A Generalized Reduction of Ordered Binary Decision Diagram

Joan Thibault

June 12, 2017

Abstract

Reduced Ordered Binary Decision Diagram ((RO)BDD) [3,14] are the state-of-the-art representation for Boolean functions. They are used in various fields such as logic synthesis, artificial intelligence or combinatorics. However, BDDs suffer from two main issues: (1) their representation is memory expensive and (2) their manipulation is memory intensive as it induces many random memory accesses.

Various variations of ROBDD exist such as Zero-suppressed Decision Diagram (ZDD) [13], Multi-valued Decision Diagram (MDD) [10,11] and variations of the reduction rules such as "output negation" [2], "input negation" [12], "shifting variables" [12], "dual edges" [9] or "copy node" [8].

In this report, we introduce a new generalization of the standard reduction rules that we call the "extraction of useless variables" or "extract U" for short. Basically, it detects useless variables (i.e. variables which have no influence on the result of a given function) and extracts them from the local variable order. This generalization allows to add and remove useless variables in linear time (in the number of variables), reduces the number of nodes and tends to reduce the overall memory cost. However, several drawbacks arise: no in-place sifting (permutation of adjacent variables), bigger nodes of variable size and it slightly complexifies manipulations.

We implemented both the "extract U" and the "output negation" variants in an OCaml program and tested it against several benchmarks [5,6,15]. We observe an average of 25% less nodes and 32% less memory when representing circuits, and 3% less nodes and memory when representing solutions from generated CNF formulas.

Contents

Introduction

Nowadays, most critical systems rely on digital circuits: in transportation (e.g. cars, train, plains) , communication (e.g. satellites), computation (e.g. data centers, super-computers), exploration (e.g. space rocket, rovers). One way to minimize risks in digital parts of these systems is to provide a formal proof that they respect their specification. On the other hand, we want to minimize costs and energy consumption while maximizing performances of these digital circuits. In order to efficiently optimize digital circuits we rely on complex programs. However, these programs are rarely proven themselves, thus, circuits optimized using them might not be equivalent to the initial design, therefore, might not respect the specification. The obvious solution would be to prove optimizing programs, however two majors issues arise : these programs are complex (thus, proving them would be expensive) and might be proprietary (thus, one cannot check that the proof is correct). A simpler alternative is to design a program which check that two digital circuits are equivalent. With this alternative, the only piece of software which needs to be proven is the "equivalence checker".

In order to prove that two digital circuits are equivalent, there is two main algorithmic solutions: Firstly, the DPLL (Davis–Putnam–Logemann–Loveland) algorithm. This backtracking procedure is usually implemented with various heuristics such as *unit propagation*, early conflict detection or *conflict driven clause learning.* Secondly, the compilation of both circuits into Reduced Ordered Binary Decision Diagrams (ROBDDs). A ROBDD is a canonical structure which represent a function, thus, once compiled, the identity test can be performed in constant time. However, the compilation might take an exponential time in the number of variable. In this report we will focus on ROBDDs.

ROBDDs have various other applications such as: Bounded Model Checking, Planning, Software Verification, Automatic Test Pattern Generation, Combinational Equivalence Checking or Combinatorial Interaction Testing.

However, BDDs are memory expensive as their size tends to grow exponentially with the number of variables. Various variants have been invented in order to capture some semantic properties of the function and reduce the memory consumption. For example, Zero suppressed binary Decision Diagram (ZDD) are better suited for representing sparse functions. In this report we will use the "output negation" variant [2], which extends the reduction rules in order to guarantee canonicity under negation. Thus, in addition to reduce the size of the structure, allows to negate a function in constant time (reducing the set of useful binary operators to XOR and AND). Other extensions of the reduction rules exist such as: "input negation" [12] (each edge can complement the first locally first input), "shifting variables" [12] (each edge stores the number of useless variables before the next significant variables) or "dual edge" [9] (we define the dual of a function f by $\bar{f} = X \longrightarrow \bar{f}(\bar{X})$, therefore the reduction works similarly to the "output inversion").

In addition to use the "output negation" variant, we introduce a new variant which allows to extract useless variables (a.k.a non-support variables). A variable is said useless iff it does not change the results such as x_1 in $f(x_0, x_1, x_2) = x_0 \wedge x_2$ or x_0 in $g(x_0) = x_0 \wedge \neg x_0$. We call this new variant "Useless variable extraction" or "U-extract" for short.

The remainder of this repport will be organised as follows. In Section 1, we formally introduce Boolean functions, useless variable and some notations. In

Section 2, we formally introduce Reducec Ordered Binary Decision Diagrams (ROBDD). In Section 3, we introduce the Generalized Reduction of Ordered Binary Decision Diagrams (GroBdd). In Section 4, we introduce the "U-extract" variant as part of the GroBdd framework, before to expose results against three different benchmqrks [5, 6, 15] using our implementation in OCaml.

1 Notations

Reduced Ordered Binary Decision Diagrams represent Boolean functions. In this section we introduce notations necessary to their manipulation.

We denote the set of Booleans $\mathbb{B} = \{0, 1\}$. The set of Boolean vector of size $n \in \mathbb{N}$ is denoted $\mathbb{B}^n$. The set of Boolean functions of arity (i.e. the number of variables) $n \in \mathbb{N}$ is denoted $\mathbb{F}_n = \mathbb{B}^n \longrightarrow \mathbb{B}$.

We denote conjunction by $\wedge$, disjunction by $\vee$, negation by $\neg$. We denote the Shannon operator by $\longrightarrow_S$, defined by $\forall x, y, z \in \mathbb{B}, x \longrightarrow_S y, z = (\neg x \wedge y) \vee (x \wedge z))$.

Restriction

Let $f \in \mathbb{F}_{n+1}$ be a Boolean function of arity $n+1$, i $(0 \leq i < n+1)$ be an integer and $b \in \mathbb{B}$ be a Boolean. We denote $f[i \leftarrow b]$ the Boolean function of arity n defined by $f[i \leftarrow b](x_1, \dots, x_n) = f(x_0, \dots, x_{i-1}, b, x_i, \dots, x_n)$. $f[i \leftarrow 0]$ (respectively $f[i \leftarrow 1]$) is called the i-th negative (respectively positive) restriction.

For each Boolean function f of arity n+1, we denote $f[i \leftarrow b] = (x_1, ..., x_n) \rightarrow f(x_1, .., x_{i-1}, b, x_i, ..., x_n)$ the function of arity n called the i-th positive restriction of f if $b = 1$, the negative one otherwise.

Let f be a function of arity $n+1$, we denote $f_0 = f[0 \leftarrow 0]$ (respectively $f_1 = f[0 \leftarrow 1]$) the function of arity n.

Construction

Let $f, g \in F_n$ be Boolean functions of arity n and i $(0 \leq i < n+1)$ be an integer. We denote $f \star_i g$ the Boolean function of arity $n+1$ defined by $(f \star_i g)(x_0, \dots, x_{i-1}, y, x_i, \dots, x_n) = y \longrightarrow_S f(x_0, \dots, x_n), g(x_0, \dots, x_n)$.

We denote $\star = \star_0$

N.B.: $(f \star_i g)[i \leftarrow 0] = f$ and $(f \star_i g)[i \leftarrow 1] = g$ $((f \star g)_0 = f$ and $(f \star g)_1 = g)$.

