the i-th input bit is said to be a *test* of the i-th bit, or simply, the algorithm *tests* the i-th bit. The space complexity of the algorithm is the number of bits of memory needed to run the algorithm. The input size n is stored in additional $\log n$ bits of memory which cannot be modified and which are thus not counted to the space complexity. Finally, let us remark that the base of all logarithms through the paper is two.

We first consider the case of semantic ℓ-BDDs. The proof follows the same idea as in the case of ordinary BDDs but we decided to include its sketch in order to demonstrate the main ideas. We remark that a similar proposition for FBDDs, i.e., 1-BDDs, can be found in [4].

Proposition 1 *Let $f_n : \{0,1\}^n \to \{0,1\}$ be a sequence of Boolean functions and $\ell \geq 1$ an integer. If there exists an algorithm which for the input of length n:*

- *computes the function f_n,*

- *works in space $k \log n + O(1)$ for some $k \geq 0$, and*
- *accesses each bit of the input string at most ℓ times,*

then there exists a semantic ℓ-BDD representing the function f_n of size $O(n^{2k})$.

Proof: Consider a directed graph G whose vertices are the triples (s, σ, t) consisting of a state s and content of memory σ which can be reached after $t \geq 0$ steps for some input. If the algorithm at the configuration (s, σ) does not test any input bits, the vertex (s, σ, t) is joined by two edges to the vertex $(s', \sigma', t+1)$ where (s', σ') is the configuration reached after (s, σ). The two edges leaving (s, σ, t) are labeled with 0 and 1. If the algorithm reads an input bit x_i at the configuration (s, σ), the vertex (s, σ, t) is assigned x_i and it is joined by an edge labeled with 0 to the vertex $(s', \sigma', t+1)$ where (s', σ') is the configuration reached by the algorithm after (s, σ) if $x_i = 0$ and by an edge labeled with 1 to the vertex $(s', \sigma', t+1)$ where (s', σ') is the configuration reached by the algorithm after (s, σ) if $x_i = 1$. There are no edges leaving the vertices corresponding to the halting configurations of the algorithm. Note that the graph G is acyclic because a vertex (s, σ, t) is joined by edges only to vertices $(s', \sigma', t+1)$.

We now modify G to a BDD. Contract all the halting configurations in which the algorithm accepts to a single vertex. This vertex will be the 1-sink. Similarly, contract all the halting configurations in which the algorithm rejects to a single vertex which will be the 0-sink. If there is a single edge leaving a vertex (s, σ, t) (this may happen if one of the results of input tests is inconsistent with the previous computation) or there are two edges leaving a vertex (s, σ, t) which lead to the same vertex of G, contract the vertex (s, σ, t) and its unique successor. The variable assigned to the resulting vertex is the variable assigned to the successor. Let G' be the resulting graph. Observe that each vertex of G' except for the sinks has out-degree two and it is assigned one of the variables. In addition, the graph G' is acyclic.

We claim that G' is a semantic ℓ-BDD representing the functions f_n if we set the source of G' to be the vertex $(s, \sigma, 0)$ where (s, σ) is the initial configuration of the algorithm. Consider the computation path for an input $x_1, \ldots, x_n$ in G'. If we expand the edges of the path which were contracted, we obtain a sequence of configurations reached by the algorithm, starting in the initial configuration and ending in a halting configuration. If the algorithm accepts, the final node of the computation path is the 1-sink. If the algorithm rejects, the final node is the 0-sink. Since the algorithm tests each variable at most ℓ times, there are at most ℓ nodes on the computation path which are assigned a variable x_i.

It remains to estimate the size of G'. The number of steps of the algorithm is bounded by the number of possible configurations. Since the number of possible configurations does not exceed $S \cdot 2^{k \log n + O(1)} = O(n^k)$ where S is the number of states of the algorithm, the number of nodes of the original graph G is at most $O(n^{2k})$. This also bounds the number of nodes of G'.

■

We now consider the case of syntactic ℓ-BDDs. The input is said to be *faulty* if the test of a bit of the input string does not necessarily yields its correct value, i.e., it may yield the opposite one. We do not assume any probability distribution of good and bad answers. The considered algorithms are always required to compute the correct answer only if all input tests resulted in correct answers, but they are required to keep their space complexity as well as the restriction on the number of tests of input bits for all faulty inputs:

Proposition 2 *Let $f_n : \{0,1\}^n \to \{0,1\}$ be a sequence of Boolean functions and $\ell \geq 1$ an integer. If there exists an algorithm which for the input of length n:*

- *terminates even if the input is faulty,*
- *computes the function f_n (if the input is not faulty),*
- *works in space $k \log N + O(1)$ for some $k \geq 0$ even in the case of faulty inputs, and*
- *accesses each bit of the input string at most ℓ times even in the case of faulty inputs,*

then there exists a syntactic ℓ-BDD representing the function f_n of size $O(n^k)$.

