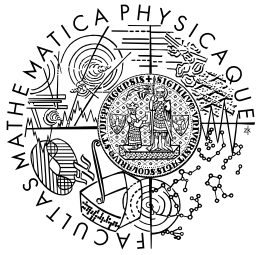

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jelena Glišić

Non-standard representations of Boolean functions for knowledge compilation

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis: prof. RNDr. Ondřej Čepek, Ph.D.

Study programme: Computer Science - Theoretical Computer Science

Study branch: Computer Science - Theoretical Computer Science

Prague 2024

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to express my gratitude to the supervisor of this thesis prof. RNDr. Ondřej Čepek, Ph.D. for continued guidance and advice during my thesis journey.

Title: Non-standard representations of Boolean functions for knowledge compilation

Author: Jelena Glišić

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: prof. RNDr. Ondřej Čepek, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: The nearest neighbor representation is a new representation of Boolean functions, introduced by Hajnal, Liu, and Turán. In this thesis, its properties within knowledge compilation are examined. In particular, results on its succinctness are shown, and some questions on the hardness of answering queries and performing transformations are answered.

Keywords: Boolean functions Knowledge representation languages Knowledge compilation

Contents

1. Introduction

Boolean functions are fundamental components in computer science, which often serve as the backbone of computation tasks and decision-making processes. Their significance extends across diverse domains including circuit design, artificial intelligence and cryptography. However, as the complexity of systems increases, the need for efficient representation and manipulation of Boolean functions becomes more and more important.

Knowledge compilation emerges as a critical area within this context, aiming to transform Boolean functions into more compact and tractable forms. This allows us to use techniques from logic, algorithms and data structures in order to bridge the gap between the expressive power of Boolean functions and the efficiency requirements of practical applications.

In this thesis, we are in particular interested in a representation of Boolean functions using nearest neighbors. We look at it through the lens of knowledge compilation, and determine whether this novel representation brings any new power to the way we view and use Boolean functions.

1.1 Outline

We begin by recalling elementary definitions and establishing notation to be used in the remainder of the thesis. In Chapter 2, we define the nearest neighbor representation of Boolean functions and restate the known results concerning this representation. Afterwards, in Chapter 3 we present an introduction to knowledge compilation. We define the main tools used in knowledge compilation. We show some simple already known results, and map out the goal of the final chapters. Finally, in Chapters 4 and 5, we place the nearest neighbor representation within the knowledge compilation map. We show original results, and discuss the advantages and limitation that they imply.

1.2 Notation and Definitions

Throughout the thesis, we make use of the following notation describing basic notions and operations in logic, set theory and algebra:

- the symbols $\vee$ and $\wedge$ denote the binary Boolean operations disjunction and conjunction, respectively, and the symbol $\neg$ denotes the unary Boolean negation operation;
- $\forall$ and $\exists$ denote the universal and existential quantifiers, respectively;
- the symbol $\oplus$ denotes the binary Boolean operation XOR
- the symbols $\cup, \cap$ and $\setminus$ denote set union, intersection and difference operations, respectively;
- the symbols $\subseteq$ and $=$ denote subset and equality relations, respectively;

- $\mathbb{N} = \{1, 2, 3, \ldots\}$ denotes the set of natural numbers and the letter n always denotes an element of $\mathbb{N}$;
- $\mathbb{R}$ denotes the set of real numbers;
- for any set $\mathcal{S}$, the symbol $\mathcal{S}^n$ denotes the Cartesian product $\underbrace{\mathcal{S} \times \mathcal{S} \times \cdots \times \mathcal{S}}_{n \text{ times}}$
- the symbol $\mathcal{B}$ denotes the set $\{0, 1\}$;
- by $\mathcal{B}^n$ we denote the Boolean hypercube;
- by $x_1, \ldots, x_n$ we denote the coordinates of $x \in \mathcal{B}^n$;
- by $d(x, y)$ for $x, y \in \mathbb{R}^n$ we denote the Euclidean distance in $\mathbb{R}^n$;
- by $d(x, \mathcal{A})$ for $x \in \mathbb{R}^n$ and $\mathcal{A} \subseteq \mathbb{R}^n$ we denote $\min\{d(x, y) \mid y \in \mathcal{A}\}$, i.e. the Euclidean distance between a vector and a set of vectors;
- by $d_H(x, y)$ for $x \in \mathcal{B}^n$ we denote the Hamming distance in the Boolean hypercube, i.e. the number of coordinates where the two Boolean vectors differ;
- by $d_H(x, \mathcal{A})$ for $x, y \in \mathcal{B}^n$ and $\mathcal{A} \subseteq \mathcal{B}^n$ we denote $\min\{d_H(x, y) \mid y \in \mathcal{A}\}$, i.e. the Hamming distance between a Boolean vector and a set of Boolean vectors;
- by $|x|$ for $x \in \mathcal{B}^n$ we denote the weight of x, i.e. the number of coordinates of x which are equal to 1;
- by $x \leq y$ for $x, y \in \mathcal{B}^n$ we denote the component-wise partial order on $\mathcal{B}$, i.e. $x \leq y$ if $\{i \mid x_i = 1\} \subseteq \{i \mid y_i = 1\}$;

We now provide definitions of some of the basic notions used throughout the thesis.

Definition 1. *A* Boolean function *is a function $\mathcal{B}^n \to \mathcal{B}$. We say that $x \in \mathcal{B}^n$ is* positive *vector or a* model *of f (resp.* negative *vector or a* non-model *of f) if $f(x) = 1$ (resp. $f(x) = 0$).*

Definition 2. *We call a Boolean function f a* symmetric function *if there exists a set $I_f \subseteq \{0, 1, \ldots, n\}$ such that $f(x) = 1$ if and only if $|x| \in I_f$.*

Definition 3. *The symmetric Boolean function with $I_f = \{i \text{ is odd} \mid 1 \leq i \leq n\}$ is called the* parity function *and we will denote it by PAR_n.*

Definition 4. *We call a Boolean function f a* threshold function *if there exist weights $w_1, \ldots, w_n \in \mathbb{R}$ and threshold $t \in \mathbb{R}$ such that $f(x) = 1$ if and only if $\sum_{i=1}^{n} w_i x_i \geq t$. By TH_n^t we denote the threshold functions with weights $w_1 = \cdots = w_n = 1$ and threshold t.*

In fact, one can observe that TH_n^t is the symmetric function f defined by $I_f = \{i \mid i \geq t\}$.

Definition 5. *The Boolean function $TH_n^{n/2}$ is called the* majority function *and we will denote it by MAJ_n.*

Definition 6. *A* graph *G is a pair (V, E), where the set V is called the* vertex *set and the set $E \subseteq \binom{V}{2}$ is called the* edge *set. We say that a vertex $u \in V$ is* adjacent *to vertex $v \in V$ if $\{u, v\} \in E$. A vertex v in a graph is* isolated *if it is not adjacent to any other vertex. A* walk *in graph is a sequence of vertices $v_1 v_2 \dots v_k$ where $v_i \in V$ for $1 \leq i \leq k$ and $\{v_i, v_{i+1}\} \in E$ for $1 \leq i \leq k-1$. We call a walk in which every vertex is visited exactly once a* path*. We say that a graph is* connected *if there is a walk between any two vertices. We say that two vertices are* connected *if there exists a walk in G between them. We call the equivalence classes of connected vertices the* connected components *of G. We say that $G' = (V', E')$ is a* subgraph *of G if $V' \subseteq V$ and $E' \subseteq E$. For $U \subseteq V$, we call the subgraph $(U, \{\{u, v\} \in E \mid u, v \in U\})$ the* subgraph induced by *U.*

Definition 7. *The* Boolean hypercube graph *is the graph whose vertex set is exactly the set $\mathcal{B}^n$ and in which there is an edge between $x \in \mathcal{B}^n$ and $y \in \mathcal{B}^n$ if and only if $d_H(x, y) = 1$.*

In the thesis, we will often use the terms Boolean hypercube and Boolean hypercube graph interchangeably. In particular, a neighbor of a vector x in the Boolean hypercube will be any vector that is adjacent to it in the Boolean hypercube graph.

2. Nearest Neighbor Representations of Boolean Functions

In this chapter we introduce the nearest neighbor representation of Boolean function, as defined by Hajnal et al. [2022]. We discuss the complexity of various functions with respect to this representation, and make remarks about modified versions of it.