Expansion Theorem

Let f be a function of arity n, then $\forall i, 0 \leq i < n \Rightarrow f = f[i \leftarrow 0] \star_i f[i \leftarrow 1]$ (in particular $f = f_0 \star f_1$)

Restriction Distributivity

- $(\neg f)[i \leftarrow b] = \neg f[i \leftarrow b]$
- $(f \wedge g)[i \leftarrow b] = f[i \leftarrow b] \wedge g[i \leftarrow b]$
- $(f \vee g)[i \leftarrow b] = f[i \leftarrow b] \vee g[i \leftarrow b]$

Useless Variables

Let $f \in \mathbb{F}_n$ be a Boolean function. We define the support set of f by: the set of variable index i such that $f[i \leftarrow 0] \neq f[i \leftarrow 1]$. We define useless variables (or non-support variables) as variables which does not belong the support set. Thus, the i-th variable of f is useless iff $f[i \leftarrow 0] = f[i \leftarrow 1]$.

2 Reduced Ordered Binary Decision Diagram (ROBDD) and Canonicity

Definition of Binary Decision Diagram (BDD)

A Reduced Ordered Binary Decision Diagram is a directed acyclic graph $(V \cup T, \Psi \cup E)$ representing a vector of Boolean functions $F = (f_1, ..., f_k)$ over an infinite set of variables (but with a finite support set). Nodes are partitioned into two sets : the set of internal nodes V and the set of terminal nodes T. Every internal node $v \in V$ has one field var, which represents the index of a variable and two outgoing edges respectively denoted $if0$ and $if1$. When using the "output negation" variant, there is only one terminal called 0, which represents the constant function returning 0. Arcs are partitioned into two sets : the set of root arcs Ψ and the set of internal arcs E. There is exactly k root arcs, a root arc is denoted Ψ_i with $0 \leq i < k$, informally, Ψ_i is the root of the ROBDD representing f_i. Every arc has an inversion field $neg \in \mathbb{B}$ and a destination node denoted $node$.

We denote $\phi(node)$ the semantic of the node $node$ and $\psi(arc)$ the semantic of the arc arc as follow:

- $\forall i, f_i = \psi(\Psi_i)$
- $\forall arc \in \Psi \cup E, \psi(arc) = arc.neg \oplus \phi(arc.node)$
- $\phi(0 \in T) = 0$
- $\forall node \in V, \phi(node) = node.var \longrightarrow_S \psi(node.if0), \psi(node.if1)$

Definition of Reduced Ordered BDD (ROBDD)

A BDD is said `ordered` if (1) $\forall v \in V$, $v.then.node \in V \Rightarrow v.var > v.if1.node.var$ and $v.else.node \in V \Rightarrow v.var > v.if0.node.var$.

A BDD is said `reduced` if (2) $\forall v \in V, v.if0 \neq v.if1$ and (3) every node has an in-degree strictly positive.

Theorem : ROBDD are canonical

Let consider a ROBDD G representing $F = (f_1, ..., f_n)$ over a set of n variables $x_n < x_{n-1} < ... < x_1$. Then, for every nodes $v_1, v_2 \in G$, $\phi(v_1) = \phi(v_2) \Leftrightarrow v_1 = v_2$.

A proof of this theorem is available in the review of Somenzi et al. [14].

2.1 Effective construction

In practice, one does not build the decision tree and then reduces it. Rather, BDDs are created starting from the BDDs for constants and variables and by applying the usual Boolean connectives and are kept reduced at all times. At the same time several functions are represented by one multi-rooted diagram. Indeed, each node of a BDD has a function associated with it. If we have several functions, they will have subfunctions in common (e.g. $f(x_0, x_1, x_2, x_3) = x_1 \longrightarrow_S x_2, x_3$ and $f(x_0, x_1, x_2, x_3) = \neg x_1 \longrightarrow_S x_2, x_3$). As a special case two equivalent functions are represented by the same BDD (not just two identical BDDs). This approach makes equivalence check a constant-time operation. It is usually implemented using a dictionary which stores all BDD nodes that are alive (i.e. created and not deleted). This dictionary is called the *unique table*. Operations that build BDDs start from the bottom (the constant nodes) and proceed up to the function nodes. If an operation needs to add a node $N = (v, f_1, f_0)$ to a BDD (which assumes that it has identifiers of the nodes representing f_1 and f_2), it first checks if N already exists in the *unique table* and if does not associate it to a new identifier and returns the latest. Doing so, the equivalence check is reduced to pointer comparison.

Operator `CONS`

Pseudo-code for dynamic construction of unique nodes.

```
let cons var if0(*else arc*) if1(*then arc*) =
  if if0 = if1
  then if0
  else
  (
    if0' = {neg = 0; node = if0.node}
    if1' = {neg = if0.neg ⊕ if1.neg; node = if1.node}
    mynode = {var; then = if0'; else = if1'}
    myid = if mynode in unique table
      then "mynode's identifier"
      else
      (
        store mynode in uniquetable;
        "mynode's identifier"
      )
    {neg = if0.neg; node = myid}
  )
```

3 Introduction of Generalized Reduction of Ordered Binary Decision Diagram (GroBdd)

3.1 Motivation

On one hand, a ROBDD can be understood as an automaton recognizing a language composed of binary words. On the other hand, a ROBDD can be understood as a logic circuit with limited conciseness. By limited conciseness, we mean that there are functions which have a polynomial representation using an And-Inverter-Graph (AIG, i.e. a logic circuit composed of `AND` and `NOT` gates), but only exponential representation when using ROBDD. For example, it has been proven [3] that the integer multiplication has a quadratic AIG representation but no polynomial ROBDD representation (at least exponential).

The "output negation" variant, by adding expressiveness to edges, improves the conciseness. We want to go further in this approach by allowing more complex transformations on edges while maintaining canonicity. Such transformation can disturb variables' order of evaluation, allowing to represent "simple" decision processes in a more concise way. Furthermore, some transformations (within the range of transformation allowed) could have their complexity drastically reduced. For example, using the "output negation" variant, the complementation's time and space complexity goes from linear in the number of node to constant.

Three set of transformation that would be of great interest are the "useless variable extraction" (or "U extract" for short), the "inputs inverters" and "1-prediction extraction" (or "X extract" for short).

The "U extract" variant allows to introduce useless variables, thus ensures that the function represented by any node, does not have useless variable. Therefore, it sets an upper bound on the number of node of arity n to 2^{2^n} (this upper bound is not reached). Furthermore, it allows to "copy" functions at (almost) no cost as the expression $f(x_1, x_3, x_5, x_7)$ and $f(x_0, x_1, x_3, x_5)$ are represented using the same structure.

The "inputs negation" variant would allow to complement inputs on any edge. The reduction rules would ensure that there is at most $2^{2^n - n}$ nodes of arity n (this upper bound is not reached). However, simply introducing such transformation breaks the canonicity, therefore, we introduce polarity-phase invariant detection: we compute for each encountered function the linear space of input/output negations which do not change the function. This variant is called "input Negation and output Negation Invariant extraction" (or "NNI extract" for short) and is generalization of the work of Burch et al. [4].

We define a 1-prediction as follow: Let $f \in \mathbb{F}_n$ be a Boolean function of arity n, i be an integer ($0 \leq i < n$), x and y be Booleans. We say that the function f admit a 1-prediction (i, x, y) iff $f[i \leftarrow x] = y \in \mathbb{F}_{n-1}$. The "X extract" variant, by allowing to extract 1-predictions, represents a generalization of ZBDD (Zero Suppressed Binary Decision Diagram), thus, allows to efficiently represent sparse functions. This variant is compatible with the "output negation" variant.

Due to the limited length of this report, we will not detail further the "NNI extract" and "X extract" variant.

The Generalized Reduction of Ordered Binary Decision Diagram (GroBdd) is a framework that aims at providing examples of such reduction rules and boundaries on future reduction rules that would fit inside this framework. The remaining of this Section is organized as follows. First, we formally define GroBdds, then we present a set of properties that the transformation must have and some implications.

3.2 Initial definition of GroBdd

A GroBdd is very similar to an actual ROBDD, the two main differences being that (1) it represents a vector of Boolean functions with a finite number of variables possibly of different arity and (2) on every edge there is a transformation (the "output negation" is an example of such transformations).

