Quasipolynomial simulation of DNNF by a non-determinstic read-once branching program

Igor Razgon

Department of Computer Science and Information Systems, Birkbeck, University of London
igor@dcs.bbk.ac.uk

Abstract. We prove that DNNFs can be simulated by Non-deterministic Read-Once Branching Programs (NROBPs) of quasi-polynomial size. As a result, all the exponential lower bounds for NROBPs immediately apply for DNNFs.

1 Introduction

Decomposable Negation Normal Forms (DNNFs) [3] is a well known formalism in the area of propositional knowledge compilation notable for its efficient representation of CNFs with bounded structural parameters. The DNNFs lower bounds are much less understood. For example, it has been only recently shown that DNNFs can be exponentially large on (monotone 2-) CNFs [2]. Prior to that, it was known that on monotone functions DNNFs are not better than monotone DNNFs [6]. Hence all the lower bounds for monotone circuits apply for DNNFs. However, using monotone circuits to obtain new DNNF lower bounds is hardly an appropriate methodology because, in light of [2], on monotone functions, DNNFs are much weaker than monotone circuits.

In this paper we show that DNNFs are strongly related to Non-deterministic Read-Once Branching Programs (NROBPs) that can be thought as Free Binary Decision Diagrams (FBDDs) with OR-nodes. In particular, we show that a DNNF can be transformed into a NROBP with a quasi-polynomial increase of size. That is, all the exponential lower bounds known for NROBPs (see e.g. [5, 8]) apply for DNNFs. As NROBPs can be linearly simulated by DNNFs (using a modification of the simulation of FBDDs by DNNFs proposed in [4]), we believe that the proposed result makes a significant progress in our understanding of complexity of DNNFs. Indeed, instead of trying to establish exponential lower bounds directly for DNNFs, we can now do this for NROBPs , which are much better understood from the lower bound perspective.

In the proposed simulation, we adapt to unrestricted DNNFs the approach that was used in [1] for quasi-polynomial simulation of *decision* DNNFs by FBDDs. For the adaptation, we find it convenient to represent NROBPs in a form where variables carry no labels and edges are labelled with literals. In particular, each input node u of the DNNF is represented in the resulting NROBP as an edge labelled with the literal of u and these are *the only edges* that are labelled (compare with [1] where the labelling is 'pertained' to OR nodes, which is impossible for unrestricted DNNFs where the OR nodes can have an arbitrary structure).

The most non-trivial aspect of the simulation is the need to transform an AND of two NROBPs Z_1 and Z_2 into a single NROBP. Following [1], this is done by putting Z_1

'on top' of Z_2. However, this creates the problem that Z_1 becomes unusable 'outside' this construction (see Section 4.1. and, in particular, Figure 2 of [1] for illustration of this phenomenon). Similarly to [1], we address this problem by introducing multiple copies of Z_1.

Formal statement of the result. A DNNF Z^* is a directed acyclic graph (DAG) with many roots (nodes of in-degree 0) called *input* nodes and one leaf (node of out-degree 0) called the *output* node. The input nodes are labelled with literals, the rest are AND, and OR nodes such that each AND node has the *decomposability* property defined as follows. Let us define $Var(u)$ for a node u of Z^* as the set of variables x such that Z^* has a path from a node labelled by x to u. Then, if u is an AND node of Z^* and v and w are two different in-neighbours of u then $Var(v) \cap Var(w) = \emptyset$. Let Z^*_u be the subgraph of Z^* induced by a node u and all the nodes from which u can be reached. Then the function $F[Z^*_u]$ computed by Z^*_u is defined as follows. If u is an input node then $F[Z^*_u] = x$, where x is the literal labelling u. If u is an OR or an AND node with in-neighbours $v_1, \ldots v_q$ then $F[Z^*_u] = F[Z^*_{v_1}] \vee \ldots \vee F[Z^*_{v_q}]$, or $F[Z^*_u] = F[Z^*_{v_1}] \wedge \ldots \wedge F[Z^*_{v_q}]$, respectively. The function $F[Z^*]$ computed by Z^* is $F[Z^*_{out}]$, where out is the output node of Z^*. In the rest of the paper we assume that the AND nodes of Z^* are binary. This assumption does not restrict generality of the result since an arbitrary DNNF can be transformed into one with binary AND nodes with a quadratic increase of size.

A Non-deterministic Read-once Branching Program (NROBP) is a DAG Z with one root (and possibly many leaves). Some edges of Z are labelled with literals of variables in the way that each variable occurs at most once on each path P of Z. We denote by $A(P)$ the set of literals labelling the edges of a path P of Z. To define a function $F[Z]$ computed by Z, let us make a few notational agreements. First, we define a truth assignment to a set of variables as the set of literals of these variables that become true as result of the assignment. For example, the assignment $\{x_1 \leftarrow true, x_2 \leftarrow false, x_3 \leftarrow true\}$ is represented as $\{x_1, \neg x_2, x_3\}$. For a function F on a set Var of variables, we say that an assignment S of Var *satisfies* F is $F(S) = true$. Now, let S be an assignment of variables labelling the edges of Z. Then S satisfies $F[Z]$ if and only if there is a root-leaf path P of Z with $A(P) \subseteq S$. A DNNF and a NROBP for the same function are illustrated on Figure 1.