Proof: We proceed similarly as in the proof of Proposition 1. Consider a directed graph G whose vertices are the configurations (s, σ) which can be reached for some possibly faulty input. If the algorithm at the configuration (s, σ) does not test any input bits, the vertex (s, σ) is joined by two edges to the vertex (s', σ') where (s', σ') is the configuration reached after (s, σ). The two edges leaving (s, σ, t) are labeled with 0 and 1. If the algorithm reads an input bit x_i at the configuration (s, σ), the vertex (s, σ) is joined by an edge labeled with 0 to the vertex (s', σ') where (s', σ') is the configuration reached by the algorithm after (s, σ) if $x_i = 0$ and by an edge labeled with 1 to the vertex (s', σ') where (s', σ') is the configuration reached if $x_i = 1$. There are no edges leaving the vertices corresponding to the halting configurations of the algorithm. Note that the out-degrees of all the vertices except for those corresponding to the halting configurations are two.

We claim that the graph G is acyclic. If G contains a cycle C, consider an infinite trail comprised by a path from the initial configuration to a vertex of C (such a path exists since we included to G only the configurations reachable for some faulty input) and by infinite number of copies of the cycle C. The obtained trail corresponds to execution of the algorithm for some faulty input which never terminates. However, this contradicts our assumption that the algorithm always terminates even for faulty inputs.

We now modify G to a BDD. Contract all the halting configurations in which the algorithm accepts to a single vertex. This vertex will be the 1-sink. Similarly, the halting configurations at which the algorithm rejects are contracted to a single vertex which will be the 0-sink. The initial configuration will be the source. If there are two edges leaving a vertex (s, σ) which lead to the same vertex of G, contract the vertex (s, σ) and its unique successor. The variable assigned to the resulting vertex is the variable assigned to the successor. Let G' be the resulting graph. Observe that each vertex of G' except for the sinks has out-degree two and it is assigned one of the variables. Moreover, since the graph G is acyclic, the graph G' is also acyclic.

We claim that G' is a syntactic ℓ-BDD representing the function f_n. Let P be a path from the source to a sink in G'. Note that each vertex of G' lies on at least one such path. The path P corresponds to execution of the algorithm for some faulty input. Since the algorithm tests each variable at most ℓ times for the considered faulty input, each variable is assigned to at most ℓ vertices of the path P. The proof that G' represents the function f_n is the same as in Proposition 1.

Since the number of configurations of the algorithm is bounded by $S \cdot 2^{k \log n + O(1)} = O(n^k)$, where S is the number of possible states of the algorithm, the size of the syntactic ℓ-BDD G' does not exceed $O(n^k)$.

■

3 Upper Bound for excl_N^d

In this section, we design a logspace algorithm computing the function excl_N^d. The algorithm terminates and tests each input bit at most twice even if the input is faulty. The vertices of an input d-uniform hypergraph H are considered to be ordered and identified with the numbers $1, \ldots, N$. The d-tuples $(A_1, \ldots, A_d)$ with $A_1 < A_2 < \ldots < A_d$ correspond to the edges of H. The edges of the hypergraph H are considered to be lexicographically ordered.

Let us now briefly sketch the main idea of the algorithm. The core of the algorithm is a procedure `CLIQUE` which accepts a d-uniform hypergraph H if and only if it is comprised by a hyperclique and isolated vertices. The hypergraph H is comprised by a hyperclique and isolated vertices if there exists a subset $V \subseteq \{1, \ldots, N\}$ such that the edges of H are precisely all the d-element subsets of V. Note that if $|V| < d$, then such a hypergraph H contains no edges at all. We explain the idea behind the procedure for $d = 2$: find the smallest non-isolated vertex X and its smallest neighbor Y. Note that $Y > X$. Next, verify that the neighbors of X (different from Y) are precisely the neighbors of Y (different from X). Observe that the input graph is formed by a clique and isolated vertices if and only if the subgraph induced by the vertices $Y, \ldots, N$ has this property. The property is then repeatedly verified when replacing Y with X. In this way, each edge of an input graph is tested at most twice.

If $d \geq 3$, we proceed similarly except for that we use vertices contained in edges together with the smallest non-isolated vertex X instead of the neighbors of X. The structure of the entire algorithm is then the following: First, the algorithm finds the lexicographically smallest edge e of H. Second, it verifies that the smallest $d-1$ vertices of e are contained in exactly $N/2-(d-1)$ edges of H, i.e., the potential hyperclique of H has the correct order. Finally, the procedure `CLIQUE` is invoked to verify that the structure of H is as required.