A Reduced Ordered Binary Decision Diagram is a directed acyclic graph $(V \cup T, \Psi \cup E)$ representing a vector of Boolean functions $F = (f_1, ..., f_k)$. Nodes are partitioned into two sets : the set of internal nodes V and the set of terminal nodes T. Every internal node $v \in V$ has two outgoing edges respectively denoted $if0$ and $if1$. Every internal node $v \in V$ has a field *index* which represents a unique identifier associated to each node. Arcs are partitioned into two sets : the set of root arcs Ψ and the set of internal arcs E. There is exactly k root arcs, a root arc is denoted Ψ_i with $0 \leq i < k$, informally, Ψ_i is the root of the GroBdd representing f_i. Every arc has a transformation descriptor field γ and a destination node denoted *node*.

We denote $\rho(\gamma) : \mathbb{F}_n \longrightarrow F_m$ the semantic interpretation of the transformation descriptor γ. We define $\phi(node)$ the semantic of the node *node* and $\psi(arc)$ the semantic of the arc *arc* as follow:

- $\forall i, f_i = \psi(\Psi_i)$
- $\forall arc \in \Psi \cup E, \psi(arc) = \rho(arc.\gamma)(\phi(arc.node))$
- $\forall node \in V, \phi(node) = \psi(node.if0) \star \psi(node.if1)$

We assume the function ϕ defined on all terminals T (we always assume, all terminal to have a different interpretation through ϕ).

3.3 Transformation Descriptor Set (TDS)

We denote $\mathbb{Y}$ the set of all transformation descriptor. We assume defined ρ the semantical interpretation of transformation descriptors, $\forall \gamma, \exists n, m \in \mathbb{N}, \rho(\gamma) \in \mathbb{F}_n \rightarrow \mathbb{F}_m$ (with $n \leq m$). In this section, we make the list of properties that the TDS must ensure to be correct.

3.3.1 Canonical

$$\forall \gamma, \gamma' \in \mathbb{Y}, (\gamma = \gamma') \Leftrightarrow (\rho(\gamma) = \rho(\gamma'))$$

We denote $\mathbb{Y}_{n,m} = \{\gamma \in \mathbb{Y} \mid \rho(\gamma) \in \mathbb{F}_n \longrightarrow \mathbb{F}_m\}$, and $\mathbb{Y}_{n,*} = \bigcup_{m \leq n} \mathbb{Y}_{n,m}$ and $\mathbb{Y}_{*,m} = \bigcup_{n \geq m} \mathbb{Y}_{n,m}$.

3.3.2 Separable

$$\forall \gamma \in \mathbb{Y}_{n,m}, \exists! \ \triangle_\gamma \in \mathbb{B}^m \rightarrow \mathbb{B}^n, \nabla_\gamma \in \mathbb{B}^m \rightarrow \mathbb{B} \rightarrow \mathbb{B},$$
$$\forall f \in \mathbb{F}_n, \forall x \in \mathbb{B}^m, \rho(\gamma)(f)(x) = \nabla(x, f(\triangle\ x))$$

This constraint allows any transformation to be represented as circuit which can be wrapped around the function. This constraint enforces transformations to be local. The function $\triangle$ is called the pre-process, and the function ∇ is called the post-process.

3.3.3 Composable (definition of C)

$$\forall \gamma \in \mathbb{Y}_{n,m}, \gamma' \in \mathbb{Y}_{m,l}, \exists \gamma'' \in \mathbb{Y}_{n,l}, \rho(\gamma') \circ \rho(\gamma) = \rho(\gamma'')$$

This constraint enforces $\mathbb{Y}_{n,n}$ to be stable by composition. Furthermore, it exists an algorithm $\mathtt{C} : \mathbb{Y}_{n,m} \rightarrow \mathbb{Y}_{m,l} \rightarrow \mathbb{Y}_{n,l}$, such that:

$$\forall \gamma \in \mathbb{Y}_{n,m}, \gamma' \in \mathbb{Y}_{m,l}, \rho(\gamma') \circ \rho(\gamma) = \rho(\mathtt{C}(\gamma, \gamma'))$$

For convenience, we denote $\gamma' \circ \gamma = \mathtt{C}(\gamma, \gamma')$.

3.3.4 Decomposable (definition of `A` and `S`)

For all $n \in \mathbb{N}$, we define $A_n = \mathbb{Y}_{n,n}$ the set of asymmetric transformations. For all $n, m \in \mathbb{N}$, it exists $S_{n,m} \subset \mathbb{Y}_{n,m}$ a set of transformations such that $\forall \gamma \in \mathbb{Y}_{n,m}, \exists a \in A_m, \exists! s \in S_{n,m}, \gamma = a \circ s$. The set $S_{n,m}$ is called the set of symmetric transformations. We ensure that $\forall n \in \mathbb{N}, S_{n,n} = \{Id_n = \mathbb{F}_n \to \mathbb{F}_n\}$.

definition : S-free
A Boolean function $f \in \mathbb{F}_m$ is said S-free, iff

$$\forall s \in S_{n,m}, \forall g \in \mathbb{F}_n, f = \rho(s)(g) \Rightarrow \rho(s) = Id_n$$

constraint : S-uniqueness

$$\forall f \in \mathbb{F}_n, f' \in \mathbb{F}_{n'}, s \in S_{n,m}, s' \in S_{n',m}, \rho(s)(f) = \rho(s')(f') \Rightarrow (s = s') \wedge (f = f')$$

(with f and f' being S-free)

constraint : S-uniqueness (equivalent definition)

$$\forall f \in \mathbb{F}_m, \exists! (s, \tilde{f}) \in (S_{n,m} \times \mathbb{F}_n), f = \rho(s)(\tilde{f})$$

(with $\tilde{f}$ being S-free)

definition : A-equivalent
Two Boolean functions $f, g \in \mathbb{F}_n$ are said A-equivalent, iff $\exists a \in A_n, f = \rho(a)(g)$. This relation is an equivalence relation (i.e. reflexive, symmetric, transitive) denoted $\sim_A$.

definition : A-invariant free
A Boolean function $f \in \mathbb{F}_n$ is said A-invariant free, iff $\forall a, a' \in A_n, \rho(a)(f) \neq \rho(a')(f)$.

constraint : S-free implies A-invariant free
For all function f, if f is S-free, then f is A-invariant free.

definition : A-reduced set of function
Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of Boolean functions. The set X is said A-reduced iff $\forall x_i, x_j \in X, (x_i \sim_A x_j) \Rightarrow (x_i = x_j)$.

definition : $\mathbb{I}_n$
For all $n \in \mathbb{N}$, we denote $\mathbb{I}_n$ the set of identifiers corresponding to node representing functions of arity n.

definition : $\mathbb{Y}-$`node`
A $\mathbb{Y}-$`node`$_{l,m,n}$ is a quadruple $(\gamma_0, I_0, \gamma_1, I_1) \in \mathbb{Y}_{l,n} \times \mathbb{I}_l \times \mathbb{Y}_{m,n} \times \mathbb{I}_m$. Let v be a $\mathbb{Y}-$`node`, we denote $v.\gamma_0$ (respectively $v.I_0$, $v.\gamma_1$ and $v.I_1$) the first (respectively second, third and fourth) component of v.

- For all $\mathbb{Y}-$`node` v, we always assume that functions $\phi(v.I_0)$ and $\phi(v.I_1)$ are `S`-free.