Remark. The above definition of NROBP is equivalent to FBDD with OR-nodes in the sense that each of them can simulate the other with a linear increase of size.

Our main result proved in the next section is the following.

Theorem 1. *Let Z^* be a* DNNF *with m nodes computing a function F of n variables. Then F can be computed by a* NROBP *of size $O(m^{\log n + 2})$.*

2 Proof of Theorem 1

This section is organised as follows. We first present a transformation of a DNNF Z^* into a graph Z, then state two auxiliary lemmas about properties of special subgraphs of Z, their proofs postponed to Section 2.1, and then prove Theorem 1.

The first step of the transformation is to fix one in-coming edge of each AND-node u of Z^* as the *light* edge of u. This is done as follows. Let u_1 and u_2 be two in-neighbours

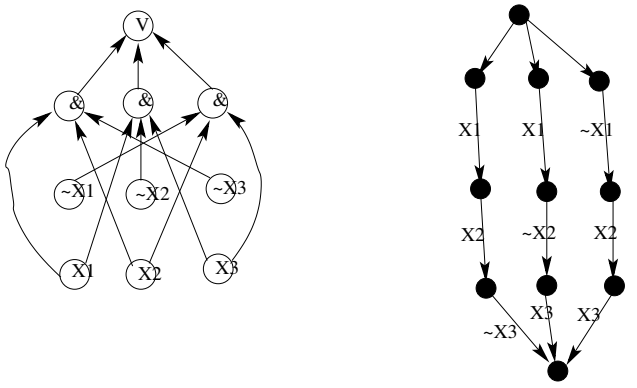

Fig. 1. DNNF and NROBP for function $(x_1 \wedge x_2 \wedge \neg x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3)$

of u such that $|Var(u_1)| \leq |Var(u_2)|$. Then (u_1, u) is the *light edge* of u and (u_2, u) is the *heavy edge* of u. (Of course if both in-neighbours of u depend on the same number of variables then u_1 and u_2 can be chosen arbitrarily.) We say that an edge (v, w) of Z^* is *a* light edge if w is an AND-node and (v, w) is its light edge. Let P be a path from u to the output node out of Z^*. Denote the set of light edges of P by $le(P)$. Denote by $LE(u)$ the set of all such $le(P)$ for a $u - out$ path P.

Now we define a graph Z consisting of the following nodes.

- (u, le, in) for all $u \in V(Z^*)$ (recall that if G is a graph, $V(G)$ denotes the set of nodes of G) and $le \in LE(u)$. The 'in' in the third coordinate stands for 'internal' to distinguish from the 'leaf' nodes defined in the next item.
- For each input node u of Z^* and for each $le \in LE(u)$, Z has a node (u, le, lf) where 'lf' stands for 'leaf'. We say that (u, le, lf) is the *leaf corresponding to* (u, le, in).

When we refer to a node of Z with a single letter, we use bold letters like $\mathbf{u}, \mathbf{v}$ to distinguish from nodes u, v of Z^*. We denote by $mnode(\mathbf{u}), coord(\mathbf{u}), type(\mathbf{u})$, the respective components of $\mathbf{u}$, that is $\mathbf{u} = (mnode(\mathbf{u}), coord(\mathbf{u}), type(\mathbf{u}))$. We also call the components *the main node* of $\mathbf{u}$, *the coordinate* of $\mathbf{u}$ and the *type* of $\mathbf{u}$. The nodes of Z whose type is in are *internal* nodes and the nodes whose type is lf are *leaf* nodes. The leaf nodes are not necessarily leaves of Z but rather leaves of special subgraphs of Z that are important for the proof.

Setting the environment for definition of edges of Z. We explore the nodes u of Z^* topologically sorted from the input to the output and *process* each internal node $\mathbf{u}$ of Z with $u = mnode(\mathbf{u})$. In particular, we introduce out-neighbours of $\mathbf{u}$, possibly, together with labelling of respective edges, the set of nodes $Leaves(\mathbf{u})$, and a subgraph $Graph(\mathbf{u})$ of Z which will play a special role in the proof. The detailed description of *processing* of $\mathbf{u}$ is provided below.

- Suppose that u is an input node. Let y be the literal labelling u in Z^* and let $\mathbf{u}'$ be the leaf corresponding to $\mathbf{u}$.