2.1 Motivation and Definition

The nearest neighbor representation of Boolean functions is inspired by already established and well studied examples of nearest neighbor representations. Formally, for any classification of vectors in $\mathbb{R}^n$, a nearest neighbor representation may be defined. This is done by picking disjoint subsets of $\mathbb{R}^n$, each of which we call a *prototype* set for some class. Afterwards, any other vector may be classified using these subsets. In the simplest case, this is done by finding the prototype closest to the vector, and assigning it the class of this prototype. Nearest neighbor representation have been studied in the context of computational geometry (e.g. in Aurenhammer et al. [1991]), machine learning (e.g. in Zhang [2016]) and various other areas. When dealing with a nearest neighbor representation, the objective is usually to minimize the number of prototypes. In this section we define the nearest neighbor representation of Boolean functions, as well as some of its variants.

Definition 8 (Hajnal et al. [2022])**.** *A* nearest neighbor (NN) *representation of a Boolean function f is a pair of disjoint subsets (P, N) of $\mathbb{R}^n$ such that for every $a \in \mathcal{B}^n$*

- *if a is positive then there exists $b \in P$ such that for every $c \in N$ it holds that $d(a, b) < d(a, c)$,*
- *if a is negative then there exists $b \in N$ such that for every $c \in P$ it holds that $d(a, b) < d(a, c)$.*

The vectors in P (resp., N) are called positive (resp. negative) prototypes*. The size of the representation is $|P \cup N|$. The nearest neighbor complexity of f, $NN(f)$, is the minimum of the sizes of the NN representations of f.*

Analogously, we may define a Boolean nearest neighbor representation of a Boolean function.

Definition 9 (Hajnal et al. [2022])**.** *A* Boolean nearest neighbor (BNN) *representation of a Boolean function f is a pair of disjoint subsets (P, N) of $\mathcal{B}^n$ such that for every $a \in \mathcal{B}^n$*

- *if a is positive then there exists $b \in P$ such that for every $c \in N$ it holds that $d_H(a, b) < d_H(a, c)$,*

- *if a is negative then there exists $b \in N$ such that for every $c \in P$ it holds that $d_H(a,b) < d_H(a,c)$.*

The Boolean nearest neighbor complexity of f, $BNN(f)$, is the minimum of the sizes of the BNN representations of f.

We remark here, that there is in fact no difference between using Euclidean and Hamming distance when considering Boolean prototypes. This is due to the fact that that for any two vectors $x, y \in \mathcal{B}^n$, $d(x,y) = \sqrt{d_H(x,y)}$. Thus any BNN representation of a function f, is also an NN representation of the same function f.

Taking inspiration from machine learning, we may not only look for the nearest neighbor, but the nearest k neighbors and look for the majority class. Thus we may also define k-nearest neighbors.

Definition 10 (Hajnal et al. [2022])**.** *A* k-nearest neighbor (k-NN) *representation of a Boolean function f is a pair of disjoint subsets (P, N) of $\mathbb{R}^n$ such that for every $a \in \mathcal{B}^n$*

- *a is positive if and only if at least $\frac{k}{2}$ of the k vectors in $P \cup N$ closest to a belong to P.*

Here we assume that for every a, the k smallest distances of a from the prototypes are all smaller than the other $|P \cup N| - k$ distances from the prototypes. The k-nearest neighbor complexity, k-$NN(f)$, of f is the minimum of the sizes of the k-nearest neighbor representations of f.

2.2 Some Functions and Their Complexity

It is easy to see that any non-constant Boolean function requires at least two prototypes. We also make the following simple observation.

Observation. For any Boolean function f, it holds that

$$NN(f) \leq BNN(f) \leq 2^n$$

Proof. There are more vectors in $\mathbb{R}^n$ then in $\mathcal{B}^n$, and thus in general at most as many real as Boolean prototypes are needed to represent any function. The last inequality follows from the fact that we may take all Boolean vectors as prototypes. □

In this section, we discuss the complexity of some specific Boolean functions. We begin by several results from Hajnal et al. [2022].

Theorem 1. *The following statements hold:*

(a) For any symmetric function f, $NN(f) \leq n + 1$.

(b) For any threshold function f, $NN(f) = 2$.

Proof. We first prove the part (a). Let f be a symmetric function. Consider the prototypes $p_l = (\frac{l}{n}, \ldots, \frac{l}{n})$ for $l \in \{0, 1, \ldots, n\}$. Let $P = \{p_l \mid l \in I_f\}$ and $N = \{p_l \mid l \notin I_f\}$. We claim that (P, N) is a nearest neighbor representation of f. Let $y \in \mathcal{B}^n$ be such that $|y| = w$. Consider the hyperplane $\sum_{i=1}^n x_i = w$, defined by its normal vector $(1, 1, \ldots, 1)$. Then y lies on this hyperplane. Moreover, the line $\mathcal{L} = \{(t, t, \ldots, t) \in \mathbb{R}^n \mid t \in \mathbb{R}\}$ is orthogonal to it, since it is the extension of the normal vector of the hyperplane. Since p_w is exactly the intersection of the hyperplane and the line, it is in fact closer to y than any other $x \in \mathcal{L}$. Thus p_w is indeed the closest prototype to y, and the result follows.

As for part (b), let $w_1, \ldots, w_n$ be the weights of threshold function f and let t be its threshold. Consider the hyperplane $\mathcal{H} = \sum_{i=1}^n w_i x_i = t$ and let $\mathcal{L}$ be the line orthogonal to it. We first show that without loss of generality we may assume that no Boolean point lies on the hyperplane $\mathcal{H}$. Suppose that there is $y \in \mathcal{B}^n$ such that $y \in \mathcal{H}$. Let $z \in \mathcal{B}^n$ be the closest negative vector to $\mathcal{H}$. Then let $\varepsilon := \frac{d(\mathcal{H}, z)}{2}$. Consider now the hyperplane $\mathcal{H}' = \sum_{i=1}^n w_i x_i = t - \varepsilon$. Then $\mathcal{H}'$ describes the same threshold function f and $y \notin \mathcal{H}$.

Let x be the intersection of the hyperplane and $\mathcal{L}$. Let p be such that $p \in \mathcal{L}$, $\sum_{i=1}^n w_i p_i > t$ and $d(x, p) = 1$. Let q be such that $q \in \mathcal{L}$, $\sum_{i=1}^n w_i q_i < t$ and $d(x, q) = 1$. Then all negative Boolean vector lie on the same side of the hyperplane as q and are therefore closer to it than to p. Similarly for positive vectors and p. Thus picking p for the single positive prototype and q for the single negative prototype gives a nearest neighbor representation of f and the claim follows. □

Theorem 2. *The following statements hold:*

(a) $BNN(PAR_n) = 2^n$.

(b) If n *is odd then* $BNN(MAJ_n) = 2$ *and if* n *is even then* $BNN(MAJ_n) \leq \frac{n}{2} + 2$.

(c) $BNN(TH_n^{\lceil n/3 \rceil}) = 2^{\Omega(n)}$

Proof. We first show part (a). Let p be a positive prototype. Let x be such that $d_H(p, x) = 1$. Then $PAR_n(x) = 0$ and there must exist a negative prototype q closer to it than p. As $d_H(p, x) = 1$, it must be that $d_H(q, x) = 0$ and thus $x = q$. Since this must hold for all positive and negative prototypes, the claim follows.

For part (b), let us first assume that n is odd. Then we may pick the all-ones vector as the single positive prototype and the all-zeros vector as the single negative prototype and obtain a Boolean nearest neighbor representation of MAJ_n. Now we may assume that n is even. Then let us pick the all-zeros vector as the single negative prototype q and arbitrary $\frac{n}{2} + 1$ of the n vectors of weight $n - 1$ as the positive prototypes. Consider x such that $|x| = n/2$. Then $MAJ_n(x) = 1$ and moreover, $d_H(x, p) = \frac{n}{2} - 1$ for the closest positive prototype to x, p. This is due to the fact that out of the $\frac{n}{2} + 1$ positive prototypes, one must match the half of the coordinates where x has ones, and further, it must have exactly one zero in the other half of the coordinates. As $d_H(x, q) = \frac{n}{2}$, p is the closest prototype for x and x is classified correctly.

Let us now show part (c). Let $t := \lceil n/3 \rceil$. Consider a positive vector $x \in \mathcal{B}^n$ such that $|x| = t$, and the positive prototype closest to it, say p. We first claim

that $x \leq p$. Suppose for contradiction that it is not. Then there is an index $i \in \{1, \ldots, n\}$ such that $x_i = 1$ and $p_i = 0$. Let y be the vector obtained by flipping the i^{th} coordinate of x to 0. Then $|y| = t - 1$ and y is a negative vector. Let q be negative prototype closest to y. Then:

$$d_H(x, p) = d_H(y, p) + 1 > d_H(y, q) + 1 \tag{2.1}$$

since q must be closer to y than p. But on the other hand,

$$d_H(x, p) < d_H(x, q) \leq d_H(y, q) + 1 \tag{2.2}$$

since p must be closer to x than q. As (2.1) and (2.2) cannot both be true, we have arrived at a contradiction. By a symmetric argument, it can be shown that for any y of weight $t - 1$ and the closest negative prototype to it q, $q \leq y$.