This section is structured as follows: first, we design the procedure for $d = 2$. The case $d = 2$ has to be handled separately since in the general case the procedure for d-uniform hypergraphs invokes the procedure for $(d-1)$-uniform hypergraphs. In addition, the case $d = 2$ is conceptually simpler than the general case, thus it is easier to explain first the main ideas just for $d = 2$. Next, we design the procedure for $d \geq 3$. In Subsection 3.3, we present the final algorithm and construct polynomial-size syntactic 2-BDDs representing excl_N^d.

3.1 Basic case

As mentioned above, we first consider the case of ordinary graphs ($d = 2$):

Lemma 1 *There is an* $(3 \log N + O(1))$*-space algorithm which accepts an* N*-vertex graph and an integer* V *if and only if* G *consists of a clique whose smallest vertex is* V *and the remaining vertices of* G *are isolated. In addition, the algorithm has the following properties:*

- *each edge of* G *is tested at most twice, and*
- *each edge* e*,* $e \leq (V, N)$*, is tested at most once.*

The space complexity and the bounds on the number of tests of input bits are preserved and the algorithm terminates even if the input is faulty. In addition, the algorithm can be modified, without increasing its space complexity, so that it outputs the vertices of the clique which are different from V *in the increasing order.*

Proof: We first explain how the algorithm works. When describing the algorithm, we will be referring to Figure 1 which contains its pseudocode. The algorithm uses two auxiliary $\lceil \log N \rceil$-bit variables A and B and the bits of the input are accessed through the predicate `edge`. At the beginning, we check that all the vertices $1, \ldots, V - 1$ are isolated (lines 1 and 2). In the main loop (lines 3–10), it is verified that the induced subgraph G' of G formed by the vertices $V, \ldots, N$ is formed by a clique whose smallest vertex is V and its remaining vertices are isolated. Note that the content of V is altered at the end of each iteration of the main loop. In the rest, let us write V^0 for the initial value stored in V.

We now describe the main loop in more detail. First, the smallest neighbor A of V in G' is found (lines 3–5). During this stage, it is also checked that all the vertices $V + 1, \ldots, A - 1$ are isolated in G' (line 5). If all the vertices of G' are isolated, the while-cycle is left with $A = N + 1$ and G is accepted (line 6). Otherwise, we test whether G' has the following property (lines 7 and 8): *Each vertex* B*,* $A < B \leq N$*, is adjacent to* A *if and only if it is adjacent to* V*.* If G' does not have this property, then the vertices V, A and a counterexample vertex B form an induced subpath of length three and we can reject. If V and A have the same neighbors, then G' is of the desired form if and only if the subgraph G'' of G induced by the vertices $A, A + 1, \ldots, N$ consists of a clique whose smallest vertex is A and

```
Input: a graph G of order N
       a vertex V

Auxiliary variables: A,B

    function independent: Boolean;
      for B:=A+1 to N do
        if edge(A,B) then return false;
      return true;

    for A:=1 to V-1 do
      if not independent then reject;
restart:
    A:=V+1;
    while not edge(V,A) and A<=N do
      if not independent then reject else A:=A+1;
    if A>N then accept;
    for B:=A+1 to N do
      if edge(V,B)<>edge(A,B) then reject;
    V:=A
    goto restart
```

Figure 1: The algorithm from Lemma 1.

its remaining vertices are isolated. So, we can replace V by A (line 9) and execute the main loop again (line 10).

The algorithm clearly accepts if and only if G consists of a clique whose smallest vertex is V and its remaining vertices are isolated. Besides the two auxiliary $\lceil \log N \rceil$-bit variables A and B, the content of the variable $\lceil \log N \rceil$-bit variable V is also modified. Hence, the space complexity is $3 \log N + O(1)$ bits. The space complexity is clearly preserved and the algorithm always terminates even if the input is faulty.

We now analyze the number of tests of the edges of G. The edges smaller than $(V, V+1)$ are accessed only in the function `independent` called on line 2 and each such edge is tested at most once. Let us fix V and let A be its smallest neighbor greater than V in G. The following edges are tested in the main loop on lines 4–9:

- each edge between $(V+1, V+2)$ and $(A-1, N)$ exactly once (line 5)
- each edge (V, x) for $x = V+1, \ldots, N$ exactly once (lines 4 and 8)
- each edge (A, x) for $x = A+1, \ldots, N$ exactly once (line 8)

Hence, each edge e, $e \geq (V, V+1)$, incident with one of the vertices $V, \ldots, A$ is tested exactly once in the main loop (for a fixed V). Since in the next loop, the present V is replaced by $A > V$, the only edges tested twice during the entire algorithm are those incident with a vertex V, $V > V^0$, for which the main loop is executed. In particular, all the edges e, $e \leq (V, N)$, are tested at most once. Observe that the above reasoning remains unchanged if the input is faulty.