- We always assume the set $X = \{\phi(v.I_0) \mid v \in \mathbb{Y}-\texttt{node}\} \cup \{\phi(v.I_1) \mid v \in \mathbb{Y}-\texttt{node}\}$ to be `A-reduced`

We extend the definition of ϕ, with : for all $\mathbb{Y}-\texttt{node}$ v, $\phi(v) = \rho(v.\gamma_0)(\phi(v.I_0)) \star \rho(v.\gamma_1)(\phi(v.I_1))$.

constraint : terminal nodes are S-free, canonical and A-reduced (definition of $\texttt{E}_0$) For all terminal node $t \in T$, t is S-free (S-free). Moreover, $\forall t, t' \in T, \phi(t) = \phi(t') \Rightarrow t = t'$ (canonical). Furthermore, the set $\{\phi(t) \mid t \in T\}$ is A-reduced (A-reduced). Moreover, we define the function $\texttt{E}_0$ such $\forall b \in \mathbb{F}_0, \exists \gamma \in \mathbb{Y}_{0,0}, \exists t \in T, \texttt{E}_0(b) = \{\gamma = \gamma, node = I_t\}$ and $\psi(\texttt{E}_0(b)) = b$.

3.3.5 Buildable (definition of `B`)

Let X be a function, we denote I_X the identifier of an hypothetical node whose semantic interpretation is X.

We define `B` an algorithm over $\mathbb{Y}$ which respect the signature:

```
$\texttt{B} : \mathbb{Y}_{n_0,m} \times \mathbb{I}_{n_0} \longrightarrow \mathbb{Y}_{n_1,m} \times \mathbb{I}_{n_1} \longrightarrow$
    | ConsNode  $\mathbb{Y}_{n',m} \times (\mathbb{Y}_{n_x,n'} \times \mathbb{I}_{n_x}) \times (\mathbb{Y}_{n_y,n'} \times \mathbb{I}_{n_y})$
    | Merge  $\mathbb{Y}_{n_z,m} \times \mathbb{I}_{n_z}$
```

(with $x, y, z \in \{0, 1\}$)
Furthermore, for all $(\gamma_g, I_g, \gamma_h, I_h) \in \mathbb{Y}_{n_0,m} \times \mathbb{I}_{n_0} \times \mathbb{Y}_{n_1,m} \times \mathbb{I}_{n_1}$,

$$\texttt{B}(\gamma_g, I_g, \gamma_h, I_h) = \texttt{ConsNode}(\gamma, (\gamma', I_X), (\gamma'', I_Y)) \Rightarrow f = \rho(\gamma)\left(\rho(\gamma')(X) \star \rho(\gamma'')(Y)\right)$$

$$\texttt{B}(\gamma_g, I_g, \gamma_h, I_h) = \texttt{Merge}(\gamma''', I_Z) \Rightarrow f = \rho(\gamma''')(Z)$$

(with $X, Y, Z \in \{g, h\}$ and $f = \rho(\gamma_g)(g) \star \rho(\gamma_h)(y)$)

definition : a node is `B`-stable
Let G be a GroBdd, we denote v an internal node of G. We denote $\gamma_0 = v.if0.\gamma, I_0 = v.if0.node, \gamma_1 = v.if1.\gamma$ and $I_1 = v.if1.node$. The node v is said B-stable iff $\texttt{B}(\gamma_0, I_0, \gamma_1, I_1) = \texttt{ConsNode}(Id, (\gamma_0, I_0), (\gamma_1, I_1))$.

constraint : `B` is `B`-stable

$$\forall \gamma_f, I_f, \gamma_g, I_g, \texttt{B}(\gamma_f, I_f, \gamma_g, I_g) = \texttt{ConsNode}(\gamma, (\gamma_0, I_0), (\gamma_1, I_1))$$
$$\Rightarrow \texttt{B}(\gamma_0, I_0, \gamma_1, I_1) = \texttt{ConsNode}(Id, (\gamma_0, I_0), (\gamma_1, I_1))$$

Informally, when B returns a node, this node is `B`-stable.

constraint : $\mathbb{Y}-$`node` are `B`-stable

1. We assume all $\mathbb{Y}-\texttt{node}$ v to be `B`-stable

$$\texttt{B}(v.\gamma_0, v.I_0, v.\gamma_1, v.I_1) = \texttt{ConsNode}(Id, (v.\gamma_0, v.I_0), (v.\gamma_1, v.I_1))$$

constraint : B is S-free preserving
The algorithm `B` is said S-free preserving iff for all $\mathbb{Y}-\texttt{node}$ v, $\phi(v)$ is S-free.

constraint : B is A-reduction preserving

The algorithm `B` is said A-reduction preserving iff

$$\forall v, w \in \mathbb{Y}-\texttt{node}, \phi(v) \sim_A \phi(w) \Rightarrow v = w$$

3.4 Reduction Rules

We define a GroBdd model as the triple $(\mathbb{Y}, \texttt{C}, \texttt{B})$. A valid model must satisfy all the previously mentioned constraints.

In addition to the previous constraints, we define two reduction rules:

1. The syntactical reduction : all sub-graphs are different up to graph-isomorphism (i.e. all identical sub-graphs are merged)
2. The local semantic reduction : all internal node $v \in V$ is `B`-stable.
3. All node has at least one incoming arc. A GroBdd is said reduced if it satisfies the reduction rules.

In this section we prove:

1. For all vector of Boolean function F it exists a reduced GroBdd G representing it.
2. A reduced GroBdd G is semi-canonical, defined as :
   (a) For all node $v \in V$, $\phi(v)$ is S-free.
   (b) The set $X = \{\phi(v_1), \ldots, \phi(v_N)\}$ representing the set of the semantic interpretation of the set of internal nodes V is A-reduced.
   (c) $\forall(\gamma, I, \gamma', I') \in (\mathbb{Y}_{n,m} \times \mathbb{I}_n)^2, \psi(\{\gamma = \gamma, node = I\}) = \psi(\{\gamma = \gamma', node = I'\}) \Rightarrow (\gamma, I) = (\gamma', I')$ (with I and I' being indexes of nodes in G).
3. Between two reduced GroBdd G and G' representing the same vector of Boolean functions F, it exists a one-to-one mapping $\sigma : V \longrightarrow V'$ such that $\forall v, v' \in V \times V', \sigma(v) = v' \Rightarrow (\exists a \in A_*, \phi(v) = \rho(a)(\phi(v')))$.

3.4.1 Additional procedures

The `Cons` procedure

We define the `Cons` procedure as follow. Let G be a GroBdd and F be its vector of root arcs of size k.
Let i and j be indexes of this vector. We denote $\gamma_i = \Psi_i.\gamma$, $\gamma_j = \Psi_j.\gamma$ and $I_i = \Psi_i.node$, $I_j = \Psi_j.node$.

- If $\texttt{B}(\gamma_g, I_g, \gamma_h, I_h) = \texttt{ConsNode}\ (\gamma, (\gamma', I_X), (\gamma'', I_Y))$. We define the node $N = \{if0 = \{\gamma = \gamma', node = I_X\}, if1 = \{\gamma = \gamma'', nodes = I_Y\}\}$.
  - If the node N already exists in G, we retrieve its identifier I, we define G' as a copy of G with a new root arc $\Psi_k = \{\gamma = \gamma, node = I\}$
  - Otherwise, we define G' as a copy of G. We generate a new identifier, and associate it the node N within the *unique table* of G'. We add a new root arc $\Psi_k = \{\gamma = \gamma, node = I\}$ to G'.
- Otherwise, $\texttt{B}(\gamma_g, I_g, \gamma_h, I_h) = \texttt{Merge}\ (\gamma, I_Z)$. We define G' as a copy of G with a new root arc $\Psi_k = \{\gamma = \gamma''', node = I_Z\}$.