We have now shown that for any x of weight t, there exists a negative prototype q such that $q \leq x$ and thus $d_H(x, q) \leq t$. So for a positive prototype p closest to x it must hold that $d_H(x, p) < t$ and thus $|p| < 2t$. In this case we say that such a p *covers* x.

We now count the prototypes. There are $\binom{n}{t}$ vectors of weight t. Each positive prototype of may cover at most $\binom{2t}{t}$ of them, since they are all of weight at most $2t$, and t changes of coordinates are needed to obtain a vector of weight t. Therefore we need at least

$$\frac{\binom{n}{t}}{\binom{2t}{t}} = 2^{\Omega(n)}$$

positive prototypes.

□

As we will discuss in the next chapter, many representation of Boolean functions are based on lists of Boolean vectors, such as $MODS$ in Darwiche and Marquis [2002] and SL in Čepek [2022]. Motivated by those results, we will only consider BNN as this language also consists of lists of Boolean vectors. Theorem 2 will prove to be crucial in proving many results about this representation, which we present in Chapters 4 and 5

3. Knowledge Compilation Map

This chapter serves as an introduction to the topic of knowledge compilation. We explain the particular measures used to compare one representation of Boolean functions to another. We set up the playing field for the next chapters, where we finally place BNN within the knowledge compilation map of Darwiche and Marquis [2002].

3.1 Standard Knowledge Representation Languages

We begin by defining some standard languages used in knowledge compilation, as defined in Darwiche and Marquis [2002]. We will later use these languages to explain how different representations are compared within the knowledge compilation map, and to build the intuition for the results that we will show in the final two chapters.

Definition 11. *A sentence in* Negation Normal Form (NNF) *is a rooted, directed acyclic graph (DAG) where each leaf node is labeled with true, false, X or $\neg X$, for some variable X; and each internal node is labeled with $\wedge$ or $\vee$ and can have arbitrarily many children. The size of a sentence Σ in NNF, denoted $|\Sigma|$, is the number of its DAG edges. Its height is the maximum number of edges from the root to some leaf in the DAG.*

We remark that an NNF can be thought of as a Boolean formula, where we are only allowed to negate variables.

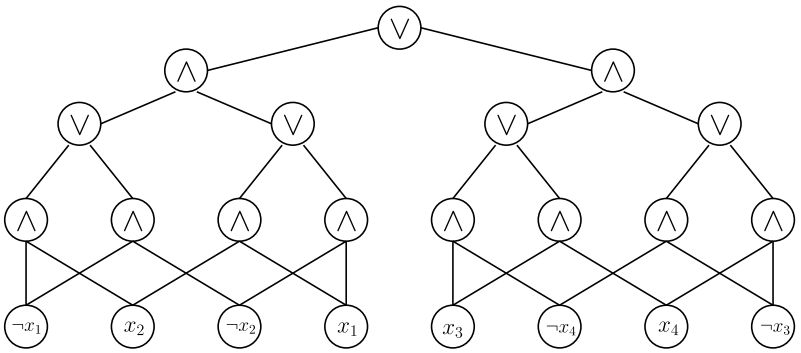

Figure 3.1: A sentence in the NNF language.

By restricting what the structure of an NNF looks like, we may define more languages. By restricting the height of the underlying DAG as well, we obtain CNF and DNF languages.

Definition 12. *We call an NNF sentence of height* 2 *which is rooted at an $\wedge$ node whose children are all $\vee$ nodes a* CNF sentence. *We call each child of the root a* clause. *We call a clause* consistent *if it does not contain both a variable and its negation. We denote the language of all such sentences by* CNF.

Definition 13. *We call an NNF sentence of height* 2 *which is rooted at an* $\vee$ *node whose children are all* $\wedge$ *nodes a* DNF sentence. *We call each child of the root a* term. *We call a term* consistent *if it does not contain both a variable and its negation. We denote the language of all such sentences by* DNF.

In order to introduce more languages, we define the notions of prime implicates and implicants.

Definition 14. *Let* C *be a clause and* f *be Boolean function. We say that* C *is an* implicate *of* f *if* $f(x) = 1 \implies C(x) = 1$ *for any* $x \in \mathcal{B}^n$. *Moreover, we call* C *a* prime implicate *if removing any literal from* C *results in a clause that is not an implicate of* f.

Definition 15. *Let* T *be a term and* f *be a Boolean function. We say that* T *is an* implicant *of* f *if* $T(x) = 1 \implies f(x) = 1$ *for any* $x \in \mathcal{B}^n$. *Moreover, we call* T *a* prime implicant *if removing any literal from* T *results in a term that is not an implicant of* f.

Definition 16. *For a Boolean function* f, *we call a CNF of* f *which contains exactly all prime implicates of* f *the* canonical CNF *of* f. *We denote the language where each function is represented by its canonical CNF by* PI.

Definition 17. *For a Boolean function* f, *we call a DNF of* f *which contains exactly all prime implicants of* f *the* canonical DNF *of* f. *We denote the language where each function is represented by its canonical DNF by* IP.

We now define some more languages which are obtained by imposing various structural restrictions on NNF. We denote by $Vars(\Sigma)$ the variables of the sentence Σ.

Definition 18. *We say that a NNF DAG is* decomposable *if for children of an and-node* $C_1, \ldots, C_n$ *it holds that* $Vars(C_i) \cap Vars(C_j) = \emptyset$, *for* $i \neq j$. *We denote the language of decomposable NNFs by* DNNF.

Definition 19. *We say that a NNF DAG is* deterministic *if for children of an or-node* $C_1, \ldots, C_n$ *it holds that* $C_i \wedge C_j \implies 0$, *for* $i \neq j$. *We denote the language of deterministic NNFs by* d-NNF. *We denote the language of decomposable and deterministic NNFs by* d-DNNF.

Definition 20. *We say that a NNF DAG is* smooth *if for children of an or-node* $C_1, \ldots, C_n$ *it holds that* $Vars(C_i) = Vars(C_j)$, *for* $i \neq j$. *We denote the language of smooth NNFs by* s-NNF. *We denote the language of decomposable, deterministic and smooth NNFs* sd-DNNF.

Definition 21. *We say that a node* N *in an NNF is a* decision node *if it is labeled with true, false, or is an or-node whose children are of the form* $X \wedge \alpha$ *and* $\neg X \wedge \beta$, *where* X *is a variable, and* α *and* β *are decision nodes. We call the the language of sentences rooted at decision nodes* BDD. *We denote by* FBDD *the language of decomposable BDDs. We denote by* $\text{OBDD}_<$ *the subset of BDD where a total ordering of decision variables is imposed. The union of all* $OBDD_<$ *is denoted by* OBDD.

In fact, BDDs are the so-called *binary decision diagrams*. For an example, see Figure 3.2. On the right, a BDD is depicted as an NNF. On the right, a more standard illustration of the BDD is shown. That is, the nodes of the new diagram correspond to the decision variables. The edges will then correspond to the decisions made, i.e. they will be labeled by either 0 or 1, depending on the value of the decision variable.

It is an easy observation that every NNF sentence that satisfies the decision property is also deterministic. Thus BDD$\subseteq$ d-NNF.

FBDDs are usually referred to as *read-once branching programs*. That is, in each from the root to a leaf in an FBDD, each variable appears at most once.

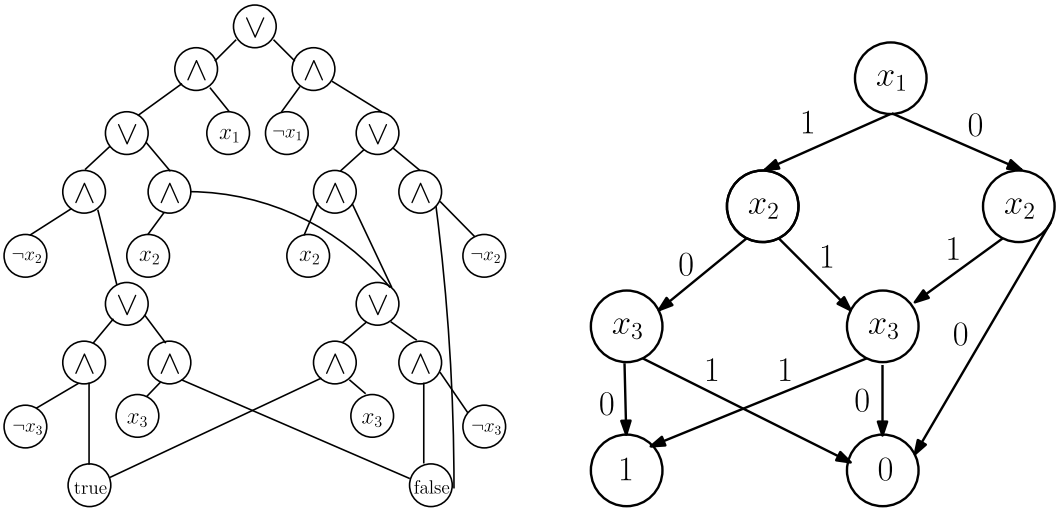

Figure 3.2: A sentence in the BDD language (left) and the corresponding binary decision diagram (right).