1. Introduce an edge $(\mathbf{u}, \mathbf{u}')$ and label this edge with y.
2. Set $Leaves(\mathbf{u}) = \{\mathbf{u}'\}$.
3. Define $Graph(\mathbf{u})$ as having node set $\{\mathbf{u}, \mathbf{u}'\}$ and the edge $(\mathbf{u}, \mathbf{u}')$.

– Suppose that u is an OR node. Let $v_1, \ldots v_q$ be the in-neighbours of u in Z^*. Let $\mathbf{v_1}, \ldots, \mathbf{v_q}$ be the internal nodes of Z with $v_1, \ldots, v_q$ being the respective main nodes and with $coord(\mathbf{v_i}) = coord(\mathbf{u})$ for all $1 \leq i \leq q$.
  1. Introduce edges $(\mathbf{u}, \mathbf{v_1}), \ldots, (\mathbf{u}, \mathbf{v_q})$.
  2. Set $Leaves(\mathbf{u}) = Leaves(\mathbf{v_1}) \cup \ldots \cup Leaves(\mathbf{v_q})$.
  3. $Graph(\mathbf{u})$ is obtained from $Graph(\mathbf{v_1}) \cup \ldots \cup Graph(\mathbf{v_q})$ by adding node $\mathbf{u}$ plus the edges $(\mathbf{u}, \mathbf{v_1}), \ldots, (\mathbf{u}, \mathbf{v_q})$.
– Suppose u is an AND node. Let u_1, u_2 be two in-neighbours of u in Z^* and assume that the edge (u_1, u) is the light one. Let $\mathbf{u_1}, \mathbf{u_2}$ be two internal nodes of Z whose respective main nodes are u_1 and u_2 and $coord(\mathbf{u_1}) = coord(\mathbf{u}) \cup \{(u_1, u)\}$ and $coord(\mathbf{u_2}) = coord(\mathbf{u})$.
  1. Introduce edges $(\mathbf{u}, \mathbf{u_1})$ and $(\mathbf{w}, \mathbf{u_2})$ for each $\mathbf{w} \in Leaves(\mathbf{u_1})$.
  2. Set $Leaves(\mathbf{u}) = Leaves(\mathbf{u_2})$.
  3. $Graph(\mathbf{u})$ is obtained from $Graph(\mathbf{u_1}) \cup Graph(\mathbf{u_2})$ by adding node $\mathbf{u}$ and the edges described in the first item.

Remark. Let us convince ourselves that the nodes $\mathbf{v_1}, \ldots, \mathbf{v_q}$, and $\mathbf{u_1}, \mathbf{u_2}$ with the specified coordinates indeed exist. Indeed, suppose that u is an OR-node of Z^* and let v be an in-neighbour of u. Let P be a path from u to the output node of Z^*. Then $le((v, u) + P) = le(P)$ confirming possibility of choice of nodes $\mathbf{v_1}, \ldots, \mathbf{v_q}$. Suppose that u is an AND-node and let (u_1, u) and (u_2, u) be the light and heavy edges of u respectively. For a P as before, $le((u_1, u) + P) = \{(u_1, u)\} \cup le(P)$ and $le((u_2, u) + P)) = le(P)$ confirming that the nodes $\mathbf{u_1}$ and $\mathbf{u_2}$ exist. Thus the proposed processing is well-defined.

Lemma 1. *Let $\mathbf{u} \in V(Z)$ with $type(\mathbf{u}) = in$ and let $u = mnode(\mathbf{u})$. Then the following statements hold.*

1. *$Graph(\mathbf{u})$ is a* DAG.
2. *$\mathbf{u}$ is the (only) root of $Graph(\mathbf{u})$ and $Leaves(\mathbf{u})$ is the set of leaves of $Graph(\mathbf{u})$.*
3. *If u is an* OR*-node and $\mathbf{v_1}, \ldots, \mathbf{v_q}$ are as in the description of processing of $\mathbf{u}$ then each root-leaf path P of $Graph(\mathbf{u})$ is of the form $(\mathbf{u}, \mathbf{v_i}) + P'$ where P' is a root-leaf path of $Graph(\mathbf{v_i})$.*
4. *Suppose u is an* AND *node and let $\mathbf{u_1}, \mathbf{u_2}$ be as in the description of processing of $\mathbf{u}$. Then each root-leaf path P of $Graph(\mathbf{u})$ is of the form $(\mathbf{u}, \mathbf{u_1}) + P_1 + (\mathbf{w}, \mathbf{u_2}) + P_2$, where P_1, P_2 are root-leaf paths of $Graph(\mathbf{u_1})$ and $Graph(\mathbf{u_2})$, respectively and $\mathbf{w}$ is the last node of P_1.*
5. *$Var(\mathbf{u}) = Var(u)$ where $Var(\mathbf{u})$ is the set of all variables labelling the edges of $Graph(\mathbf{u})$.*
6. *$Graph(\mathbf{u})$ is read-once (each variable occurs at most once on each path).*

It follows from the first, second, and the last statements of Lemma 1 that $Graph(\mathbf{u})$ is a NROBP. Therefore, we can consider the function $F[Graph(\mathbf{u})]$ computed by $Graph(\mathbf{u})$.