The `Remove` procedure

We define the `Remove` procedure as follow. Let G be a GroBdd and F be its vector of root arcs of size k. Let i be an index of this vector.
We define G' as a copy of G with its vector of root arcs $F' = (\Psi_1, \dots, \Psi_{i-1}, \Psi_{i+1}, \dots, \Psi_n)$.
In an iterative process, we remove nodes which have no incoming arc, until all nodes have at least one incoming arc (in order to satisfy the third reduction rule). As G' is a subset of G, thus, G' satisfies the first and second reduction rule.

The `Reduction` procedure

Let G be a GroBdd and F be its vector of root arcs F. Using an iterative process, we remove nodes which have no incoming arc, until all nodes have at least one incoming arc (satisfying the third reduction rules). The `Reduction` procedure consists in going through the GroBdd G (starting with nodes with the smallest depth), applying the `Cons` procedure on each node creating a new GroBdd G'. The GroBdd G' is by construction equivalent to G, however, the GroBdd G' satisfies all three reduction rules.

The `Merge` procedure

We define the `Merge` procedure as follow. Let G and G' be two GroBdds (based on the same GroBdd model). We define G'' a GroBdd which is the union of both GroBdds. We define $F'' = (\Psi_1, \dots, \Psi_n, \Psi'_1, \dots, \Psi'_{n'})$. Due to possible conflicts on identifiers, we re-generate identifiers of the nodes in G''. Finally, we apply the `Reduction` procedure on G''.

3.4.2 Existence

We inductively define the procedure `E` with:

- $\forall b \in \mathbb{F}_0, \mathtt{E}(b) = \mathtt{E}_0(b)$
- Let f be a Boolean function of arity n (with $n \geq 1$). Let G be a GroBdd representing functions f_0 (the negative restriction of f according to its first variable) and f_1 (the positive restriction of f according to its first variable.) by using the procedure `E` on f_0 and f_1. Let G' be the output of the `Cons` procedure on G, in order to create $f = f_0 \star f_1$ (expansion theorem).
  $E(f) = G'$ The GroBdd G' satisfies the reduction rules:
  - If no node is created, the proof is straightforward.
  - If a node is created, this node is syntactically unique by definition of `Cons` and is `B`-stable (as `B` is `B`-stable)

By construction, the procedure `E` (generalized to accept a vector of function as input) returns a GroBdd satisfying the reduction rules.

3.4.3 Semi-Canonical

Let G be a reduced GroBdd.

S-free and A-reduced For all node $v \in V$, we denote $h(v) = max(h(v.if0.node), h(v.if1.node))$ with $\forall t \in T, h(t) = 0$. For all $n \in \mathbb{N}$, we define $V_n = \{v \in V \mid h(v) \leq n\}$ For all $n \in \mathbb{N}$, we define the recurrence hypothesis $H(n)$:

- For all $v \in V_n$, $\phi(v)$ is S-free.
- The set of Boolean functions $X = \{\phi(v) \mid v \in V_n\}$ is A-reduced.

Initialization We prove $H(0)$ using the constraints that (1) terminals are S-free and (2) the set of terminal nodes is A-reduced.

Induction Let $n \in \mathbb{N}$, we assume $\forall k \leq n, H(k)$. Let v be a node of depth $n+1$, thus the depth of $v.if0.node$ and $v.if1.node$ is lower than n (we can apply the recurrence hypothesis). Therefore, the quadruple $\bar{v} = (v.if0.\gamma, v.if0.node, v.if1.\gamma, v.if1.node)$ is a $\mathbb{Y}-$`node`. Thus, using the constraints that `B` is S-free preserving, we prove that $\phi(v) = \phi(\bar{v})$ is S-free. Let v' be a node of depth $k \leq n+1$, we can prove that the quadruple $\bar{v'} = (v'.if0.\gamma, v'.if0.node, v'.if1.\gamma, v'.if1.node)$ is a $\mathbb{Y}-$`node`. Therefore, we can use the constraint that `B` is A-reduction preserving to prove that $\phi(\bar{v}) \sim_A \phi(\bar{v'}) \Rightarrow \phi(\bar{v}) = \phi(\bar{v'})$ However, $\phi(v) = \phi(\bar{v})$ and $\phi(v') = \phi(\bar{v'})$ Thus, $\forall v, w \in V_{n+1}, \phi(v) \sim_A \phi(v') \Rightarrow \phi(v) = \phi(v')$. Thus, the set of Boolean function $X = \{\phi(v) \mid v \in V_{n+1}\}$ is A-reduced. Therefore $(\wedge_{k \leq n} H(k)) \Rightarrow H(n+1)$.

Using the strong recurrence theorem, we prove that $\forall n \in \mathbb{N}, H(n)$. Therefore proving properties (2.a) "all nodes are S-free" and (2.b) "the set $X = \{\phi(v_1), \dots, \phi(v_N)\}$ is A-reduced".

Semantic Reduction We prove the property "$\forall (\gamma, I, \gamma', I') \in (\mathbb{Y}_{n,m} \times \mathbb{I}_n)^2, \psi(\{\gamma = \gamma, node = I\}) = \psi(\{\gamma = \gamma', node = I'\}) \Rightarrow (\gamma, I) = (\gamma', I')$ (with I and I' being indexes of nodes in G)" by induction on $n \in \mathbb{N}$ the arity of $f = \psi(\{\gamma = \gamma, node = I\})$.

Initialization The induction property holds for $n = 0$:
Let $f \in \mathbb{F}_0$, we assume it exists a quadruple $(\gamma, I, \gamma', I') \in (\mathbb{Y}_{n,m} \times \mathbb{I}_n)^2$ such that $f = \psi(\{\gamma = \gamma, node = I\}) = \psi(\{\gamma = \gamma', node = I'\})$. However, $\mathbb{Y}_{0,0} = \{Id_0\}$, therefore, γ and γ' are asymmetric transformation descriptors. Hence, $\rho(a)(\phi(I)) = \rho(a')(\phi(I'))$, thus $\phi(I) \sim_A \phi(I')$. Using the constraint, that terminals are A-reduced and canonical, we have that $\phi(I) = \phi(I')$, thus $I = I'$.

Induction Let $k \in \mathbb{N}$, we assume the induction property holds for all $n \leq k$, let prove it holds for $n = k+1$. Let f be a Boolean function, we assume it exists a quadruple $(\gamma, I, \gamma', I') \in (\mathbb{Y}_{n,m} \times \mathbb{I}_n)^2$ such that $f = \psi(\{\gamma = \gamma, node = I\}) = \psi(\{\gamma = \gamma', node = I'\})$. We decompose γ and γ' to their symmetric and asymmetric components: $\gamma = s \circ a$ and $\gamma' = s' \circ a'$. Using the S-uniqueness constraint, on $f = \rho(s)(\rho(a)(\phi(I))) = \rho(s')(\rho(a')(\phi(I')))$, we have $s = s'$ and $\rho(a)(\phi(I)) = \rho(a')(\phi(I'))$. Therefore, $\phi(I) \sim_A \phi(I')$, however, we proved that the set of the semantic interpretations of the nodes is A-reduced, thus $\phi(I) = \phi(I')$ Using the induction hypothesis (as $\phi(I)$ as an arity strictly smaller than $k+1$, thus smaller than k), we deduce that $I = I'$.

Therefore, applying the strong induction theorem, we prove the property (2.c).

3.4.4 Canonical modulo graph-isomorphism and A-equivalence

In order to prove that "Between two reduced GroBdd G and G' representing the same vector of Boolean functions F, it exists a one-to-one mapping $\sigma : V \longrightarrow V'$ such that $\forall v, v' \in V \times V', \sigma(v) = v' \Rightarrow (\exists a \in A_*, \phi(v) = \rho(a)(\phi(v'))$", we start by proving that within the `Merge` procedure, each node is replaced by an A-equivalent one. Secondly, using the `Merge` procedure without re-generating the identifiers of the nodes of G and using the property (2.c), we prove that the nodes of G' collapse on the nodes of G during the `Reduction` procedure, providing a one-to-one mapping. Details of this proof are cumbersome and not of great interest.