Definition 22. MODS *is the language at the intersection DNF and sd-NNF.*

We remark here that MODS corresponds exactly to the language where each Boolean function is represented by listing all of its models. Similarly, it is possible to define the language ¬MODS, by a list of all non-models.

Recently, several new languages have been defined and studied in the context of knowledge compilation.

Definition 23 (Berre et al. [2018])**.** *A* pseudo-Boolean constraint *is of the form* $\sum_{i=1}^{n} a_i l_i \ \Delta \ k$ *where* $n \in \mathbb{N}$, $a_i \in \{\mathbb{Z}\}$, l_i *is a literal,* $\Delta \in \{<, \leq, =, \geq, >\}$ *and* $k \in \mathbb{Z}$. *A pseudo-Boolean constraint is* normalized *if* $\forall i \in \{1, \ldots, n\}$*:* $a_i \in \mathbb{N}$, $\Delta = \geq$, $k \in \mathbb{N}$ *and each variable appears in the constraint exactly once. A normalized pseudo-Boolean constraint is a* cardinality constraint *if* $\forall i \in \{1, \ldots, n\}$*:* $a_i = 1$. *We denote the language of conjunctions of pseudo-Boolean constraints by* PBC *and the language of conjunctions of cardinality constraints by* CARD.

Definition 24 (Čepek and Chromý [2020])**.** *Let* $<$ *define a total order on the set of all propositional variables. Let* X *be a subset of propositional variables of size* n*, and let* f *be a Boolean function on variables from* X*. Consider a vector* $x \in \{0,1\}^n$ *where the bits of* x *correspond to the variables of* X *in the prescribed order* $<$*. Each such vector* x *can be in a natural way identified with a binary*

number from $[0, 2^n - 1]$*, so for every* $x > 0$ *the vector* $x - 1$ *is well defined. We call* $x \in \{0,1\}^n$ *a* switch *of* f *with respect to order* $<$*, if* $f(x-1) \neq f(x)$*. The list of all switches of* f *with respect to* $<$ *is called the* switch-list *of* f *with respect to* $<$*. The switch-list of* f *with respect to* $<$ *together with the function value* $f(0)$ *is called the* switch-list representation (SLR) *of* f *with respect to* $<$*. The set of switch-list representations with respect to* $<$ *(of all functions) forms the propositional language* $\mathrm{SL}_<$*. Finally, the language* SL *is the union of* $\mathrm{SL}_<$ *languages over all total orders on the set of propositional variables.*

3.2 Succinctness

We now discuss the way we may compare different representations of Boolean functions with respect to their size. This is the notion of succinctness.

Definition 25. *Let* L_1 *and* L_2 *be two representations of Boolean functions* L_1 *is* at least as succinct as L_2*, denoted* $L_1 \leq L_2$*, if and only if there exists a polynomial* p *such that for every sentence* $\alpha \in L_2$*, there exists an equivalent sentence* $\beta \in L_1$ *where* $|\beta| \leq p(|\alpha|)$*.*

As usual, we denote by $<$ the strict relation. That is, L_1 is strictly more succinct than L_2, denoted by $L_1 < L_2$, if $L_1 \leq L_2$ but $L_2 \not\leq L_1$. In Figure 3.3 we show a diagram representing the known succinctness results.

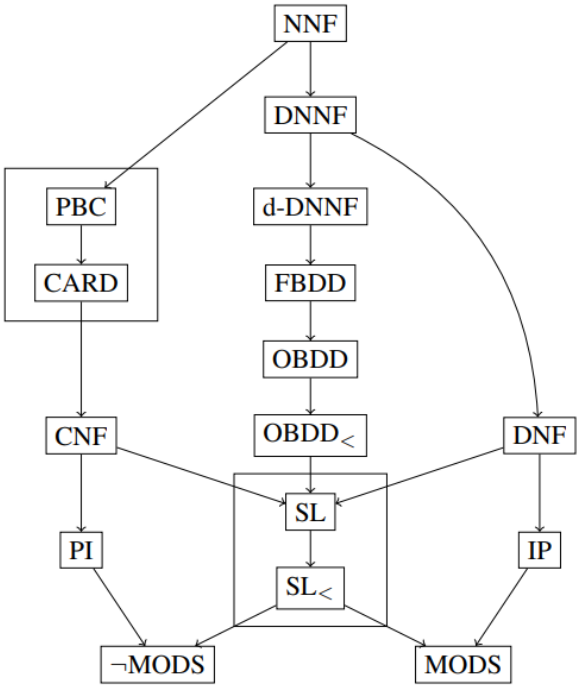

Figure 3.3: Diagram representing known succinctness results. The languages included are from Darwiche and Marquis [2002], Berre et al. [2018] and Čepek and Chromý [2020]. An edge $L_1 \to L_2$ indicates that L_1 is strictly more succinct than L_2.

.

We present here the standard technique of showing succinctness results. In order to show that $L_1 \leq L_2$, it is necessary to find a construction transforming a sentence in language L_2 to one in L_1 polynomial in its size. To show that the relation is strict, it is necessary to show that there exists a family of functions with an increasing number of variables which has small representations in L_1, but exponentially large ones in L_2.

For example, consider the languages CNF and PI. Since PI is a subset of CNF, CNF$\leq$PI holds trivially. However, since there are functions with exponentially many prime implicates but a small CNF, the relation is strict.

3.3 Queries

We now move on to the measure of tractability of a language. The first way to do this is by seeing how hard it may be to answer certain queries.

Definition 26. *We say that knowledge representation language L supports query $\boldsymbol{Q}$ if there is a polynomial time algorithm that provides the answer to the query, given a sentence in L which represents a function f and possibly some additional input. The query $\boldsymbol{Q}$ may be one of the following:*

- ***CO**, checking whether there is a model of f;*
- ***VA**, checking whether f is a tautology;*
- ***CE**, checking whether f implies a given clause C;*
- ***IM**, checking whether f is implied by a given term T;*
- ***EQ**, checking whether two sentences Σ and Φ from L represent the same function f;*
- ***SE**, checking whether a given sentence Σ implies another given sentence Φ;*
- ***CT**, finding the number of models of f;*
- ***ME**, enumerating all models of f.*

In Darwiche and Marquis [2002] (page 2), a representation language is called a *target compilation language* if it supports **CE**. That is, languages supporting this query qualify as possibly useful, as it may be easy to deduce some information about a function in polytime.

3.4 Transformations

The next measure of tractability is one which allows us to see how efficiently we can transform a sentence in a language.

Definition 27. *We say that knowledge representation language L supports transformation $\boldsymbol{T}$ if there is a polynomial time algorithm that, given a sentence in L which represents a function f and possibly some additional input, outputs the representation in the same language of a new function g obtained via the transformation. The transformation $\boldsymbol{T}$ may be one of the following:*

	CO	VA	CE	IM	EQ	SE	CT	ME
NNF	○	○	○	○	○	○	○	○
DNNF	✓	○	✓	○	○	○	○	✓
d-DNNF	○	○	○	○	○	○	○	○
CARD	○	✓	○	✓	○	○	○	○
PBC	○	✓	○	✓	○	○	○	○
BDD	○	○	○	○	○	○	○	○
FBDD	✓	✓	✓	?	○	○	✓	✓
OBDD	✓	✓	✓	✓	✓	○	✓	✓
$\text{OBDD}_{<}$	✓	✓	✓	✓	✓	✓	✓	✓
CNF	○	✓	○	✓	○	○	○	○
DNF	✓	○	✓	○	○	○	○	✓
IP	✓	✓	✓	✓	✓	✓	○	✓
SL	✓	✓	✓	✓	✓	✓	✓	✓
$\text{SL}_{<}$	✓	✓	✓	✓	✓	✓	✓	✓
MODS	✓	✓	✓	✓	✓	✓	✓	✓

Figure 3.4: Table of query results, as per Čepek [2022]. Here ✓ denotes that a transformation can be done in polynomial time, ○ denotes that it cannot be done in polynomial time unless $P = NP$ and ? denotes unknown results.