Lemma 2. *For each* $\mathbf{u} \in V(Z)$ *with* $type(\mathbf{u}) = in$, $F[Graph(\mathbf{u})] = F[Z^*_u]$ *where* $u = mnode(\mathbf{u})$ *and* Z^*_u *is as defined in the first paragraph of formal statement part of the introduction.*

Proof of Theorem 1. Let out be the output node of Z^* and $\mathbf{out} = (out, \emptyset, in)$ be a node of Z. $Graph(\mathbf{out})$ is a NROBP by Lemma 1 and, by Lemma 2, it computes function $F(Z^*_{out})$. By definition, $Z^*_{out} = Z^*$ and hence $Graph(\mathbf{out})$ computes the same function as Z^*.

To upper-bound the size of Z^*, observe that for each $\mathbf{u} \in V(Z)$, $|coord(\mathbf{u})| \leq \log n$. Indeed, let us represent $coord(\mathbf{u})$ as $(u_1, u'_1), \ldots, (u_q, u'_q)$, a sequence of edges occurring in this order on a path of Z^*. Then each (u_i, u'_i) is the light edge of an AND-node u'_i. By the decomposability property of DNNF, $|Var(u_i)| \leq |Var(u'_i)|/2$. Also, since Z^* has a path from u'_i to u_{i+1}, $|Var(u'_i)| \leq |Var(u_{i+1})|$. Applying this reasoning inductively, we conclude that $|Var(u_1)| \leq |Var(u'_q)|/2^q$. Since $|Var(u_1)| \geq 1$ and $|Var(u'_q)| \leq n$, it follows that $|coord(\mathbf{u})| = q \leq \log n$. Thus $coord(\mathbf{u})$ is a set of light edges of Z^* of size at most $\log n$. Since there is at most one light edge per element of Z^*, there are at most m light edges in total. Thus the number of possible second coordinates for a node of Z is $\sum_{i=1}^{\log n} \binom{m}{i} \leq m^{\log n + 1}$. As the number of distinct first and third coordinates is at most m and 2, respectively, the result follows. ■

2.1 Proofs of auxiliary lemmas for Theorem 1

Proof of Lemmas 1 and 2 requires two more auxiliary lemmas.

Lemma 3. *Let* $\mathbf{u} \in V(Z)$ *with* $type(\mathbf{u}) = in$ *and let* $u = mnode(\mathbf{u})$. *Then for each* $\mathbf{v} \in V(Graph(\mathbf{u}))$, $coord(\mathbf{u}) \subseteq coord(\mathbf{v})$. *Moreover, if* $type(\mathbf{v}) = lf$ *then* $coord(\mathbf{u}) = coord(\mathbf{v})$ *if and only if* $\mathbf{v} \in Leaves(\mathbf{u})$.

Proof. By induction on nodes $\mathbf{u}$ of Z according to the topological sorting of the nodes $u = mnode(\mathbf{u})$ of Z^* from input to output nodes. That is if v is an neighbour of u then for any node $\mathbf{v}$ with $v = mnode(\mathbf{v})$ the lemma holds by the induction assumption.

If u is an input node then $V(Graph(\mathbf{u}))$ consists of $\mathbf{u}$ and the leaf corresponding to $\mathbf{u}$, hence the first statement holds by construction. Otherwise, $V(Graph(\mathbf{u}))$ consists of $\mathbf{u}$ itself and the union of all $V(Graph(\mathbf{v}))$, where, following the description of processing of $\mathbf{u}$, $\mathbf{v}$ is one of $\mathbf{v_1}, \ldots, \mathbf{v_q}$ if u is an OR-node and $\mathbf{v}$ is either $\mathbf{u_1}$ or $\mathbf{u_2}$ if u is an AND-node. For each such $\mathbf{v}$ it holds by definition that $coord(\mathbf{u}) \subseteq coord(\mathbf{v})$. That is, each node $\mathbf{w} \neq \mathbf{u}$ of $Graph(\mathbf{u})$ is in fact a node of such a $Graph(\mathbf{v})$. By the induction assumption, $coord(\mathbf{v}) \subseteq coord(\mathbf{w})$ and hence $coord(\mathbf{u}) \subseteq coord(\mathbf{w})$ as required.