3.5 Conclusion

In this section, we introduced the concept of Generalized Reduction of Ordered Binary Decision Diagrams (GroBdd) and provided a set of necessary conditions and reduction rules, which guarantee the reduced structure to be semi-canonical. This approach still allows to perform the equality test in a reasonable amount of time: proportional to the size of the transformation descriptor, which (for practical reasons) should be polynomial in the arity of the post-transformation function. In the remaining of this report (if not mentioned otherwise), a GroBdd is to be assumed reduced. In the next section, we will introduce the "useless variable extraction" variant (or "U-extract" for short) as being a simple model (called "U") in the GroBdd formalism that we just introduced.

4 Useless Variable Extraction : GroBdd model NU

4.1 Motivation

There are several interests in extracting useless variables, the more obvious one is that it tends to reduce the number of nodes (relatively to a regular ROBDD representing the same function with the same order). Here is a list of other interesting properties:

- When formalizing a problem it may appear that it exists a sub-problem that is present several times but on different variables with the same relative order, for example: in the n-queens problem : "there is exactly one queen per line" is a constraint that either you replicate or solve once and replicate the result. With the "U-extract" variant, you can solve the problem once and then, by creating the appropriate arc, replicating the solution without creating new nodes.

- Given a function, you can know which variables are useless, just by looking at the transformation descriptor, thus without going through the all structure.

4.2 Definition of the Transformation Descriptor Set (TDS)

We define the Transformation Descriptor Set (TDS) of the "output Negation" + "Useless variable extraction" model (or "NU" model for short) by $\mathbb{Y} = \mathbb{B} \times \bigcup_{n\in\mathbb{N}} \mathbb{B}^n$. Let γ be a transformation, we denote $\gamma.neg \in \mathbb{B}$ the first component of γ and $\gamma.sub \in \bigcup_{n\in\mathbb{N}} \mathbb{B}^n$ the second component of γ. Furthermore, we denote $\gamma.sub_k \in \mathbb{B}$ the k-th component of the Boolean vector $\gamma.sub$. For all transformation descriptor $\gamma \in \mathbb{B} \times \mathbb{B}^m$, we define $m(\gamma) = m$ and $n(\gamma) = \sum_{0\leq k<m} \gamma.sub_k$. We define $\mathbb{Y}_{n,m} = \{\gamma \in \mathbb{Y} \mid n(\gamma) = n \wedge m(\gamma) = m\}$. For all transformation descriptor $\gamma \in \mathbb{Y}_{n,m}$, we denote $\rho(\gamma) \in \mathbb{F}_n \to \mathbb{F}_m$ its semantic interpretation. Informally, let $\gamma \in \mathbb{Y}_{n,m}$ be a transformation descriptor and $f \in \mathbb{F}_n$ be a function without useless variables, then the k-variable of the Boolean function $\rho(\gamma)(f)$ is useless iff $\gamma.sub_k$ is false.

Additionally, we define the set of terminals $T = \{0\}$, we define $\phi(0) = () \longrightarrow 0 \in \mathbb{F}_0$, we define $\forall b \in \mathbb{B}, E_0(() \longrightarrow b \in \mathbb{F}_0) = \{\gamma = \{neg = b, sub = ()\}, node = I_0\}$.

4.2.1 Definition of the semantic interpretation

For all transformation descriptor $\gamma \in \mathbb{Y}_{n,m}$, we denote $S_\gamma = (i_1 < i_2 < \cdots < i_n)$ the exhaustive list for which $\gamma.sub_k$ is true. Therefore, for all function $f \in \mathbb{F}_n$ we define $\rho(\gamma)(f) = (x_1, \ldots, x_m) \longrightarrow \gamma.neg \oplus f(x_{i_1}, x_{i_2}, \ldots, x_{i_n})$. For convenience reasons, we allow ourselves to define the $\gamma.sub$ component using the S_γ notation instead of the Boolean vector one.

transformations are canonical

Let $\gamma', \gamma'' \in \mathbb{Y}_{n,m}$ be a pair of transformation descriptors, such that for all function $f \in \mathbb{F}_n$, $\rho(\gamma')(f) = \rho(\gamma'')(f)$. Let $f \in \mathbb{F}_n$ a function with no useless variable. We denote $S_{\gamma'} = (i'_1, \ldots, i'_n)$ and $S_{\gamma''} = (i''_1, \ldots, i''_n)$. We absurdly assume that it exists an index k such that $i'_k \notin S_{\gamma''}$. Therefore, in one hand, the i'_k-th variable of $\rho(\gamma')(f)$ is useless. In the other hand, the i'_k-th variable of $\rho(\gamma'')(f)$ is mapped to some variable of f which by definition has no useless variable, therefore, the i_k-th variable of $\rho(\gamma'')(f)$ is not useless. Hence, it leads to a contradiction as $\rho(\gamma'')(f) = \rho(\gamma'')(f)$ and that useless and not useless are incompatible states. Therefore, it does not exist such an index k, thus, $\gamma'.sub = \gamma''.sub$. We absurdly assume that $\gamma'.neg \neq \gamma''.neg$, thus $\gamma'.neg = \neg\gamma''.neg$. Hence, it leads to a contradiction as $\rho(\gamma')(f)(\vec{0}) = \rho(\gamma'')(f)(\vec{0})$ and $\rho(\gamma')(f)(\vec{0}) = \neg\rho(\gamma'')(f)(\vec{0})$. Therefore, $\gamma'.neg = \gamma''.neg$. Hence, $\gamma' = \gamma''$.

transformations are separable

Let $\gamma \in \mathbb{Y}_{n,m}$ be a transformation descriptor, we denote $S_\gamma = (i_1, \ldots, i_n)$. We define the function $\triangle_\gamma \colon \mathbb{B}^m \longrightarrow \mathbb{B}^n$ by $\forall x \in \mathbb{B}^m, \triangle_\gamma\ (x_1, \ldots, x_m) = (x_{i_1}, \ldots, x_{i_n})$. And define the function $\nabla_\gamma : \mathbb{B}^m \longrightarrow \mathbb{B} \longrightarrow \mathbb{B}$ by $\forall x \in \mathbb{B}^m, y \in \mathbb{B}, \nabla_\gamma(x, y) = \gamma.neg \oplus y$. Hence, $\forall \gamma \in \mathbb{Y}_{n,m}, \forall f \in \mathbb{F}_n, \forall x \in \mathbb{B}^m, \rho(\gamma)(f)(x) = \nabla_\gamma(x, f(\triangle_\gamma\ (x)))$.

transformation are composable

Let $\gamma \in \mathbb{Y}_{n,m}$ and $\gamma' \in \mathbb{Y}_{m,l}$, we denote $S_\gamma = (i_1, \ldots, i_n)$ and we denote $S_{\gamma'} = (i'_1, \ldots, i'_m)$. We define $C(\gamma, \gamma') = \gamma''$ with $\gamma'' = \{neg = \gamma.neg \oplus \gamma'.neg, sub = (i'_{i_1}, i'_{i_2}, \ldots, i'_{i_n})\}$. We can prove that $\forall f \in \mathbb{F}_n, \rho(\gamma')(\rho(\gamma)(f)) = \rho(C(\gamma, \gamma'))(f)$.

transformations are decomposable

We denote $A_n = \mathbb{B} \times \{1\}^n \subset \mathbb{Y}_{n,n}$ (with $n \in \mathbb{N}$) and $S_{n,m} = \{0\} \times \{x \in \mathbb{B}^m \mid \sum_k x_k = n\}$. We can prove that $\forall \gamma \in \mathbb{Y}_{n,m}, \exists a \in A_m, \exists ! s \in S_{n,m}, \gamma = a \circ s$. We can notice that, for all $n \in \mathbb{N}$, $\rho(A_n) = \{Id, \neg\}$.

an S-free function is A-invariant free

We absurdly assume that it exists a Boolean function $f \in \mathbb{F}_n$ without useless variable (i.e. S-free), and a pair of distinct asymmetric transformations $a, a' \in A$ such that $\rho(a)(f) = \rho(a')(f)$. As A_n is of cardinal 2, we enumerate all cases, which lead to a contradiction as $f \neq \neg f$. Therefore f is A-invariant free.

terminal nodes are S-free

The function $\phi(0) = () \longrightarrow 0$ has no variable therefore, no useless variable, thus is S-free.

the set of semantic interpretation of terminal nodes is A-reduced

There is only one terminal node, therefore the set is A-reduced.