- ***CD***, *conditioning f on given term T; that is, finding the function obtained by satisfying all literals in the term T, denoted $f \mid T$;*
- ***FO***, *forgetting a subset of variables X from f; that is, finding the function $\exists X.f$;*
- ***SFO***, *forgetting a single variable x; that is, finding the function $\exists x.f$;*
- $\wedge\boldsymbol{C}$, *conjunction of a finite set of functions;*
- $\wedge\boldsymbol{BC}$, *conjunction of two functions;*
- $\vee\boldsymbol{C}$, *disjunction of a finite set of functions;*
- $\vee\boldsymbol{BC}$, *disjunction of two functions;*
- $\neg\boldsymbol{C}$, *negation of a function.*

	CD	FO	SFO	∧C	∨C	¬C
NNF	✓	○	✓	✓	✓	✓
DNNF	✓	✓	✓	○	✓	○
d-DNNF	✓	○	✓	✓	✓	✓
CARD	✓	○	?	✓	•	•
PBC	✓	•	•	✓	•	•
BDD	✓	○	✓	✓	✓	✓
FBDD	✓	•	○	•	•	✓
OBDD	✓	•	✓	•	•	✓
$\text{OBDD}_<$	✓	•	✓	•	•	✓
CNF	✓	○	✓	✓	•	•
DNF	✓	✓	✓	•	✓	•
IP	✓	•	•	•	•	•
SL	✓	✓	✓	•	•	✓
$\text{SL}_<$	✓	✓	✓	•	•	✓
MODS	✓	✓	✓	•	•	•

Figure 3.5: Table of transformation results, as per Čepek [2022]. Here ✓ denotes that a transformation can be done in polynomial time, • denotes that it cannot be done in polynomial time and ○ denotes that it cannot be done in polynomial time unless $P = NP$.

4. Queries and Transformations of BNN

In this chapter, we discuss results about tractability of **BNN**. In particular, we show hardness results for some transformations and construct polytime algorithms for one transformation and multiple queries. We manage to qualify BNN as a target compilation language.

4.1 Transformations

We first consider what happens when transforming a Boolean function represented by a **BNN**. First, we make an easy observation about negation, which follows directly from the definition of negation. Afterwards, we discuss hardness of conditioning and forgetting.

Proposition 3. ***BNN*** *supports* $\neg$***C***.

Proof. We need to show that we can compute the negation of a Boolean function in polynomial time. Consider an algorithm that given (P, N), outputs (N, P). Since each vector in the hypercube is classified by (P, N) by definition of BNN, the output gives exactly the opposite classification. Then this is a well-defined BNN. □

Theorem 4. ***BNN*** *does not support* ***CD***.

Proof. It suffices to find a family of BNN formulas (P_n, N_n) and consistent terms T_n, such that a representation of the function $(P_n, N_n) \mid T_n$ cannot be computed in polynomial time.

Let (P_n, N_n) be the smallest BNN representing the Boolean majority function on $n = 4k$ variables, for some $k \in \mathbb{N}$. That is, (P_n, N_n) represent the function TH^{2k}_{4k}. We know that $|(P_n, N_n)| \leq \frac{n}{2} + 2 = 2(k+1)$ by Theorem 2. Let $x_1, \ldots, x_n$ denote the variables of the threshold function. Notice that by conditioning on some term $T = x_i$ we obtain a threshold function which has one variable fewer and threshold smaller by one.

Let $T_k = x_1 \wedge x_2 \wedge \cdots \wedge x_k$. Then $(P_n, N_n) \mid T_k = TH^k_{3k}$. By Theorem 2, we know that in order to represent such a function, we need a BNN of size $2^{\Omega(3k)} = 2^{\Omega(n)}$, and thus cannot produce it in polynomial time. □

Lemma 5. *Let* $m, n \in \mathbb{N}$, $m \leq n$ *and let* $f = TH^m_n$ *be a threshold function such that* $f(x_1, x_2, \ldots, x_n) = 1$ *if and only if* $x_1 + x_2 + \cdots + x_n \geq m$. *Fix some* $i \in \{1, 2, \ldots, n\}$. *Then*

$$f|x_i \equiv \exists x_i . f(x_1, x_2, \ldots, x_n)\,.$$

Proof. From the proof of Theorem 4, we know that $f|x_i \equiv TH^{m-1}_{n-1}$. Thus it suffices to show that also $\exists x_i . f(x_1, x_2, \ldots, x_n) \equiv TH^{m-1}_{n-1}$. We know that by definition:

$$\exists x_i . f(x_1, x_2, \ldots, x_n) \equiv$$
$$f(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \vee f(x_1, x_2, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \equiv$$
$$TH^m_{n-1} \vee TH^{m-1}_{n-1}$$

Notice now that the positive vectors of TH^m_{n-1} form a subset of the positive vectors of TH^{m-1}_{n-1}. We may then omit TH^m_{n-1} and hence $\exists x_i.f(x_1, x_2, \ldots, x_n) \equiv TH^{m-1}_{n-1}$, as desired. □

Theorem 6. ***BNN*** *does not support* ***FO****.*

Proof. We combine Theorem 4 and Lemma 5. Then it is easy to represent $TH^{n/2}_n$, but hard to transform it by forgetting the first $n/4$ variables. □

We have shown that BNN does not support two of the standard transformations from knowledge compilation. As can be observed in Figure 3.5, all representations that have been considered and studied before supported at least CD. This leads us to conclude that BNN would not be a suitable representation to use when transforming Boolean functions.

While we have not proven that other non-trivial transformations are hard, we conjecture that disjunction and conjunction are hard, even in the bounded case. Thus far it is not clear how one would be able to find the $\vee$ or $\wedge$ of two BNNs.

We have seen that FO is hard when it comes to BNNs. We conjecture that its restricted variant, SFO, is hard as well. That is, we conjecture that there is a family of functions where conditioning on a single variable yields a function requiring a BNN of exponential size.

Finally, as we have observed that for threshold functions forgetting and conditioning behave in the same way, we note that a transformation analogous to singleton forgetting can be considered. That is, we conjecture that even singleton conditioning (SCD) is hard for BNN. Note that SCD is not considered in Darwiche and Marquis [2002] because all standard representation languages support CD and hence support SCD as well.

4.2 Queries

We begin with a couple of simple observations regarding query answering. Just from the definition of **BNN**, we derive two simple algorithms, for consistency and validity checking.

Proposition 7. ***BNN*** *supports* ***CO****.*

Proof. Notice that for a Boolean function represented by a **BNN** to evaluate to 1 for some input, it is necessary and sufficient that there exists at least one positive prototype. Then there is an algorithm which checks consistency in constant time, by simply checking whether the set of positive prototypes in non-empty. □

Proposition 8. ***BNN*** *supports* ***VA****.*

Proof. Similarly as above, we observe that for a Boolean function represented by a **BNN** to ever evaluate to 0, it is necessary and sufficient for there to be any negative prototype. Thus an algorithm outputting 1 if and only if there are no negative prototypes checks validity in constant time. □

We now show that BNN qualifies as a target compilation language, as defined by Darwiche and Marquis [2002]. In particular, this means that BNN supports clausal entailment queries.

Theorem 9. ***BNN*** *supports* ***IM****.*

Proof. Let f be a Boolean function and let (P, N) be a BNN representing it. Let $T = l_1 \wedge \cdots \wedge l_k$ be a consistent term. We wish to find a polytime algorithm determining whether $T \implies f$. Without loss of generality, we may assume that $\forall 1 \leq i \leq k : l_i \in \{x_i, \neg x_i\}$, since otherwise we may relabel the variables. Let $y \in \mathcal{B}^k$ be such that $y_i = 1$ if $l_i = x_i$ and $y_i = 0$ if $l_i = \neg x_i$. Let H denote the subcube of $\mathcal{B}^n$ determined by the term T. That is, $H := \{x \in \mathcal{B}^n \mid \forall 1 \leq i \leq k : x_i = y_i\}$. Then $T \implies f$ if and only if there is no negative vector inside H.
If there is a negative prototype inside H, we are done as T cannot be an implicate of f. Suppose then that there are no negative prototypes inside H and let N' be the set of projections of all negative prototypes into H. In particular, $N' := \{\text{proj}_T(q) \mid q \in N\}$, where $\text{proj}_T(q)_i = y_i$ for $1 \leq i \leq k$ and $\text{proj}_T(q)_i = q_i$ otherwise.
Suppose that for every $q' = \text{proj}_T(q) \in N'$ there exists a positive prototype $p \in P$ which is closer to it than q. That is, suppose that for every projection q', $f(q') = 1$. We now claim that in this case, there are no negative vectors inside H.
Let us assume for contradiction that there exists such a vector x. Let $x \in H$ be such that $f(x) = 0$ and let q be the closest prototype to x. Then

$$d_H(q, x) = d_H(q, q') + d_H(q', x) > d_H(p, q') + d_H(q', x) \geq d_H(p, x),$$

where p is the positive prototype closest to q' and the equality holds because in $d_H(q, q')$ only distance within the coordinates fixed by H is measured and in $d_H(q', x)$ only distance within H is measured. The first inequality follows from the assumption and the second from the triangle inequality for Hamming distance.
Therefore in order to check whether there is a negative vector inside H, it suffices to look for a projection of a negative prototype which is negative as well.
This idea gives us Algorithm 1. The algorithm runs in time $O(n|P||N|)$, as the main part consists of the two nested *for* loops. In the worst case, the algorithms considers all projections of negative prototypes, and calculates their distances to each of the positive prototypes. □