It remains to show how to modify the algorithm to output the vertices of the clique different from V^0 in the increasing order. Since the vertices forming the clique are precisely those stored in V when the main loop is executed, it is enough just to output the new value of V between lines 9 and 10 (recall that the value V is increasing during the execution of the algorithm). The value of V should be output at the end of the main loop in order to skip outputting V^0.

■

3.2 General case

We now extend the procedure from Subsection 3.1 to d-uniform hypergraphs, $d \geq 3$. The procedure `CLIQUE` for d-uniform hypergraphs invokes itself for $(d-1)$-uniform ones. In the proof of Lemma 2, the following notation is used: If H is a d-uniform hypergraph and V is a vertex of it, then $H[V,*,\dots,*]$ is the $(d-1)$-uniform hypergraph whose vertex set is formed by the vertices greater than V and a $(d-1)$-element subset A is an edge of $H[V,*,\dots,*]$ if and only if $\{V\} \cup A$ is an edge of H. In the recursive instance, the vertices of $H[V,*,\dots,*]$ are considered to be numbered from 1 starting at the vertex $V+1$.

Lemma 2 *Let $d \geq 2$ be a fixed integer. There exists a procedure* `CLIQUE` *which accepts a d-uniform N-vertex hypergraph H together with a sequence $V_1,\dots,V_{d-1}$ of its vertices if and only if the hypergraph H is formed by a hyperclique whose smallest vertices are $V_1,\dots,V_{d-1}$ and the vertices of H which are not contained in the hyperclique are isolated. Moreover, the procedure has the following properties:*

- *the procedure terminates,*
- *each edge of H is tested at most twice,*
- *each edge e of H, $e \leq (V_1,\dots,V_{d-1},N)$, is tested at most once,*
- *the space complexity of the algorithm is $\frac{d^2+3d-4}{2}\log N + O(1)$ bits, and*
- *the procedure outputs the vertices of the hyperclique larger than V_{d-1} in the increasing order.*

The first four properties are preserved even if the input is faulty.

Proof: As in the proof of Lemma 1, we first describe the procedure `CLIQUE` and we then show that it has the required properties. The pseudocode of the procedure for $d \geq 3$ can be found in Figure 2. If $d = 2$, the algorithm described in Lemma 1 is used instead.

First, let us make few comments on the syntax used in the pseudocode. The bits of the input are accessed through the d-ary predicate `edge`. The macro `independent(A,*,...,*)` returns true if H contains no edge whose smallest vertex is A. This macro can be easily implemented as $d-1$ nested for-cycles.

Line 3 of the procedure is quite tricky: it is not just a recursive call of the procedure `CLIQUE` but rather initiating an instance of the procedure running in parallel. The new instance of the procedure `CLIQUE` receives as parameters the hypergraph $H[V_1,*,\dots,*]$ together with the vertices $V_2,\dots,V_{d-1}$ (the

```
Procedure CLIQUE( H, V[1], ..., V[d-1] );

Input: a d-uniform hypergraph H of order N, d>=3
       V[1], V[2], ..., V[d-1]

Auxiliary variables: V[d], A

macro independent: Boolean;

for A:=1 to V[1]-1 do
  if not independent(A, *, ..., *) then reject;
restart:
CLIQUE ( H[V[1],*,...,*], V[2], ..., V[d-1] ) >> PIPE;
if PIPE.closed then
  for A:=V[1]+1 to N do
    if not independent(A, *, ..., *) then reject;
  return;
V[d]:=PIPE.head; PIPE.pop;
A:=V[d]+1;
while not PIPE.closed do
  while A<PIPE.head do
    if edge( V[2], ..., V[d-1], V[d], A ) then reject;
    A:=A+1;
  if not edge ( V[2], ..., V[d-1], V[d], A ) then reject;
  A:=A+1; PIPE.pop;
while A<=N do
  if edge( V[2], ..., V[d-1], V[d], A ) then reject;
  A:=A+1;
while V[1]<V[2]-1 do
  V[1]:=V[1]+1;
  if not independent(V[1], *, ..., *) then reject;
V[1]:=V[2]; V[2]:=V[3]; ...; V[d-2]:=V[d-1]; V[d-1]:=V[d];
output V[d];
goto restart;
```

Figure 2: The procedure CLIQUE from Lemma 2.

vertices $V_2 - V_1, \ldots, V_{d-1} - V_1$ in the numbering used in $H[V_1, *, \ldots, *]$). In particular, the new instance of the procedure CLIQUE is initiated for a $(d-1)$-uniform hypergraph. If $d = 3$, the procedure from Lemma 1 is initiated instead of CLIQUE. The original instance and the new instance are connected through a pipe PIPE (a first-in first-out queue). The output of the new instance is stored in the pipe and it is accessed by the original instance using the following methods (we view the pipe as a data object):

- `PIPE.head`
  This method returns the value of the number stored at the head of the pipe queue.