4.2.2 Definition of the building algorithm B

We define the building algorithm B as follow:

```
B($\gamma_0, I_0, \gamma_1, I_1$){
  if $(I_0 = I_1) \wedge \gamma_0 = \gamma_1$ then{
    Merge $\{\gamma = \{neg = \gamma_0.neg, sub = (0, \gamma_0.sub_1, \ldots, \gamma_0.sub_n)\}, node = I_0\}$
  }else{
    we define $\gamma, \gamma'_0, \gamma'_1$ by:
      $\gamma.neg = \gamma_0$
      $\gamma'_0.neg = 0$
      $\gamma'_1.neg = \gamma_0 \oplus \gamma_1$
      $S_\gamma = \{i_1 < ... < i_{m'}\} = S_{\gamma_0} \cup S_{\gamma_1}$
      $S_{\gamma'_0} = \{j_1 < ... < j_n\}$ the indexes of $S_{\gamma_0}$ in $S_\gamma$.
      $S_{\gamma'_1} = \{k_1 < ... < k_{n'}\}$ the indexes of $S_{\gamma_1}$ in $S_\gamma$.
    ConsNode $(\gamma, (\gamma'_0, I_0), (\gamma'_1, I_1))$
  }
}
```

The proof that B is correct is left to the reader. Furthermore, we prove that $S_{\gamma'_0} \cup S_{\gamma'_1} = \{1, \ldots, m'\}$.

B is B-stable We have to prove that

$$\mathtt{B}(\gamma_0, I_0, \gamma_1, I_1) \in \mathbb{Y}_{n,m}) = \mathtt{ConsNode}(\gamma, (\gamma'_0, I'_0), (\gamma'_1, I'_1))$$

$$\Rightarrow \mathtt{B}(\gamma'_0, I'_0, \gamma'_1, I'_1) = \mathtt{ConsNode}(Id, (\gamma'_0, I'_0), (\gamma'_1, I'_1))$$

We absurdly assume that $\mathtt{B}(\gamma'_0, I'_0, \gamma'_1, I'_1) \neq \mathtt{ConsNode}(Id, (\gamma'_0, I'_0), (\gamma'_1, I'_1))$:

- We absurdly assume that $\mathtt{B}(\gamma'_0, I'_0, \gamma'_1, I'_1) = \mathtt{Merge}(\gamma, I)$. Therefore, by definition of B, $\gamma'_0 = \gamma'_1$ and $I'_0 = I'_1$, thus $\gamma_0 = \gamma_1$ and $I_0 = I_1$, hence it leads to a contradiction as $\mathtt{B}(\gamma_0, I_0, \gamma_1, I_1) \in \mathbb{Y}_{n,m}) = \mathtt{Merge}\ldots$.

- Therefore, $\mathtt{B}(\gamma'_0, I'_0, \gamma'_1, I'_1) = \mathtt{ConsNode}(\gamma', (\gamma''_0, I''_0), (\gamma''_1, I''_1))$:
  - We absurdly assume that $\gamma' \neq Id$. However, $\gamma'.neg = \gamma'_0.neg = 0$ (by definition of B, therefore $\gamma'.sub \neq (1, \ldots, 1) \in \mathbb{B}^n$. Thus, it exists i such that $\gamma'.sub_i = 0$, thus, it exists an index k' such that $k' \notin S_{\gamma'_0} \cup S_{\gamma'_1}$, which leads to a contradiction as $S_{\gamma'_0} \cup S_{\gamma'_1} = \{1, \ldots, m'\}$.
  - Therefore, $\gamma' = Id$. Thus, $\gamma''_0 = \gamma'_0$, $\gamma''_1 = \gamma'_1$, $I''_0 = I'_0$ and $I''_1 = I'_1$. Hence it leads to a contradiction as we assumed that $\mathtt{B}(\gamma'_0, I'_0, \gamma'_1, I'_1) \neq \mathtt{ConsNode}(Id, (\gamma'_0, I'_0), (\gamma'_1, I'_1))$.

Therefore, B is B-stable.

B is S-free preserving

Let $v = (\gamma_0, I_0, \gamma_1, i_1)$ be a $\mathbb{Y}-\mathtt{node}$, therefore, the Boolean functions $\phi(I_0)$ and $\phi(I_1)$ are S-free and $\phi(I_0) \sim_A \phi(I_1) \Rightarrow I_0 = I_1$ We absurdly assume that the Boolean function $\phi(v) = \rho(\gamma_0)(\phi(I_0)) \star \rho(\gamma_1)(\phi(I_1))$ is not S-free, thus, it exists an index k such that the k-th variable of $\phi(v)$ is useless.

- If $k = 0$, then, $\rho(\gamma_0)(\phi(I_0)) = \rho(\gamma_1)(\phi(I_1))$. Using the S-uniqueness constraint ($\phi(I_0)$ and $\phi(I_1)$ being S-free), we prove that $\phi(I_0) \sim_A \phi(I_1$, thus $I_0 = I_1$. Hence, leading to a contradiction as the $\mathbb{Y}-\mathtt{node}$ v is assumed to be B-stable.

- If $k > 0$, then, the $(k-1)$-th variables of the Boolean functions $\rho(\gamma_0)(\phi(I_0))$ and $\rho(\gamma_1)(\phi(I_0))$ are useless. Using the S-uniqueness constraint ($\phi(I_0)$ and $\phi(I_1)$ being S-free), we prove that $\gamma_0.sub_{k-1} = \gamma_1.sub_{k-1} = 0$. Hence, leading to a contradiction as the $\mathbb{Y}-\mathtt{node}$ v is assumed to be B-stable.

B is A-reduction preserving

Let $v = (\gamma_0, I_0, \gamma_1, I_1)$ and $v' = (\gamma'_0, I'_0, \gamma'_1, I'_1)$ be $\mathbb{Y}-\mathtt{nodes}$, such that $\phi(v) \sim_A \phi(v')$. Thus, it exists an asymmetric transformation descriptor a such that $\phi(v) = \rho(a)(\phi(v'))$ Therefore, the Boolean functions $\phi(I_0), \phi(I_1), \phi(I'_0)$ and $\phi(I'_1)$ are S-free and the set of Boolean functions $\{\phi(I_0), \phi(I_1), \phi(I'_0), \phi(I'_1)\}$ is A-reduced. Thus, using the expansion theorem, $\rho(\gamma_0)(\phi(I_0)) = \rho(a \circ \gamma'_0)(\phi(I'_0))$ and $\rho(\gamma_1)(\phi(I_1)) = \rho(a \circ \gamma'_1)(\phi(I'_1))$. Using the S-uniqueness constraint, we prove that $\phi(I_0) \sim_A \phi(I'_0)$ and $\phi(I_1) \sim_A \phi(I'_1)$ (and $\gamma_0 = a \circ \gamma'_0$ and $\gamma_1 = a \circ \gamma'_1$), thus, using the A-reduction property, $I_0 = I'_0$ and $I_1 = I'_1$.