Corollary 10. ***BNN*** *supports* ***CE****.*

Proof. Let f be a Boolean function and let (P, N) represent it. Let $C = l_1 \vee \cdots \vee l_k$ be a consistent clause. We wish to find a polytime algorithm determining whether $f \implies C$. We know that $(f \implies C) \equiv (\neg C \implies \neg f)$. By DeMorgan laws, the negation of a clause is a term. So let $T = \neg C$. Now, we may answer the question by calling Algorithm 1 for inputs $\neg f = (N, P)$ and T, and get the correct answer. Since by Proposition 3 and Theorem 9 negation and implicant check can be done in polynomial time, the claim follows. □

We have now seen that BNN supports half of the standard knowledge compilation queries. In this case, it is interesting that there exist polynomial time algorithms for IM and CE. In previous results, the fact that a language supports IM usually follows from the fact that it supports CD and VA. We have shown that this is not the case for BNN, and we make note of this interesting behaviour of the BNN language.

Algorithm 1 Checking whether a consistent term implies a **BNN**.

Input: (P, N) representing $f : \mathcal{B}^n \to \mathcal{B}$ and a consistent term $T = l_1 \wedge \cdots \wedge l_k$
Output: $\mathbf{IM}(f, T)$

```
N' ← {proj_T(q) | q ∈ N}
if N ∩ N' ≠ ∅ then
    return 0
end if
for q' ∈ N' do
    d ← +∞
    for p ∈ P do
        d ← min{d, d_H(p, q')}
        if d_H(q, q') < d then
            return 0
        end if
    end for
end for
return 1
```

The questions about whether BNN supports any of the other queries remain open. We conjecture that BNN supports ME, while the remaining queries are hard, i.e. that there are no polynomial time algorithms for EQ, SE and CT.

5. Succinctness of BNN

In this chapter, we present results about succinctness of **BNN**. In particular, we prove the relations from the diagram in Picture 5.1.

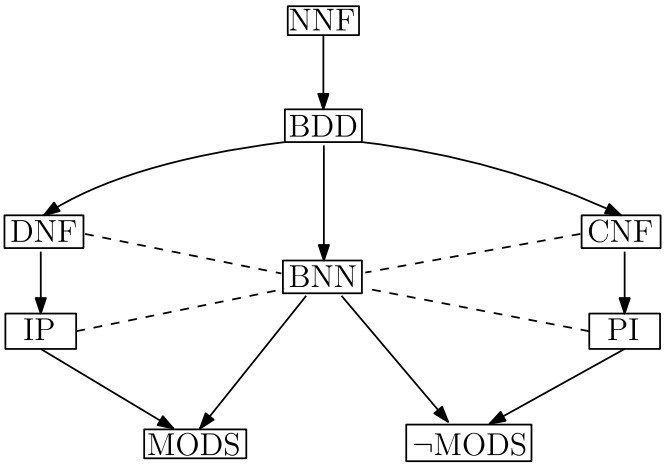

Figure 5.1: Diagram representing BNN succinctness results. A solid directed edge $L_1 \to L_2$ indicates that L_1 is strictly more succinct than L_2. A dashed edge represents incomparability results.

As described in Chapter 3, BDD is a subset of d-NNF. From the diagram on Figure 3.3, we know that NNF<d-NNF and thus the top arrow in Figure 5.1 is correct. We therefore begin by showing that BNN is strictly less succinct than BDD.

Theorem 11. ***BDD*** $<$ ***BNN***.

Proof. It suffices to show that there exists a polynomial p such that for every sentence $\alpha \in \mathbf{BNN}$, there exists an equivalent sentence $\beta \in \mathbf{BDD}$ where $|\beta| \leq p(|\alpha|)$. For any $\alpha = (P, N)$, we will construct such a a binary decision diagram. Let us assume that $P \cup N = \{p^1, \ldots, p^k\}$ and let $x \in \mathcal{B}^n$. We build the BDD in two steps:

1. We construct a gadget which for two fixed prototypes p^i and p^j checks which one is closer to x.

2. We put a number of the gadgets together so that the prototype closest to x is found and its value outputted.

We begin by building a BDD gadget. For any $p^i, p^j \in P \cup N$, we construct a diagram $G_{i,j}$, as seen in Figure 5.2 for $n = 3$. We first compare each coordinate of x with the corresponding coordinate of p^i. Thus the nodes on level $n + 1$ of the gadget reflect the value $d_H(x, p^i)$. Consider for an example the gadget in Figure 5.2 with $x = (1, 0, 1)$ and $p^i = (0, 0, 1)$. In the first three coordinated comparisons, we take the edges labelled N, Y and Y. As only one coordinate differs, on level 4 we know that the value of $d_H(x, p^i)$ is 1.

In the next n levels, we compare each coordinate of x with the corresponding coordinate of p^j. For the example above, consider $p^j = (0, 0, 0)$. Then the next

edges visited will have labels N, Y and N and thus $d_H(x, p^j) = 2$. The gadget will thus determine which of the two prototypes is closer, and send us either to a new gadget $G_{i,k}$ or $G_{j,k}$ for some k. For simplicity, this is depicted as either i or j in Figure 5.2. We define the gadgets to break ties in favor of i. This choice is arbitrary, as we will use the gadgets to find one of the nearest prototypes, and those must all necessary belong to the same class, by the definition of BNN.

We note here that while the gadgets are defined to have nodes which check for equality, those can simply be converted into decision nodes. This is due to the fact that the decisions are predefined by the prototypes, as shown in Figure 5.3.

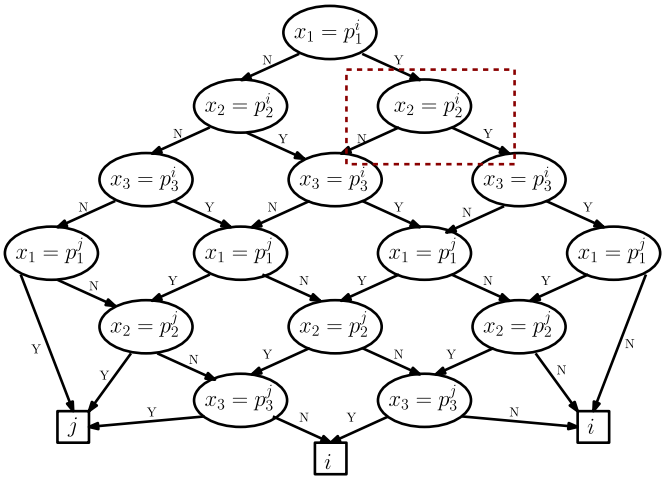

Figure 5.2: Gadget $G_{i,j}$ for $n = 3$. Each equality node can be replaced by a decision node, as shown in Figure 5.3.

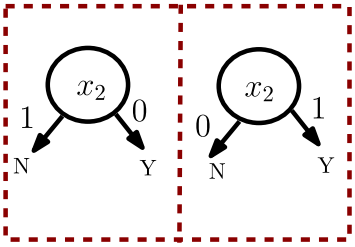

Figure 5.3: A decision node that corresponds to the equality node if $p_2^i = 0$ (left) and if $p_2^i = 1$ (right).

It now remains to build the final BDD using the gadgets. We can sequentially test pairs of prototypes until the closest one is found. We do so by making a tree of gadgets. It contains $k - 1$ levels. At each level i, we compare p^{i+1} with every p^j such that $j < i$, and add a directed edge pointing to the appropriate gadget on the next level. After level $k-1$, a closest prototype has been determined, and a directed edge pointing to 1 (resp. 0) is added if the closest protype is positive (resp. negative). The final BDD for a function with k prototypes is shown in Figure 5.4.

