A Generalized Reduction of Ordered Binary Decision Diagram

Joan Thibault

June 9, 2017

Abstract

Reduced Ordered Binary Decision Diagram ((RO)BDD) [2,12] 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 to main issues: (1) their representation is memory expansive 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) [11], Multi-valued Decision Diagram (MDD) [8, 9] and variations of the reduction rules such as "output inverter" [1], "input negation" [10], "shifting variables" [10], "dual edges" [7] or "copy node" [6].

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/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 [3, 5, 13]. 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

1 Introduction

Nowadays, it exists many critical systems which 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 of minimizing 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 expansive) 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, their 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.

ROBDD 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 expansive 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 inverter" variant [1], 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" [10] (each edge can complement the first locally first input), "shifting variables" [10] (each edge store the number of useless variables before the next significant variables) or "dual edge" [7] (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 inverter" variant, we introduce a new variant which allows to extract useless variables (a.k.a non-support variables). A useless variables, is a variable which 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 "extraction useless variables" or "U-extract" for short.

This report will be organized as follows. In Section 1, we formally introduce

Boolean functions and useless variable. In Section 2, we introduce ROBDDs. In Section 3, we introduce the "U-extract" variant and prove that it maintains the canonicity. In Section 4, we expose an estimations of improvements on three different benchmarks [3, 5, 13] using our implementation in OCaml.

2 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 $n \in \mathbb{N}$ 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 $f[i \leftarrow b](x_1, \ldots, x_n) = f(x_0, \ldots, x_{i-1}, b, x_i, \ldots, x_n)$. $f[i \leftarrow 0]$ (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 $(f \star_i g)(x_0, \ldots, x_{i-1}, y, x_i, \ldots, x_n) = y \longrightarrow_S f(x_0, \ldots, x_n), g(x_0, \ldots, x_n)$.

We denote $\star = \star_0$

Nb: $(f \star_i g)[i \leftarrow 0] = f$ 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

We define, the support set of a function f, the set of variable index i such that $f[i \leftarrow 0] \neq f[i \leftarrow 1]$. A variable that does not belong to the support, is said to be a non-support variable or useless variable. Thus, the i-th variable of f is useless iff $f[i \leftarrow 0] = f[i \leftarrow 1]$.

3 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 $F = (f_1, ..., f_k)$ other an infinite set of variables. 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 represent the index of a variable and two outgoing edges respectively denoted $if0$ and $if1$. When using the "output inverter" variant, there is only one terminal called 0, which represent the functions which always return 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 a proof of this theorem, is available in the review of Somenzi et al. [12].