- `PIPE.pop`
  This method removes from the pipe one element (the head of the pipe queue).

- `PIPE.closed`
  This method returns true if all the elements have been popped out from the pipe queue and the new instance of the procedure has finished, i.e., all the elements have been read and no new elements can be added.

The two procedures run in parallel in the following sense: The original one proceeds until a method of the pipe is called. Then, the new procedure is executed until the result of the method is known. At this moment, the new one is interrupted and the original one proceeds until it reaches a method of the pipe which cannot be completed without continuing execution of the new procedure. At this moment, the original one is interrupted and the new one is resumed. The executions of the procedures alternate in this way until the new one is finished (or the input is rejected). Note that the value stored at the head of the queue is determined by the status and the memory content of the new procedure and thus we do not need any additional space to maintain the pipe if the algorithm proceeds as described above. Since the depth of the recursion is at most $d-1$ (the uniformity of the hypergraph is always decreased by one), we can view the state of the computation as a $(d-1)$-ary vector of the states of the invoked procedures. In particular, the number of states of the procedure CLIQUE is independent of the input.

We are now ready to explain how the procedure works. The procedure uses two auxiliary $\lceil \log N \rceil$-bit variables V_d and A. At the beginning, we check that all the vertices $1, \ldots, V_1 - 1$ are isolated (lines 1 and 2). In the main loop (lines 3–25), the subhypergraph H' of H induced by the vertices $V_1, \ldots, N$ is verified to be formed by a hyperclique whose smallest $d-1$ vertices are $V_1, \ldots, V_{d-1}$ and its remaining vertices are isolated. Recall that if H' contains no edges, then it is of the required form (and the new instance of the procedure `CLIQUE` outputs no vertices). The content of the variables $V_1, \ldots, V_{d-1}$ is changed at the end of each iteration of the main loop. Similarly as in the proof of Lemma 1, $V_1^0, \ldots, V_{d-1}^0$ denote the initial values stored in the variables $V_1, \ldots, V_{d-1}$.

If the new instance of the procedure outputs no vertices and H is of the required form, then the hyperclique of H can contain no vertices except for $V_1, \ldots, V_{d-1}$. In particular, H must be formed by isolated vertices only. This case is handled on lines 4–7. Otherwise, the new instance of the procedure outputs vertices of the (possible) hyperclique of H greater than V_{d-1} in the increasing order. The smallest vertex of this hyperclique greater than V_{d-1} is stored in V_d (line 8). Let further Γ be the set of the vertices different from V_d which have been output. On lines 9–19, we check for each vertex A, $V_d < A \leq N$, that the hypergraph H contains the edge $(V_2, \ldots, V_d, A)$ if and only if $A \in \Gamma$. Note that $A \in \Gamma$ if and only if $H[V_1, *, \ldots, *]$ contains an edge $(V_2, \ldots, V_{d-1}, A)$ (which is the case if and only if $(V_1, \ldots, V_{d-1}, A)$ is an edge of H).

Observe now that the hypergraph H is of the required form if and only if each of the following holds:

- the $(d-1)$-uniform hypergraph $H[V_1, *, \ldots, *]$ is formed by a hyperclique on the vertex set $\{V_2, \ldots, V_d\} \cup \Gamma$ and its remaining vertices are isolated,
- each vertex V, $V_1 < V < V_2$, is isolated in H, and
- the d-uniform subhypergraph H' of H induced by the vertices greater or equal to V_2 consists of a hyperclique with the vertex set $\{V_2, \ldots, V_d\} \cup \Gamma$ and its remaining vertices are isolated.

The first condition has been already verified in the new instance of the procedure `CLIQUE`. The second condition is verified on lines 20–22. It remains to verify the third one. Since H' contains an edge $(V_2, \ldots, V_d, A)$ if and only if $A \in \Gamma$, the third condition is equivalent to the following:

- H' is formed by the hyperclique whose smallest $d-1$ vertices are $V_2, \ldots, V_d$ and its remaining vertices are isolated.

We verify this condition by replacing the variables $V_1, \dots, V_{d-1}$ by $V_2, \dots, V_d$ (line 23) and running the main loop again (line 25) with the new content of the variables $V_1, \dots, V_{d-1}$. On line 24, we output the vertex V_d which is (if H is of the required form) the smallest vertex contained in the hyperclique larger than V_{d-1}.