We now show that we have constructed a BDD of polynomial size with respect to the size of α. For vectors of length n, each gadget consists of $(n+1)^2-1 = O(n^2)$ decision nodes. As there are k prototypes, we need exactly $\sum_{i=1}^{k-1} i = \frac{(k-1)k}{2} =$

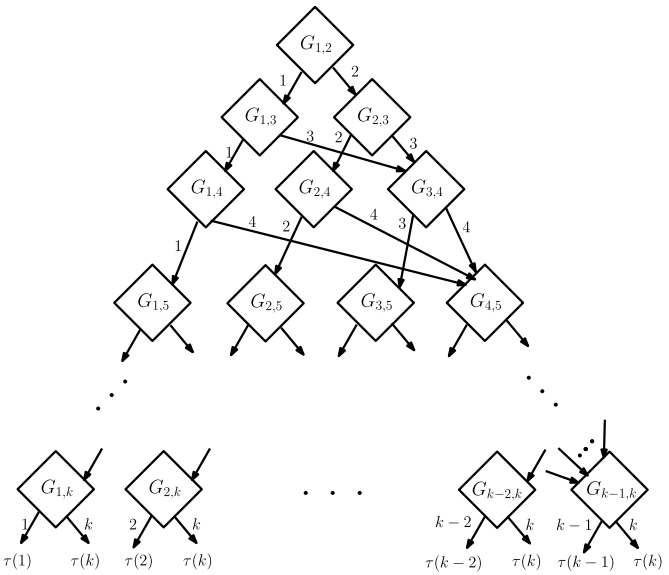

Figure 5.4: BDD of a function with k prototypes, where $\tau(i) = 1$ if and only if the i^{th} prototype is positive.

$O(k^2)$ gadgets. Hence the size of the constructed BDD is $O(n^2k^2) = O(|\alpha|^2)$, as desired.

To show that the inequality is strict, it suffices to consider the parity function. From Chapter 2, we know that a BNN of this function requires 2^n prototypes. However, a BDD of the parity function on n variables can be of size $2n + 1$, as shown in Figure 5.5 for $n = 4$. □

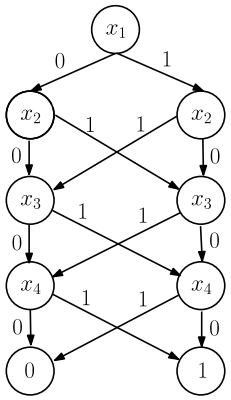

Figure 5.5: A binary decision diagram for the parity function on 4 variables.

In order to shown the rest of the results, we will need the following lemma.

Lemma 12 (DiCicco et al. [2024])**.** *Let f be a Boolean function on n variables. If the subgraph of $\mathcal{B}^n$ induced by $f^{-1}(1)$ has m connected components then any BNN representation of f has at least m prototypes.*

Proof. Let C be a connected component of the subgraph of $\mathcal{B}^n$ induced by $f^{-1}(1)$. Let $\delta(C) = \{x \in \mathcal{B}^n \mid d_H(x, C) = 1\}$. Then $\delta(C) \subseteq f^{-1}(0)$.

Let $P \cup N$ be a BNN representing f and let $p \in P$ be the nearest positive prototype to some $x \in C$. Suppose for contradiction that $p \notin C$. Then there must be $y \in \delta(C)$ such that y is on a shortest path from x to p and thus $d_H(x,p) = d_H(x,y) + d_H(y,p)$. Since $f(y) = 0$, there must be a negative prototype $q \in N$ such that $d_H(y,q) < d_H(y,p)$. But then

$$d_H(x,q) \leq d_H(x,y) + d_H(y,q) < d_H(x,y) + d_H(y,p) = d_H(x,p),$$

where the first inequality follows from the triangle inequality for Hamming distance. But then q is closer to x than p, contradicting the minimality of p. Thus, every connected component C must contain a positive prototype. □

Corollary 13. *Let f be a Boolean function and let x be positive vector such that x is isolated in the subgraph induced by $f^{-1}(1)$. Then x must be a positive prototype in any BNN representation of f.*

Corollary 14. *Let f be a Boolean function on n variables. If the subgraph of $\mathcal{B}^n$ induced by $f^{-1}(0)$ has m connected components then any BNN representation of f has at least m prototypes. In particular, if x is a negative vector such that x is isolated in the subgraph induced by $f^{-1}(0)$, then x must be a negative prototype in any BNN representation of f.*

Proposition 15. ***BNN*** $<$ ***MODS***.

Proof. We first show that **BNN** $\leq$ **MODS**. Suppose that a Boolean function f has m models and let M be the set of all models and let $\delta(M)$ be as in the proof of Lemma 12. Let

$$(P,N) = (M, \delta(M)).$$

We claim that this (P,N) is a BNN representation of f.

Suppose that x is such that $f(x) = 1$. Then $x \in M = P$. Otherwise, we have $f(x) = 0$. If $x \in N$, we are done. If $x \notin N$, then $d_H(x,M) \geq 2$. But then any path from x to a model must pass through a point in $\delta(M)$ and thus through a negative prototype.

In the worst case, all neighbors of all models need to be picked as negative prototypes. Thus we can construct a BNN with at most $m + nm$ prototypes, and we have shown **BNN** $\leq$ **MODS**.

It remains to show that **MODS** $\not\leq$ **BNN**. Let f be the function on n-dimensional vectors which is constantly 1. Then we may represent f by a single positive prototype and no negative ones. On the other hand, there are 2^n models of f, and MODS cannot be more succinct than BNN. □

Corollary 16. ***BNN*** $<$ $\neg$***MODS***.

Proof. The corollary follows by an argument analogous to the one for MODS. □

We now show that BNN is incomparable to CNFs and DNFs.

Lemma 17. ***BNN*** $\not\leq$ ***CNF***.

Proof. It suffices to show that there is a Boolean function with CNF representation of size polynomial in the number of variables, which cannot be represented by a BNN of polynomial size. We construct a family of such functions as in Di-Cicco et al. [2024].
Let $n = 4k$ for some positive integer k. Let $S_i := \{x_i, x_{i+1}, x_{i+2}, x_{i+3}\}$ for $1 \leq i \leq 4(k-1)+1$. We define the CNFs

$$\begin{aligned}\phi_i :=&(x_i \vee x_{i+1} \vee x_{i+2} \vee x_{i+3}) \wedge (x_i \vee x_{i+1} \vee x_{i+2} \vee \neg x_{i+3}) \wedge \\ &(x_i \vee x_{i+1} \vee \neg x_{i+2} \vee x_{i+3}) \wedge (x_i \vee \neg x_{i+1} \vee x_{i+2} \vee x_{i+3}) \wedge \\ &(\neg x_i \vee x_{i+1} \vee x_{i+2} \vee x_{i+3}) \wedge (\neg x_i \vee \neg x_{i+1} \vee \neg x_{i+2} \vee \neg x_{i+3}) \wedge \\ &(\neg x_i \vee \neg x_{i+1} \vee \neg x_{i+2} \vee x_{i+3}) \wedge (\neg x_i \vee \neg x_{i+1} \vee x_{i+2} \vee \neg x_{i+3}) \wedge \\ &(\neg x_i \vee x_{i+1} \vee \neg x_{i+2} \vee \neg x_{i+3}) \wedge (x_i \vee \neg x_{i+1} \vee \neg x_{i+2} \vee \neg x_{i+3}).\end{aligned}$$

That is, $\phi_i = 1$ if and only if exactly two of the variables in S_i are 1. Let

$$\phi := \bigwedge_{j=1}^{k} \phi_{4(j-1)+1}.$$

Let f be the function that ϕ represents. Then ϕ is a CNF representation of f of size $40k = 10n$.
We now claim that a BNN representation of f cannot be of polynomial size. Let $x \neq y \in \mathcal{B}^n$. Suppose that $f(x) = 1$ and $f(y) = 1$. Since $x \neq y$, they must differ in at least one of the k disjoint sets of coordinates S_i, and therefore in at least 2 coordinates. Hence $d_H(x, y) \geq 2$, and each positive vector is isolated in the subgraph of $\mathcal{B}^n$ induced by $f^{-1}(1)$.

It now remains to count the positive vectors. Consider first some ϕ_i. There are exactly $\binom{4}{2}$ ways to pick the two positive variables. As the variables of each ϕ_i form disjoint sets, there are $\binom{4}{2}^k = 6^k$ positive vectors. Hence by Corollary 13, at least 6^k prototypes are needed. □

Lemma 18. ***CNF*** $\not\leq$ ***BNN***.

Proof. Consider the majority function on n variables. Any CNF of this function must be of exponential size [Darwiche and Marquis [2002]]. However, as shown in Chapter 2, this function has a BNN representation of size $\leq \frac{n}{2} + 2$, and so the claim follows. □

Theorem 19. ***BNN*** *is incomparable with* ***CNF***.