Therefore, $v' = (\gamma'_0, I_0, \gamma'_1, I_1)$. We proved that $\gamma_0 = a \circ \gamma'_0$, however, v and v' and B-stable, thus $\gamma_0.neg = \gamma'_0.neg = 0$, thus $a = Id$. Hence, $\gamma_1 = \gamma'_1$. We proved that $v = v'$.

4.2.3 The GroBdd model NU is semi-canonical

In this section, we proved that the model "NU" ("output negation" and "useless variable extraction") satisfies all the constraints formulated in Section 4 (Introduction of GroBdd). Therefore, we proved that a GroBdd which uses the "NU" model is semi-canonical. Actually, as the building algorithm B only uses the equality test between node's identifier, we can prove that the structure is canonical (up to graph isomorphism). In the next section, we compare representations generated by this new variant and regular ROBDD (with just the "output negation" variant) on three different benchmarks in terms of number of nodes and estimated memory cost.

4.3 Results

In this section, we start by quickly presenting our implementation of previous concepts, then, average results of our approach on three different benchmarks (involving circuits and CNF formulas), finally, we show several examples for which the reduction is significant.

4.3.1 Implementation details

Our implementation in OCaml of the previous concepts is available on GitHub as part of the DAGaml framework within the `grobdd` branch [7]. The core program, i.e. the GroBdd abstraction, is about 1 900 lines of code. Implementation details for models is about 6 100 lines of code, with 728 lines for the model "NU", the remaining lines implementing models "NNI" (1 181 lines, mentioned in Section 4, is a generalization of "inputs negation"), "NU-X" (2 587 lines, mentioned in Section 4, allows to extract "1-predictions"), but also regular ROBDD (318 lines) and ZBDD (201 lines). Various useful tools are implemented (about 4 300 lines). All in all, this project is about 12 400 lines long. The implementation is designed in order to maximize code reusing through several layers of abstraction. This choice was made in order to reduce the size of the source code and maximizing the chance of finding evidence of bugs if one was to be introduced. Additionally, a significant part of the source code and computation time is dedicated to self-checking the consistency of the operation, in order to further reduce the risk of undetected issue. During this project, we focused our efforts into finding and validating new generalizations ("NU", "NNI" and "NU-X" so far) which would reduce the number of nodes. However, we did not spent much time on minimizing the memory cost of our implementation, therefore, the estimated memory cost introduces in the next section should be considered as an indicator. Especially, because the nodes in a regular ROBDD are of constant size (e.g. 22 bytes in Minoto et al. implementation [12]), however the model "NU" ("NNI" and "NU-X" as well) use a variable size encoding of the transformation descriptor, which should complexifies the memory management of an industrial-proof software using these concepts.

4.3.2 Results

We compare the number of nodes and the estimated memory cost for each one of the following model : `N` (regular ROBDD + "output negation" variant), `Z` (Zero-suppressed BDD), `NU` (GroBdd with "output negation" and "useless variable extraction"), `NNI` ("negation negation invariant", mentioned in Section 4, allows to complement inputs within transformation descriptors) and `NU-X` ("output negation", "useless variable" and "1-prediction", this variant allows to capture 'logical deductions' similar to what is done in SAT-solver with the unit propagation). The results with models `NNI` and `NU-X` are here, in order to prove that the GroBdd formalism can be used to define other models of significant interest.

We tested out implementation on three different benchmarks : `lgsynth91` [15], `iscas99` [5] and `satlib/uf20-91` [6]. Benchmarks `lgsynth91` and `iscas99`, which represent various circuits, were pre-processed using the framework of logic synthesis ABC [1], in order to turn an arbitrary circuits into a simpler to parse Anf Inverter Graph (AIG). The benchmark `satlib/uf20-91` represents a set of 1 000 satisfiable CNF formulas with 20 variables and 91 clauses. Results are given relatively to the model `N`. In columns "#node" we have the relative number of nodes and in columns "**memo**" the relative estimated memory cost.

	`lgsynth91`		`iscas99`		`satlib/uf20-91`	
model	#node	**memo**	#node	**memo**	#node	**memo**
`Z`	+233%	+203%	+162%	+135%	-41%	-42%
`NU`	-25%	-42%	-25%	-41%	-3%	-3%
`NNI`	-60%	-64%	-57%	-61%	-29%	-14%
`NU-X`	-64%	-68%	-55%	-58%	-93%	-95%

Conclusion

ROBDD allows to efficiently manipulate functions appearing in various fields of computer science such as: Bounded Model Checking, Planning, Software Verification, Automatic Test Pattern Generation, Combinational Equivalence Checking or Combinatorial Interaction Testing.

However, ROBDD manipulation is memory intensive and various variants exist (such as "output negation") in order to reduce the memory cost.

In this report, we introduced a new class of variant called GroBdd (Generalized Reduction of Ordered Binary Decision Diagram). We presented a set of constraints which ensure a GroBdd to be semi-canonical (i.e. canonical up to graph-isomorphism and A-equivalence, introduced in Section 4). Moreover, we defined a GroBdd model called "NU" ("output Negation" and "Useless variable extraction") which allows to significantly reduce the number of nodes (relatively to a regular ROBDD with the "output negation" variant).

Using our implementation in OCaml against several benchmarks, we observed an average reduction of 25% when representing circuits, and 3% when representing generated CNF formula. Furthermore, we estimated the memory cost of this variant, in average, we observed a 40% reduction when representing circuits and 3% when representing CNF formulas.

References

[1] Berkeley Logic Synthesis and Verification Group, Berkeley, Calif. *ABC: A System for Sequential Synthesis and Verification.* `http://www.eecs.berkeley.edu/~alanmi/abc/`.

[2] Karl S. Brace, Richard L. Rudell, and Randal E. Bryant. Efficient implementation of a BDD package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, DAC '90, pages 40–45, New York, NY, USA, 1990. ACM.

[3] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 35(8):677–691, August 1986.

[4] Jerry R. Burch and David E. Long. Efficient boolean function matching. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '92, pages 408–411, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[5] F. Corno, M.S. Reorda, and G. Squillero. Rt-level itc'99 benchmarks and first atpg results. *Design Test of Computers, IEEE*, 17(3):44–53, Jul 2000.

[6] Holger H. Hoos and Thomas Stützle. Satlib: An online resource for research on sat. pages 283–292. IOS Press, 2000.

[7] Thibault J. Abstract manipulations of directed acyclic graph using ocaml. `https://github.com/JoanThibault/DAGaml/tree/grobdd-dev`, 2016.

[8] D. Jankovic, R. S. Stankovic, and R. Drechsler. Decision diagram optimization using copy properties. In *Proceedings Euromicro Symposium on Digital System Design. Architectures, Methods and Tools*, pages 236–243, 2002.

[9] D. M. Miller and R. Drechsler. Dual edge operations in reduced ordered binary decision diagrams. In *Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*, volume 6, pages 159–162 vol.6, May 1998.

[10] D. M. Miller and R. Drechsler. Implementing a multiple-valued decision diagram package. In *Proceedings. 1998 28th IEEE International Symposium on Multiple- Valued Logic (Cat. No.98CB36138)*, pages 52–57, May 1998.

[11] D. M. Miller and R. Drechsler. On the construction of multiple-valued decision diagrams. In *Proceedings 32nd IEEE International Symposium on Multiple-Valued Logic*, pages 245–253, 2002.

[12] S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. In *27th ACM/IEEE Design Automation Conference*, pages 52–57, Jun 1990.

[13] Alan Mishchenko. An introduction to zero-suppressed binary decision diagrams. Technical report, in 'Proceedings of the 12th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, 2001.

[14] Fabio Somenzi. Binary decision diagrams. 1999.

[15] S. Yang. Logic synthesis and optimization benchmarks user guide: Version 3.0. Technical report, 1991.