3.1 Effective construction

In practice one does not build the decision tree and then reduces it. Rather, BDDs are created starting from the BDDs for the constants and the variables by application of 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. For instance, if we have $f(x_0, x_1, x_2, x_3) = x_1 \longrightarrow_S x_2, x_3$ 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. Its implementation is usually based on a dictionary of all BDDs nodes in existence in an application. 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. Whenever an operation needs to add to a BDD that it is building, it knows already the two nodes (say f_1 and f_0 represented by some identifier (usually implemented as pointers)) that going to be the new node's children and the decision variable v so it just has to check if the node (v, f_1, f_0) already exist and if so return its identifier and if not generate a new identifier and return it. 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}
  )
```

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

4.1 Motivation

In one hand, a ROBDD can be understood as an automaton recognizing a language compose 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 their is 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,

the integer multiplication has a quadratic AIG representing it, however, it is proven [2] that a ROBDD representing it has at least an exponential number of nodes (thus, of gates).

The "output inverter" variant, by adding expressiveness to edges, improves the conciseness. We want to go further in this approach by allowing more complex transformation on edges while maintaining canonicity. Such transformation can disturb variables' order of evaluation, allowing to represent "simple" decision processes in more concise way. Furthermore, some transformation (within the range of transformation allowed) could have their complexity drastically reduced. For example, using the "output inverter" variant, the complementation's time and space complexity which 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-predictions extraction" (or "X extract" for short).

The "U extract" variant allows to introduce useless variables, thus ensure 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 inverters" 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, introducing this variant breaks the canonicity, therefore, while working on a generalization of it, we do not discuss it further in this report.

We define a 1-prediction as follow: Let f 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 $f[i \leftarrow x] = y$. 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 function. This variant is compatible with the "output inverter" variant, however it over-complexifies the reduction rules, therefore, while reformulating reduction rules in a simpler way, we do not discuss it further in this report.

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.

4.2 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 their is a transformation (the "output inverter" 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 represent 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).

4.3 Transformation Descriptor Set (TDS)

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

4.3.1 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 $\mathbb{Y}_{*,m} = \bigcup_{n \geq m} \mathbb{Y}_{n,m}$.

4.3.2 Separable

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

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.

4.3.3 Composable (definition of $\mathtt{C}$)

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

This constraint enforces $\mathbb{Y}_{n,n}$ to be stable by composition. Furthermore, it exists an algorithm $\mathtt{C} : \mathbb{Y}_{n,m} \to \mathbb{Y}_{m,l} \to \mathbb{Y}_{n,l}$, 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 $\gamma' \circ \gamma = \mathtt{C}(\gamma, \gamma')$.

4.3.4 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 transformation.

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

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

constraint : S-uniqueness

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

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) denote $\sim_A$.

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 $\mathbb{Y}-\texttt{node}_{l,m,n}$ 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 $\mathbb{Y}-$`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 and A-reduced (definition of $\mathtt{E}_0$) For all terminal node $t \in T$, t is S-free. Furthermore, the set $\{\phi(t) \mid t \in T\}$ is A-reduced. Moreover, we define the function $\mathtt{E}_0$ such $\forall b \in \mathbb{F}_0, \exists \gamma \in \mathbb{Y}_{0,0}, \exists t \in T, \mathtt{E}_0(b) = \{\gamma = \gamma, node = I_t\}$ and $\psi(\mathtt{E}_0(b)) = b$.

4.3.5 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:

$$\begin{aligned}
&\mathtt{B} : \mathbb{Y}_{n_0,m} \times \mathbb{I}_{n_0} \longrightarrow \mathbb{Y}_{n_1,m} \times \mathbb{I}_{n_1} \longrightarrow \\
&\quad | \ \text{ConsNode} \ \ \mathbb{Y}_{n',m} \times (\mathbb{Y}_{n_x,n'} \times \mathbb{I}_{n_x}) \times (\mathbb{Y}_{n_y,n'} \times \mathbb{I}_{n_y}) \\
&\quad | \ \text{Merge} \ \ \mathbb{Y}_{n_z,m} \times \mathbb{I}_{n_z}
\end{aligned}$$

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

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

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

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

constraint : B is B-stable

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

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

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

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

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

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

constraint : B is A-reduction preserving
The algorithm B is said A-reduction preserving iff

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

4.4 Reduction Rules

We define a GroBdd model as the triple $(\mathbb{Y}, \mathtt{C}, \mathtt{B})$. 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 $\forall v, v' \in V \times V', \sigma(v) = v' \Rightarrow (\exists a \in A_*, \phi(v) = \rho(a)(\phi(v'))$.