We have verified that the procedure accepts the hypergraph H if and only if H is formed by a hyperclique whose smallest vertices are $V_1, \dots, V_{d-1}$ and the vertices of H not contained in the hyperclique are isolated. The procedure also outputs the vertices of the hyperclique larger than V_{d-1} in the increasing order. Let us now estimate the number of tests of input bits. The only edges tested in the main loop, started for $V_1, \dots, V_{d-1}$, are the following:

- edges e, $(V_1, V_1+1, \dots, V_1+d-1) \le e \le (V_1, \dots, V_{d-1}, N)$
  Each of them is tested at most once in the recursive call.

- edges e, $(V_1, \dots, V_{d-1}, N) < e \le (V_1, N-(d-2), \dots, N)$
  Each of them is tested at most twice in the recursive call.

If the recursive call outputs some vertices, then the following edges are tested in addition:

- edges $e = (V_2, \dots, V_d, A)$ with $V_d < A \le N$
  Each of them is tested at most once on one of lines 13, 15 or 18.

- edges e, $(V_1+1, \dots, V_1+d) \le e < (V_2, V_2+1, \dots, V_2+d-1)$
  Each of them is tested once on line 22.

If the recursive call outputs no vertices, then each edge e, $e \ge (V_1+1, \dots, V_1+d)$, is tested exactly once (line 6).

The next claim can be established by induction on the number of the remaining iterations of the main loop. The only tests performed in the main loop including the tests from the recursive calls are the following:

- each edge e, $(V_1^0, V_1^0+1, \dots, V_1^0+d-1) \le e \le (V_1^0, \dots, V_{d-1}^0, N)$ is tested at most once, and

- each edge e, $e > (V_1^0, \dots, V_{d-1}^0, N)$ is tested at most twice.

Since the only edges tested before the main loop are the edges smaller than $(V_1^0, V_1^0+1, \dots, V_1^0+d-1)$ and each of them is tested at most once (line 6), the estimate on the number of tests from the statement of the lemma readily follows. It is routine to verify that we have not used assumption on the consistency of the input in our argumentation and thus the same bounds also hold for faulty inputs. Similarly, the procedure always terminates.

It remains to establish the bound on the space complexity of the procedure. The number a_d of auxiliary $\lceil \log N \rceil$-bit variables is computed by induction on d. If $d = 2$, Lemma 1 yields $a_2 = 3$. Assume now that $d \geq 3$. Besides the auxiliary variables needed in the macro `independent`, there are $d+1$ auxiliary $\lceil \log N \rceil$-bit variables $V_1, \ldots, V_d$ and A. Note that since the content of the variables $V_1, \ldots, V_{d-1}$ is altered, they have to be considered when estimating the space complexity of the procedure. An induced sub-hypergraph of H passed to the recursive call is determined by the value of V_1. However, since the value of V_1 is preserved until the pipe is closed, this does not result in additional space requirements. We do also not need any additional space to maintain the pipe as explained when describing the procedure. Finally, the macro `independent` is executed only when there is no recursive copy of `CLIQUE` running. In particular, we can utilize the space dedicated for the recursive calls of `CLIQUE` for storing the $d-1$ auxiliary $\lceil \log N \rceil$-bit variables needed in the macro. This leads us to the following equality for a_d:

$$a_d = (d+1) + \max\{d-1, a_{d-1}\} = (d+1) + a_{d-1} =$$

$$(d+1) + \frac{(d-1)^2 + 3(d-1) - 4}{2} = \frac{d^2 + 3d - 4}{2} .$$

Hence, the space complexity of the procedure `CLIQUE` is $\frac{d^2+3d-4}{2} \log N + O(1)$ bits. Again, we have not used the assumption on the consistency of the input, and therefore the bound holds for the case when the input is faulty, too.

■

3.3 Main results

We are now ready to design a log-space algorithm for the function excl_N^d:

Theorem 1 *Let $d \geq 2$ be a fixed integer. There is an algorithm which accepts a d-uniform N-vertex hypergraph H if and only if the hypergraph H is formed by a hyperclique of order $N/2$ and the vertices of H not contained in the hyperclique are isolated. Moreover, the algorithm has the following properties (even if the input is faulty):*

```
Input: a d-uniform hypergraph H of order N, d>=3

Auxiliary variables: V[1], ..., V[d], K

 1 for V[1]:=1 to N do
 2   for V[2]:=V[1]+1 to N do
 3     ...
 4     for V[d]:=V[d-1]+1 to N do
 5       if edge( V[1], ..., V[d] ) then
 6         break all for-cycles;
 7 if not broken then reject;
 8 K=1;
 9 while V[d]<N do
10   V[d]:=V[d]+1;
11   if edge( V[1], ..., V[d] ) then K:=K+1;
12 if K <> N/2-(d-1) then reject;
13 CLIQUE(H, V[1], ..., V[d-1]);
```

Figure 3: The algorithm from Theorem 1.