Using the same inductive reasoning, we show that for each $\mathbf{w} \in Leaves(\mathbf{u})$, $coord(\mathbf{w}) = coord(\mathbf{u})$. This is true by construction if u is an input node. Otherwise, $Leaves(\mathbf{u})$ is defined as the union of one or more $Leaves(\mathbf{v})$ such that $coord(\mathbf{v}) = coord(\mathbf{u})$ by construction. Then, letting $\mathbf{v}$ be such that $\mathbf{w} \in Leaves(\mathbf{v})$ we deduce, by the induction assumption that $coord(\mathbf{w}) = coord(\mathbf{v}) = coord(\mathbf{u})$.

It remains to prove that for each $\mathbf{w} \in V(Graph(\mathbf{u})) \backslash Leaves(\mathbf{u})$ such that $type(\mathbf{w}) = lf$, $coord(\mathbf{u}) \subset coord(\mathbf{w})$. This is vacuously true if u is an input node. Otherwise, the induction assumption can be straightforwardly applied as above if $\mathbf{w} \in Graph(\mathbf{v})$ for

some $\mathbf{v}$ as above and $\mathbf{w} \notin Leaves(\mathbf{v})$. The only situation where it is not the case is when u is an AND node and $\mathbf{v} = \mathbf{u_1}$ where $\mathbf{u_1}$ is as in the description of processing of an AND node. In this case $coord(\mathbf{u}) \subset coord(\mathbf{u_1})$ by construction and, since $coord(\mathbf{u_1}) \subseteq coord(\mathbf{w})$, by the first statement of this lemma, $coord(\mathbf{u}) \subset coord(\mathbf{w})$. ■

Lemma 4. *1. For each internal $\mathbf{u} \in V(Z)$, the out-going edges of $\mathbf{u}$ in Z are exactly those that have been introduced during processing of $\mathbf{u}$.*
2. For each $\mathbf{v} \in V(Z)$ with $type(\mathbf{v}) = lf$, the out-degree of $\mathbf{v}$ in Z is at most 1. Moreover the out-degree of $\mathbf{v}$ is 0 in each $Graph(\mathbf{u}')$ such that $\mathbf{v} \in Leaves(\mathbf{u}')$.
3. Let $\mathbf{u}$ be an internal node and let $(\mathbf{w_1}, \mathbf{w_2})$ be an edge where $\mathbf{w_1} \in V(Graph(\mathbf{u}))\setminus Leaves(\mathbf{u})$. Then $(\mathbf{w_1}, \mathbf{w_2})$ is an edge of $Graph(\mathbf{u})$.

Proof. The first statement follows by a direct inspection of the processing algorithm. Indeed, the only case where an edge $(\mathbf{u}, \mathbf{v})$ might be introduced during processing of a node $\mathbf{u}' \neq \mathbf{u}$ is where $mnode(\mathbf{u}')$ is an AND node. However, in this case $type(\mathbf{u})$ must be lf in contradiction to our assumption.

Consider the second statement. Consider an edge $(\mathbf{v}, \mathbf{w})$ such that $type(\mathbf{v}) = lf$. Suppose this edge has been created during processing of a node $\mathbf{u}$. Then $u = mnode(\mathbf{u})$ is an AND-node. Further, let $\mathbf{u_1}, \mathbf{u_2}$ be as in the description of processing of $\mathbf{u}$. Then $\mathbf{v} \in Leaves(\mathbf{u_1})$ and $\mathbf{w} = \mathbf{u_2}$. By construction, $coord(\mathbf{w}) = coord(\mathbf{u})$ and by Lemma 3, $coord(\mathbf{v}) = coord(\mathbf{u_1})$. Hence, by definition of $\mathbf{u_1}$, $coord(\mathbf{w}) \subset coord(\mathbf{v})$. Suppose that $\mathbf{v} \in Leaves(\mathbf{u}')$ for some internal $\mathbf{u}' \in V(Z)$. Then, by Lemma 3, $coord(\mathbf{v}) = coord(\mathbf{u}')$ and hence $coord(\mathbf{w}) \subset coord(\mathbf{u}')$. It follows from Lemma 3 that $\mathbf{w}$ is not a node of $Graph(\mathbf{u}')$ and hence $(\mathbf{v}, \mathbf{w})$ is not an edge of $Graph(\mathbf{u}')$. Thus the out-degree of $\mathbf{v}$ in $Graph(\mathbf{u}')$ is 0.

Continuing the reasoning about edge $(\mathbf{v}, \mathbf{w})$, we observe that $coord(\mathbf{w}) = coord(\mathbf{v})\setminus \{(u_1, u)\}$ where $u_1 = mnode(\mathbf{u_1})$. Notice that all the edges of $coord(\mathbf{v}) = coord(\mathbf{u_1})$ lie on a path from u_1 to the output node of Z^* and (u_1, u) occurs first of them. Due to the acyclicity of Z^*, (u_1, u) is uniquely defined (in terms of $\mathbf{v}$) and hence so is $coord(\mathbf{w})$. Furthermore, as specified above, $mnode(\mathbf{w}) = mnode(\mathbf{u_2})$ which is the neighbour u other than u_1. Since (u_1, u) is uniquely defined, u and u_1 are uniquely defined as its head and tail and hence so is $mnode(\mathbf{u_2})$. Finally, by construction, we know that $\mathbf{w}$ is an internal node. Thus all three components of $\mathbf{w}$ are uniquely defined and hence so is $\mathbf{w}$ itself. That is, $\mathbf{v}$ can have at most one neighbour.