4.4.1 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 $\mathtt{B}(\gamma_g, I_g, \gamma_h, I_h) =$ `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 with add N to the list of nodes of G' and create it a new identifier I, we add a new root arc $\Psi_k = \{\gamma = \gamma, node = I\}$.
- Otherwise, $\mathtt{B}(\gamma_g, I_g, \gamma_h, I_h) =$ `Merge` (γ, 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 arc of size k.
Let i be an index of this vector We define G' as a copy of G with its vector of root arc $F' = (\Psi_1, \ldots, \Psi_{i-1}, \Psi_{i+1}, \ldots, \Psi_n)$. 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 consist 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 GroBdd. We define $F'' = (\Psi_1, \ldots, \Psi_n, \Psi'_1, \ldots, \Psi'_{n'})$. Due to possible conflicts on identifiers, we re-generate identifiers of the nodes in G''. Finally, we apply the `Reduction` procedure on G''.

4.4.2 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.

4.4.3 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 function $X = \{\phi(v) \mid v \in V_n\}$ 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 $\forall k \leq nH(k)$. 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 $(land_{k \leq n} H(k) \Rightarrow H(n+1)$.

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 $X = \{\phi(v_1), \ldots, \phi(v_N)\}$ 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 $f = \psi(\{\gamma = \gamma, node = I\}) = \psi(\{\gamma = \gamma', node = I'\})$. 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 $\phi(I) = \phi(I')$ However, the only node representing function of arity 0 are terminal nodes, therefore $I = I'$.

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).

4.4.4 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 (1) proving that within the `Merge` procedure, each node is replace by an A-equivalent one then (2) 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.

4.5 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.

5 Pattern extraction : useless variables

5.1 Motivation

We may notice that the respective representations of $f_1 = f(x_{i_1}, ..., x_{i_n})$ $(i_1 < ... < i_n)$ and $f_2 = f(x_{j_1}, ..., x_{j_n})$ $(j_1 < ... < j_n)$ are very similar. The only difference being the values of the *var* field of each nodes within the representation of f. This suggest the possibility of actually using the same sub-graph to represent f, f_1 and f_2.

Suppose we have a BDD representing f, then in order to represent f_1 we just have to remember which variable are in the support of f (the relative order of variables in the support does not change). In this attribute, we will store the support set in the *coreduce* field of arcs.

Removing useless variables slightly complexify (e.g. we have to add an other reduction rule) the manipulation of BDDs, but has three advantages:

- reduce memory usage (in some cases there is an exponential gain, but on most cases its less significant).
- copying a function (if relative order of variables is unchanged) can be performed in constant time.

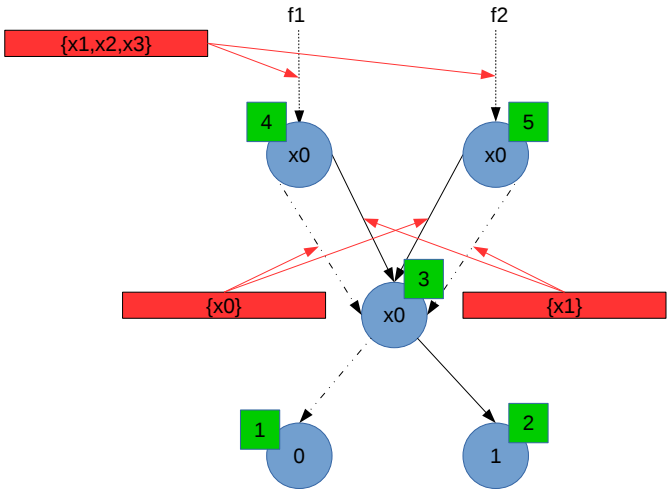

Figure 1: Same BDD as in Figure **??**, when detecting and removing useless variables. Each red rectangle contain the support set of each arc's function.

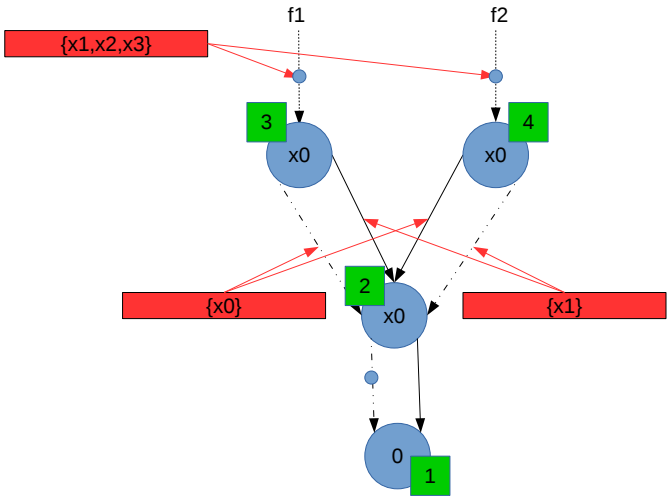

Figure 2: Same BDD as in Figure **??** and 1, with both additional reduction rules : complemented arcs and extraction of useless variables.

- checking $g = f$ modulo their useless variables -when having g and f- can be performed in constant time (not very useful).

5.2 Proof of Canonicity

We define the following reductions rules:

- $\forall v \in V, v.then.coreduce \cup v.else.coreduce = \{0, ..., n\}$
- $\forall v \in V \cup T, \phi(v)$ has no useless variable

For every function f, we denote $\tilde{f}$ the restriction of f to its support (maintaining the relative order of the remaining variables).

Existence Let $f \in \mathbb{B}^n \to \mathbb{B}$ be a Boolean Function. We recursively build a valid representation $R(f)$ of f which verify the reduction rules:

- $R((x_1, ..., x_n) \to b) = arc(coreduce = \emptyset, node = 1)$, where $b \in \mathbb{B}$
- $R(f) =$
  - We denote $S = \{i_1 < ... < i_k\}$ the support set of f.
  - $R(f) = arc(coreduce = S, node = node(then = R(\tilde{f}[0 \leftarrow 1]), else = R(\tilde{f}[0 \leftarrow 0])))$

Uniqueness We prove canonicity by induction on n the arity of the represented function.

Initialization Let R be a representative of a function of arity 0. We consider the root arc a, so $a.coreduce = \emptyset$ (the only set of cardinality lower than zero). However the `in-arity` of $\emptyset$ is 0, so $a.node$ represent a constant function. Hence either $R = arc(coreduce = \emptyset, node = 0)$ representative of the constant function 0 or $R = arc(coreduce = \emptyset, node = 1)$ representative of the constant function 1.

Induction Assuming that the hypothesis holds for every $0 \leq k \leq n$. Let R be a representative of f, we denote a its root arc.

- We assume that $a.coreduce \neq \{0, ..., n\}$. Then the arity of $g = \phi(a.node)$ is lower strictly lower than n. However g has a unique representation (induction hypothesis) and g has no useless variable (because of the reduction rules on applied on node $a.node$). So the only useless variable in f are the one that does not appear in $a.coreduce$. Therefore R is unique.
- We assume that $a.coreduce = \{0, ..., n\}$. We denote a_1 (respectively a_0) the root arc of the representation (unique by induction hypothesis) of $f_1 = \psi(a.node.then)$ (respectively $f_0 = \psi(a.node.else)$).
  - We assume that $f_1 = f_0$ then $a_1 = a_0$ then either induce that $0 \notin a.coreduce$ (which induce a contradiction with $S = \{0, ..., n\}$) or R does not satisfies the reduction rules.
  - Therefore $f_1 \neq f_0$.

* We assume that $a_1.coreduce \cup a_0.coreduce \neq \{0, ..., n-1\}$. Then it exist $0 \leq i \leq n-1$ such that x_i is useless for both f_1 and f_0. Hence $i+1$ is useless for f. So R does not satisfies the reduction rules.
* Therefore $a_1.coreduce \cup a_0.coreduce = \{0, ..., n-1\}$. Then $a = arc(coreduce = \{0, ..., n\},\ node = node(then = a_1,\ else = a_0)$. Therefore R is unique.

5.3 Reduction in construction

Moreover, as we already define the main algorithm for construction, we just have to write the specific sub-routines:

```
RDT-consensus(then, else, cmp){
  if (cmp = 0) & then.coreduce = else.coreduce then{
    we denote {i_1 < ... < i_k} = then.coreduce
    Solved arc(then = {i_1 + 1 < ... < i_k + 1}, node = then.node)
  }else{
    we denote U = {i_1 < ... < i_k} = then.coreduce ∪ else.coreduce
    we denote A = {a_1 < ... < a_l} the indexes of then.coreduce in
        U.
    we denote B = {b_1 < ... < b_m} the indexes of else.coreduce in
        U.
    Partial (coreduce = {0, i_1 + 1, ..., i_k + 1}, reduce = node(
       then = arc(coreduce = A, node = then.node), else =
       arc(coreduce = B, node = else.node) ) )
  }
}
```

```
compose({i_1 < ... < i_n}, arc(coreduce = {j_1 < ... < j_k}, node =
    node){
  return arc(coreduce = {i_{j_1} < ... < i_{j_k}}, node = node)
}
```

5.4 Reduction in operators

Moreover, as we already define the main algorithm for computing AND, we just have to write the specific sub-routines:

```
and-unpack(reduced-problem){
  return (reduced-problem.arcX, reduced-problem.arcY)
}
```

```
and-solver(arcX, arcY){
  if (terminal cases){
    return result //removing useless variables does not
       add terminal cases
  }else{
    we denote U = {i_1 < ... < i_k} = then.coreduce ∪ else.coreduce
    we denote A = {a_1 < ... < a_l} the indexes of then.coreduce in
        U.
```

```
7       we denote B = {b_1 < ... < b_m} the indexes of else.coreduce in
            U.
8       if arcX ≤ arcY
9       then{
10        Partial (coreduce = U, subproblem = (arcX = arc(
              coreduce = A, node = arcX.node), arcY = arc(
              coreduce = B, node = arcY.node)))
11      }else{
12        Partial (coreduce = U, subproblem = (arcY = arc(
              coreduce = A, node = arcX.node), arcX = arc(
              coreduce = B, node = arcY.node)))
13      }
14    }
15  }
```

6 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 inverter") in order to reduce the memory cost.

In this report, we introduce a new variant called "useless variable extraction" (a.k.a. "U-extract" for short). This variant, by capturing useless variable, allows to reduce the number of nodes.

Using our implementation in OCaml against several benchmarks, we obtain 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 observe a 30% reduction when representing circuits and 3% when representing CNF formula.

References

[1] 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.

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

[3] 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.

[4] Justin E. Harlow, III, and Franc Brglez. Design of experiments and evaluation of bdd ordering heuristics, 2001.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

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