- *the algorithm terminates,*
- *each edge of H is tested at most twice, and*
- *the space complexity of the algorithm is* $\frac{d^2+3d-4}{2}\log N + O(1)$ *bits.*

Proof: The pseudocode of the algorithm can be found in Figure 3. The algorithm tests the variables corresponding to edges in increasing order until it finds the first edge $(V_1, \ldots, V_d)$ present in H (lines 1–6). If H contains no edges at all, the algorithm rejects (line 7). Otherwise, the algorithm verifies that the number of vertices A such that H contains the edge $(V_1, \ldots, V_{d-1}, A)$ is precisely $N/2-(d-1)$ (lines 8–12). If this is not the case, the algorithm rejects. Finally, the procedure `CLIQUE` from Lemma 2 is applied for $V_1, \ldots, V_{d-1}$ (line 13).

If the procedure `CLIQUE` rejects, then the hypergraph H is clearly not of the required form. If it accepts, then H is formed by a hyperclique containing vertices $V_1, \ldots, V_{d-1}$ and its remaining vertices are isolated. Since there are exactly $N/2-(d-1)$ vertices A such that the edge $(V_1, \ldots, V_{d-1}, A)$ is contained in H, the order of the hyperclique is $N/2$ and the algorithm should accept. Note that we actually test twice that the vertices $1, \ldots, V_1 - 1$ are isolated: for the first time when searching for the smallest edge, and for the second time in the procedure `CLIQUE`.

The only edges tested by the algorithm before the call of the procedure `CLIQUE` are those smaller or equal to $(V_1, \ldots, V_{d-1}, N)$. Hence, each edge is tested at most twice by Lemma 2 even in the faulty environment. Since the preprocessing steps requires only $d+1$ auxiliary $\lceil \log N \rceil$-bit variables, the space demands of the algorithm are dominated by space needed for the procedure `CLIQUE`. Therefore, the space complexity does not exceed $\frac{d^2+3d-4}{2} \log N + O(1)$ bits by Lemma 2. Finally, the algorithm always terminates, again, by Lemma 2.

■

Proposition 2 and Theorem 1 immediately yield:

Theorem 2 *Let $d \geq 2$ be a fixed integer. There exists a syntactic 2-BDD representing the function excl_N^d which has at most $O(N^{\frac{d^2+3d-4}{2}})$ nodes.*

4 Lower bound for excl_N^d

We modify the lower bound proof (Theorem 6.2.6 [12]) on the size of 1-BDDs representing the function excl_N^d for $d = 2$ to the case of hypergraphs. Note that the bound of Theorem 3 is slightly better than the bound presented in [12] for $d = 2$.

Theorem 3 *The size of an FBDD representing the function excl_N^d is at least $\binom{N}{N/2}$ for every $d \geq 2$ and every even integer $N \geq 2d$.*

Proof: Let N and d be fixed integers throughout the proof and G be an FBDD representing the function excl_N^d. Let $\mathcal{H}$ be the set of all the $\binom{N}{N/2}$ hypergraphs H accepted by G. For $H \in \mathcal{H}$, we define v_H to be the first node on the computation path for H such that for each vertex x of the hyperclique of H, at least one edge containing x has been tested (including the test at the node v_H).

We claim that if H and H' are different hypergraphs from $\mathcal{H}$, then $v_H \neq v_{H'}$. Assume the opposite. Let P be the path in G comprised by the computation path for H to the node $v_H = v_{H'}$ and the computation path for H' from the node $v_H = v_{H'}$. Since G is a free binary decision diagram, no edge can be tested twice on the path P. In particular, the path P corresponds to some hypergraph H_P which is accepted by G.

Let A_H and $A_{H'}$ be the set of the vertices of the hyperclique in H and H', respectively. By the choice of the node v_H, for each vertex x of A_H, at least one edge containing x has been tested on P before v_H (inclusively). Therefore, the degree of each vertex of A_H in H_P is at least one. Let x_0 be the vertex of $A_{H'}$ such that the first edge containing x_0 is tested at $v_{H'}$. Because of the choice of $v_{H'}$ and x_0, all the edges of H' containing x_0 are tested on the path P from the node $v_{H'}$ (inclusively). In particular, the degree of each vertex of $A_{H'}$ in H_P is at least one (because each such vertex is contained together with x_0 in at least one edge).

We can now conclude that the degree of each vertex of $A_H \cup A_{H'}$ in H_P is at least one. Therefore, H_P has less than $N/2$ isolated vertices and it should be rejected by G — a contradiction. We have shown that all the nodes v_H, $H \in \mathcal{H}$, are different. This directly implies that the size of G is at least $|\mathcal{H}| = \binom{N}{N/2}$.