The third statement is proved by induction analogous to Lemma 3. The statement is clearly true if $u = mnode(\mathbf{u})$ is an input node of Z^* because then $Graph(\mathbf{u})$ has only one edge. Assume this is not so. If $\mathbf{w_1} = \mathbf{u}$ then the statement immediately follows from the first statement of this lemma and the definition of $Graph(\mathbf{u})$. Otherwise, $\mathbf{w_1} \in V(Graph(\mathbf{v}))$ where $\mathbf{v}$ is as defined in the proof of Lemma 3. If $\mathbf{w_1} \notin Leaves(\mathbf{v})$ then, by the induction assumption, $(\mathbf{w_1}, \mathbf{w_2})$ is an edge of $Graph(\mathbf{v})$ and hence of $Graph(\mathbf{u})$. This may be not the case only if $\mathbf{u}$ is an AND node and $\mathbf{w_1} \in Leaves(\mathbf{u_1})$ ($\mathbf{u_1}$ and $\mathbf{u_2}$ are as in the description of processing of $\mathbf{u}$). By definition of $Graph(\mathbf{u})$, $(\mathbf{w_1}, \mathbf{u_2})$ is an edge of $Graph(\mathbf{u})$ and, according to the previous paragraph, $\mathbf{w_1}$ does not have other outgoing edges. Thus it remains to assume that $\mathbf{w_2} = \mathbf{u_2}$ and the statement holds by construction. ■

Proof sketch of Lemma 1 All the statements except 3 and 4 are proved by induction like in Lemma 3. If $u = mnode(\mathbf{u})$ is an input node then $Graph(\mathbf{u})$ is a labelled edge for which all the statements of this lemma are clearly true. So, we assume below that u is either an OR-node or an AND-node.

Statement 1. Assume that $Graph(\mathbf{u})$ does have a directed cycle C. Since $Graph(\mathbf{u})$ is loopless by construction, C contains at least one node $\mathbf{w} \neq \mathbf{u}$. By construction, $\mathbf{w}$ is a node of some $Graph(\mathbf{v})$ where $\mathbf{v}$ is as defined in the proof of Lemma 3. Then C intersects with $Leaves(\mathbf{v})$. Indeed, if we assume the opposite then, applying the last statement of Lemma 4 inductively starting from $\mathbf{w}$, we conclude that all the edges of $\mathbf{C}$ belong to $Graph(\mathbf{v})$ in contradiction to its acyclicity by the induction assumption. Now, C does not intersect with $Leaves(\mathbf{u})$ because they have out-degree 0 by Lemma 4. Thus if C intersects with $Graph(\mathbf{v})$ then $Leaves(\mathbf{v})$ cannot be a subset of $Leaves(\mathbf{u})$. This is only possible if $\mathbf{u}$ is an AND-node and $\mathbf{v} = \mathbf{u_1}$ ($\mathbf{u_1}$ and $\mathbf{u_2}$ are as in the description of the processing of $\mathbf{u}$). Let $\mathbf{w}' \in Leaves(\mathbf{u_1})$ be a node of C. By construction, $\mathbf{u_2}$ is an out-neighbour of $\mathbf{w}'$ and by Lemma 4, $\mathbf{w}$ does not have other out-neighbours in Z. Thus $\mathbf{u_2}$ is the successor of $\mathbf{w}'$ in C and hence C intersects with $Graph(\mathbf{u_2})$ while $Leaves(\mathbf{u_2}) \subseteq Leaves(\mathbf{u})$ in contradiction to what we have just proved. Thus C does not exists.

Statement 2. It is easy to verify by induction that $Graph(\mathbf{u})$ has a path from $\mathbf{u}$ to the rest of vertices, hence besides $\mathbf{u}$, $Graph(\mathbf{u})$ does not have any other roots. Now $\mathbf{u}$ itself is a root by the acyclicity proved above. Since vertices of $Leaves(\mathbf{u})$ have out-degree 0 in $Graph(\mathbf{u})$, clearly, they are all leaves. Suppose that some $\mathbf{w} \in Graph(\mathbf{u}) \setminus Leaves(\mathbf{u})$ is a leaf of $\mathbf{u}$. By construction, $\mathbf{w} \neq \mathbf{u}$ and hence $\mathbf{w}$ is a node of some $Graph(\mathbf{v})$ as above. Then $\mathbf{w}$ is a leaf of $Graph(\mathbf{v})$ because the latter is a subgraph of $Graph(\mathbf{u})$ and hence, by the induction assumption, $\mathbf{w} \in Leaves(\mathbf{v})$. Hence, as in the previous paragraph, we need $\mathbf{v}$ such that $Leaves(\mathbf{v}) \nsubseteq Leaves(\mathbf{u})$ and we conclude as above that $\mathbf{v} = \mathbf{u_1}$. But then $\mathbf{u_2}$ is an out-neighbour of $\mathbf{w}$ and hence $\mathbf{w}$ cannot be a leaf of $\mathbf{u}$, a contradiction showing that the leaves of $Graph(\mathbf{u})$ are precisely $Leaves(\mathbf{u})$.