Proof. We combine Lemma 17 and Lemma 18 and obtain the result. □

Corollary 20. ***BNN*** *is incomparable with* ***DNF***.

Proof. To show that **DNF** $\not\leq$ **BNN** it again suffices to consider the majority function.
To show that **BNN** $\not\leq$ **DNF** consider the function $\neg\phi$ where ϕ is as defined in the proof of Lemma 17. After an application of DeMorgan laws, we obtain a DNF formula of polynomial size representing the function $\neg f$. However, we may repeat the argument from Lemma 17 for the negative vectors of $\neg f$ and conclude that an exponential number of negative prototypes is required. □

Corollary 21. ***BNN*** *is incomparable with both* ***IP*** *and* ***PI****.*

Proof. Consider the function ϕ from the proof of Lemma 17. Let us first identify the prime implicates of each ϕ_i. In order to see that $\phi_i = 0$ it suffices to check whether any three variables have either all been set to 1 or to 0. Thus the canonical CNF of ϕ_i is

$$
\begin{aligned}
&(x_i \vee x_{i+1} \vee x_{i+2}) \wedge (x_i \vee x_{i+1} \vee x_{i+3}) \wedge \\
&(x_i \vee x_{i+2} \vee x_{i+3}) \wedge (x_{i+1} \vee x_{i+2} \vee x_{i+3}) \wedge \\
&(\neg x_i \vee \neg x_{i+1} \vee \neg x_{i+2}) \wedge (\neg x_i \vee \neg x_{i+1} \vee \neg x_{i+3}) \wedge \\
&(\neg x_i \vee \neg x_{i+2} \vee \neg x_{i+3}) \wedge (\neg x_{i+1} \vee \neg x_{i+2} \vee \neg x_{i+3}).
\end{aligned}
$$

Furthermore, as the subformulas ϕ_i are defined on disjoint sets of variables, the set of all their prime implicates is exactly the set of the prime implicates of f. Thus there are exactly $8k$ prime implicates of f, and so BNN$\not\leq$PI follows.

The relation PI$\not\leq$BNN follows from the fact that CNF$\not\leq$BNN as the PI language is a subset of the CNF language.

By symmetric arguments, the same can be shown for the language IP. □

We now briefly discuss OBDD. In particular, we show one side of its incomparability to BNN.

Proposition 22. ***BNN*** $\not\leq$ ***OBDD****. That is, there is a Boolean function with OBDD representation of polynomial size, which cannot be represented by a BNN of polynomial size.*

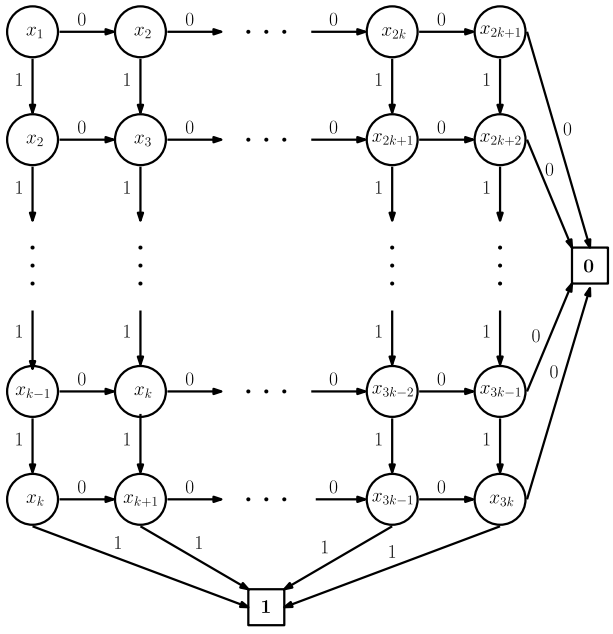

Figure 5.6: The construction of OBDD for $TH_n^{\lfloor n/3 \rfloor}$

Proof. Recall that the function $TH_n^{\lfloor n/3 \rfloor}$ can only be represented by a BNN of exponential size. We claim that this function has a polysize OBDD representation. For simplicity we consider only n such that $n = 3k$. We construct a diagram representing this function as follows:

We start by making an undirected $k \times (2k+1)$ grid. We orient all horizontal edges to the right, and label them by 0. The vertical edges should be oriented downwards, and labeled by 1.

Now we add labels to the nodes. For each row of the grid $i \in \{1, 2, \ldots, k\}$, we label its nodes as $x_i, x_{i+1}, \ldots, x_{i+2k}$.

Finally, we add two nodes, **1** and **0**. It remains to add edges labeled 1 connecting each vertical path with **1**, and edges labeled 0 connecting each horizontal path to **0**. The constructed OBDD is shown in Figure 5.6.

We now claim that the construction is indeed an OBDD representation of $TH_n^{n/3}$. Consider first all paths which lead to **1**. Each of them consists of exactly $n/3$ edges labeled 1, and thus at least $n/3$ variables have value 1. Similarly, all paths to **0** have exactly $2k+1$ edges labelled 0, and thus at most $n/3-1$ variables can be set to 1 in that case. □

We note here that it can in fact be shown that any symmetric function can be represented by a OBDD of polynomial size, as shown by Wegener [2000]. Thus the proof of Proposition 22 is just a special case of a more general result but we present it here as it is much simpler and suffices for the proof of BNN$\not\leq$OBDD.

The other direction of the above comparison remains an open question. We conjecture that also OBDD$\not\leq$BNN. It also remains to answer similar questions about the newer knowledge representation languages.

6. Conclusion

We have examined the Boolean nearest neighbor representation of Boolean functions. After reviewing the already known results about the complexity of this representation, as introduced by Hajnal et al. [2022], as well as the setup of the knowledge compilation map as per Darwiche and Marquis [2002], we were able to compare the BNN representation to standard knowledge representation languages.

We have shown that it is easy to negate a BNN, but that some other transformations are hard. In particular, unlike all standard knowledge representation languages, BNN does not support polynomial-time conditioning. As a consequence of this, we showed that BNN does not support polynomial-time forgetting as well.

The results are different for the standard queries. It is easy to check whether a BNN is consistent or valid. We have presented an algorithm which performs implicant and clausal entailment checks in polynomial time. We conjecture that more queries can be answered in polynomial time.

With respect to its support of different queries and transformations, BNN appears to have some unique properties. In particular, the hardness of conditioning makes it stand up among other languages. However, it is possible to check clausal entailment in polynomial time, which makes it a target compilation language. For other target languages, this follows from the fact that they support polynomial-time conditioning and consistency check, which is not the case for BNN.

In terms of succinctness, BNN lies between BDD and MODS. It is incomparable with CNF, DNF, PI and IP. We conjecture that it is also incomparable with OBDD.

Recent research on nearest neighbor representations of Boolean functions in DiCicco et al. [2024] and Kilic et al. [2023] asks and answers many questions about the representation, but not in the context of knowledge compilation. We have answered several questions that appear in this context.

Bibliography

Franz Aurenhammer, Gerd Stöckl, and Emo Welzl. The post office problem for fuzzy point sets. In *Workshop on Computational Geometry*, volume 553 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 1991.

Daniel Le Berre, Pierre Marquis, Stefan Mengel, and Romain Wallon. Pseudo-boolean constraints from a knowledge representation perspective. In *IJCAI*, pages 1891–1897. ijcai.org, 2018.

Ondřej Čepek. Switch lists in the landscape of knowledge representation languages. In *FLAIRS*, 2022.

Ondřej Čepek and Milos Chromý. Properties of switch-list representations of boolean functions. *J. Artif. Intell. Res.*, 69:501–529, 2020.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002. doi: 10.1613/jair.989. URL `https://doi.org/10.1613/jair.989`.

Mason DiCicco, Vladimir Podolskii, and Daniel Reichman. Nearest neighbor complexity and boolean circuits. *Electron. Colloquium Comput. Complex.*, pages TR24–025, 2024.

Péter Hajnal, Zhihao Liu, and György Turán. Nearest neighbor representations of boolean functions. *Inf. Comput.*, 285(Part B):104879, 2022. doi: 10.1016/j.ic.2022.104879. URL `https://doi.org/10.1016/j.ic.2022.104879`.

Kordag Mehmet Kilic, Jin Sima, and Jehoshua Bruck. On the information capacity of nearest neighbor representations. In *ISIT*, pages 1663–1668. IEEE, 2023.

Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000.

Zhongheng Zhang. Introduction to machine learning: k-nearest neighbors. *Annals of Translational Medicine.*, 4(11):218, 2016.

List of Figures

List of Algorithms