■

The bound presented in Theorem 3 is of order $2^{\Omega(n^{1/d})}$ where $n = \binom{N}{d} = \Theta(N^d)$ is the number of input bits. Since it is not hard to construct FBDDs representing the function excl_N^d of size $2^{O(N \log(N^d))} = 2^{O(dn^{1/d} \log(n^{1/d}))}$, a significantly better lower bound cannot be established for excl_N^d.

5 Conclusion

The function excl_N^d is not the only hypergraph function which was suggested as a possible candidate for separation of polynomial-size ℓ-BDDs and $(\ell+1)$-BDDs. In the monograph [12], the function exreg_N^d which is described in the sequel is considered (Problem 7.7): The input is formed by $\binom{N}{d}$ bits corresponding to possible edges of a d-uniform hypergraph H on N vertices. The function $\mathrm{exreg}_N^d(H)$ is true if the hypergraph H is $\lfloor \binom{N}{d-1}/2 \rfloor$-regular, i.e., each vertex of H is contained in exactly $\lfloor \binom{N}{d-1}/2 \rfloor$ edges. It is not hard to construct a polynomial-size d-BDD for exreg_N^d, but the existence of a polynomial-size $(d-1)$-BDDs for exreg_N^d is unsettled:

Problem 1 *Are there polynomial-size $(d-1)$-BDDs representing* exreg_N^d*?*

Let us remark that the exponential lower on the size of $(d-1)$-BDDs representing exreg_N^d for $d = 2$ can be found in [8].

Another Boolean function of the similar kind defined for d-uniform hypergraphs is a "perfect matching" function. Again, the input is formed by $\binom{N}{d}$ bits corresponding to possible edges of a d-uniform hypergraph H. The function $\mathrm{match}_N^d(H)$ is true if H is 1-regular, i.e., each vertex is contained in exactly one edge of H. It is not hard to prove an exponential lower bound on the size of $(d-1)$-BDDs for the function match_N^d if $d = 2$. However, if $d \geq 3$, the situation becomes more difficult:

Problem 2 *Are there polynomial-size $(d-1)$-BDDs representing* match_N^d*?*

Acknowledgement

This research was conducted while the author was visiting Simon Fraser University in May 2004. The author would like to thank his host, Ladislav Stacho, and Veselin Jungić for providing an excellent atmosphere during his entire stay in Burnaby, BC. He would also like to thank Ondřej Pangrác for careful reading a preliminary version of the manuscript.

References

[1] M. Ajtai, L. Babai, P. Hajnal, J. Komlós, P. Pudlák, V. Rödl, E. Szemerédi, G. Turán: *Two lower bounds for branching programs*, in: Proc. 18th ACM Symposium on Theory of Computing (STOC), 1986, 30–38.

[2] A. Borodin, A. A. Razborov, R. Smolensky: *On lower bounds for read-k-times branching programs*, Computational Complexity 3, 1993, 1–18.

[3] A. Cobham: *The recognition problem for the set of perfect squares*, in: Proc. 7th IEEE Symposium on Foundations of Computer Science (FOCS), 1966, 78–87.

[4] J. Kára, D. Kráľ: *Optimal Free Binary Decision Diagrams for Computation of EAR_n*, in: Proc. 27th International Symposium Mathematical Foundations of Computer Science (MFCS), LNCS vol. 2420, 2002, 411–422.

[5] P. Pudlák, S. Žák: *Space complexity of computations*, Math. Inst., ČSAV, Prague, 1983, 30 pp.

[6] A. A. Razborov: *Lower bounds for deterministic and nondeterministic branching programs*, in: Proc. 8th International Symposium Fundamentals of Computation Theory (FCT), LNCS vol. 529, 1991, 47–61.

[7] M. Sauerhoff: *On nondeterminism versus randomness for read-once branching programs*, available as technical report in Electronic Colloquium on Computational Complexity (ECCC) TR97-030, 1997.

[8] J. Simon, M. Szegedy: *A new lower bound theorem for read only once branching programs and its applications*, in: Advances in computational complexity (J. Cai, ed.), DIMACS Series in Discrete Mathematics vol. 13, 1993, 183–193.

[9] J. S. Thathachar: *On separating the read-k-times branching program hierarchy*, in: Proc. 30th ACM Symposium on Theory of Computing (STOC), 1998, 653–662.

[10] I. Wegener: *The complexity of Boolean functions*, B. G. Teubner, 1987.

[11] I. Wegener: *On the complexity of branching programs and decision trees for clique functions*, J. of ACM 35(2), 1988, 461–471.

[12] I. Wegener: *Branching Programs and Binary Decision Diagrams — Theory and Applications*, SIAM Monographs on Discrete Mathematics and Applications 4, 2000.

[13] S. Žák: *An exponential lower bound for one-time-only branching programs*, in: Proc. 11th International Symposium on Mathematical Foundations of Computer Science (MFCS), LNCS vol. 176, 1984, 562–566.