Important remark. In the rest of the section we use $\mathbf{u}$ and the root of $Graph(\mathbf{u})$ as well as $Leaves(\mathbf{u})$ and the leaves of $Graph(\mathbf{u})$ interchangeably without explicit reference to statement 2 of this lemma.

Statements 3 and 4 Suppose that u is an OR node and let P' be the suffix of P starting at the second node $\mathbf{v}$ of P. By statement 1 of Lemma 4, $\mathbf{v}$ is some $\mathbf{v_i}$ as in the description of processing of $\mathbf{u}$. Hence vertices of $Leaves(\mathbf{v}) \subseteq Leaves(\mathbf{u})$ have out-degree 0 in $Graph(\mathbf{u})$ (statement 2 of Lemma 4) and do not occur in the middle of P'. Applying statement 3 of Lemma 4 inductively to P' starting from $\mathbf{v}$, we observe that P' is a path of $Graph(\mathbf{v})$. The last node of P' is a leaf of $Graph(\mathbf{v})$ because it is a leaf of $Graph(\mathbf{u})$. Thus statement 3 holds.

Suppose that u is an AND-node. Then by statement 1 of Lemma 4, the second node of P is $\mathbf{u_1}$. Hence, one of $Leaves(\mathbf{u_1})$ must be an intermediate node of P. Indeed, otherwise, by inductive application of statement 3 of Lemma 4, the suffix of P starting at $\mathbf{u_1}$ is a path of $Graph(\mathbf{u_1})$. In particular, the last node $\mathbf{w}'$ of P is a node of $Graph(\mathbf{u_1})$. However, by Lemma 3, $coord(\mathbf{w}') = coord(\mathbf{u}) \subset coord(\mathbf{u_1})$. Hence, by Lemma 3, $coord(\mathbf{w}')$ cannot belong to $Graph(\mathbf{u_1})$, a contradiction. Let $\mathbf{w} \in Leaves(\mathbf{u_1})$ be the first such node of P. By inductive application of the last statement fo Lemma 4, the

subpath P_1 of P between $\mathbf{u_1}$ and $\mathbf{w}$ is a root-leaf path of $Graph(\mathbf{u_1})$. By construction and the second statement of Lemma 4, $\mathbf{u_2}$ is the successor of $\mathbf{w}$ in P. Let P_2 be the suffix of P starting at $\mathbf{u_2}$. Arguing as for the OR case we conclude that P_2 is a root-leaf path of $\mathbf{u_2}$. Thus statement 4 holds.

Statement 5. We apply induction, taking into account that $Var(u)$ is the union of all $Var(v)$ where v is an in-neighbour of and $Var(\mathbf{u})$ is the union of all $Var(\mathbf{v})$ where $\mathbf{v}$ is as defined in the proof of Lemma 3. The details are omitted due to space constraints.

Statement 6. Let P be a root-leaf path of $Graph(\mathbf{u})$. Suppose $u = mnode(\mathbf{u})$ is an OR-node. Then $P = (\mathbf{u}, \mathbf{v_i}) + P'$, the notation as in statement 3. P' is read-once by the induction assumption and the edge $(\mathbf{u}, \mathbf{v_i})$ is unlabelled by construction. Hence P is read-once. If u is an AND-node then $P = (\mathbf{u}, \mathbf{u_1}) + P_1 + (\mathbf{w}, \mathbf{u_2}) + P_2$, all the notation as in statement 4. P_1 and P_2 are read-once by the induction assumption, edges $(\mathbf{u}, \mathbf{u_1})$ and $(\mathbf{w}, \mathbf{u_2})$ are unlabelled. The variables of P_1 and of P_2 are respective subsets of $Var(\mathbf{u_1})$ and $Var(\mathbf{u_2})$ equal to $Var(u_1)$ and $Var(u_2)$, respectively, by statement 5 which, in turn, do not intersect due to the decomposability property of AND nodes of Z^*. Hence the variables of P_1 and P_2 do not intersect and P is read-once. ■

Proof of Lemma 2 By induction as in Lemma 3. If $u = mnode(\mathbf{u})$ is an input node then $F[Z^*_u] = x$ where x is the literal labelling u and $Graph(\mathbf{u})$ is a single edge labelled with x, hence $F[Graph(\mathbf{u})] = x$.

In the rest of the proof, we assume that the set of variables of all the considered functions is Var, the set of variables of Z^*. Introduction of redundant variables will simplify the reasoning because we can now make the induction step without modifying the considered set of variables.

Assume that u is an OR node. If S satisfies $F[Z^*_u]$ then there is an in-neighbour v of u such that S satisfies $F[Z^*_v]$. By construction there is an out-neighbour $\mathbf{v}$ of $\mathbf{u}$ such that $v = mnode(\mathbf{v})$. By the induction assumption, S satisfies $F[Graph(\mathbf{v})]$. Let P' be a $\mathbf{v} - Leaves(\mathbf{v})$ path such that $A(P') \subseteq S$. Then, by construction $(\mathbf{u}, \mathbf{v}) + P'$ is a $\mathbf{u} - Leaves(\mathbf{u})$ path with the edge $(\mathbf{u}, \mathbf{v})$ unlabelled. Hence $A(P) = A(P') \subseteq S$ and hence S satisfies $F[Graph(\mathbf{u})]$. Conversely, if S satisfies $F[Graph(\mathbf{u})]$ then there is a $\mathbf{u} - Leaves(\mathbf{u})$ path P with $A(P) \subseteq S$. By statement 3 of Lemma 1, $P = (\mathbf{u}, \mathbf{v_i}) + P'$, the notation as in the statement. $A(P') \subseteq A(P)$ and hence S satisfies $F[Graph(\mathbf{v_i})]$ and, by the induction assumption, S satisfies $F[Z^*_{v_i}]$ where $v_i = mnode(\mathbf{v_i})$. By definition of $\mathbf{v_i}$, v_i is an in-neighbour of u, hence S satisfies $F[Z^*_u]$.

Assume that u is an AND node. Let $\mathbf{u_1}, \mathbf{u_2}$ be as in the description of processing of $\mathbf{u}$ with $u_1 = mnode(\mathbf{u_1})$ and $u_2 = mnode(\mathbf{u_2})$. Suppose that S satisfies $F[Z^*_u]$. Then S satisfies both $F[Z^*_{u_1}]$ and $F[Z^*_{u_2}]$. Hence, by the induction assumption S satisfies $F[Graph(\mathbf{u_1})]$ and $F[Graph(\mathbf{u_2})]$. For each $i \in \{1, 2\}$, let P_i be a $\mathbf{u_i} - Leaves(\mathbf{u_i})$ path of $Graph(\mathbf{u_i})$ with $A(P_i) \subseteq S$. Let $\mathbf{w}$ be the last node of P_1. Then $P = (\mathbf{u}, \mathbf{u_1}) + P_1 + (\mathbf{w}, \mathbf{u_2}) + P_2$ is a $\mathbf{u} = Leaves(\mathbf{u})$ path with the edges $(\mathbf{u}, \mathbf{u_1})$ and $(\mathbf{w}, \mathbf{u_2})$ unlabelled. Hence $A(P) = A(P_1) \cup A(P_2) \subseteq S$ and thus S satisfies $F[Graph(\mathbf{u})]$. Conversely, suppose that S satisfies $F[Graph(\mathbf{u})]$ and let P be a $\mathbf{u} - Leaves(\mathbf{u})$ path with $A(P) \subseteq S$. Then by statement 4 of Lemma 1, $P = (\mathbf{u}, \mathbf{u_1}) + P_1 + (\mathbf{w}, \mathbf{u_2}) + P_2$, the notation as in the statement. Clearly, $A(P_1) \subseteq S$ and $A(P_2) \subseteq S$, hence S satisfies both $F[Z^*_{u_1}]$ and $F[Z^*_{u_2}]$ by the induction assumption and thus S satisfies $F[Z^*_u]$. ■

References

1. Paul Beame, Jerry Li, Sudeepa Roy, and Dan Suciu. Lower bounds for exact model counting and applications in probabilistic databases. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, Bellevue, WA, USA, August 11-15, 2013*, 2013.
2. Simone Bova, Florent Capelli, Stefan Mengel, and Friedrich Slivovsky. Expander cnfs have exponential DNNF size. *CoRR*, abs/1411.1995, 2014.
3. Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48(4):608–647, 2001.
4. Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res. (JAIR)*, 17:229–264, 2002.
5. Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*. Springer-Verlag, 2012.
6. Matthias P. Krieger. On the incompressibility of monotone DNFs. *Theory Comput. Syst.*, 41(2):211–231, 2007.
7. Umut Oztok and Adnan Darwiche. On compiling CNF into decision-dnnf. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, pages 42–57, 2014.
8. Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.