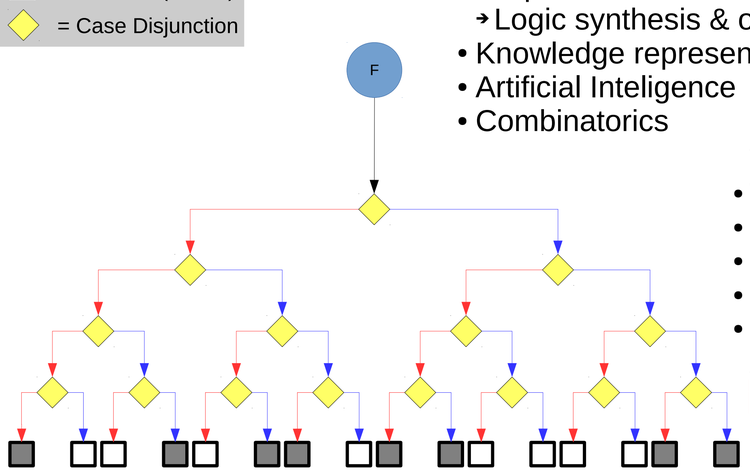
A Generalized Reduction of Ordered Binary Decision Diagram (GroBdd)

Joan Thibault, supervisor : Rolf Drechsler

Internship from 15/03/2017 untill 15/06/2017

[1] Shanon's Binary Decision Tree

= if False (if 0)
= if True (if 1)
= False (then 0)
= True (then 1)
= Case Disjunction



Efficiently manipulating Boolean functions is required in various fields :

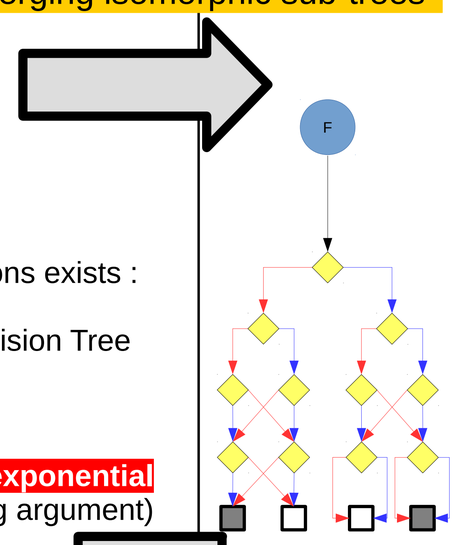
- Computer aided design
  - ➔ Equivalence checking
  - ➔ Logic synthesis & optimization
- Knowledge representation
- Artificial Inteligence
- Combinatorics

Various representations exists :

- Truth table
- Shanon's Binary Decision Tree
- CNF/DNF formulas
- Circuits
- …

All of them have an **exponential worst case** (counting argument)

merging isomorphic sub-trees



[2] Shannon's Binary Decision Diagram

Objective: find a representation which size grows with the function's complexity and allows to quickly apply operators. Reduced Ordered Binary Decision Diagrams (ROBDDs, cf. [4]) tend to fulfill these requirements:

- Constant time operator : functional equality
- Linear time (O(#variables)) operators: Evaluation, AnySAT, MinSAT, MaxSAT
- Linear time (O(#node)) operators: Partial evaluation, conversion to {CNF, DNF, Circuits}, intputs/output negation
- Quadratic time (O(#node)) operators: Binary Boolean operators {AND, XOR}, quantification

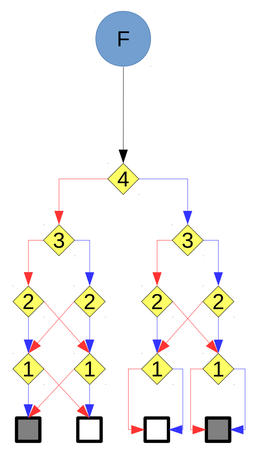
For aesthetic reasons, we do not merge all terminals

[3] Ordered Binary Decision Diagram (OBDD)

Labeling nodes with inputs' indexes

However, ROBDDs have several issues:

1) The representation is not stable by input permutation (unlike CNF/DNF formulas and circuits).
2) There exists polynomial size circuits which have no polynomial size ROBDD, e.g. integer multiplication.



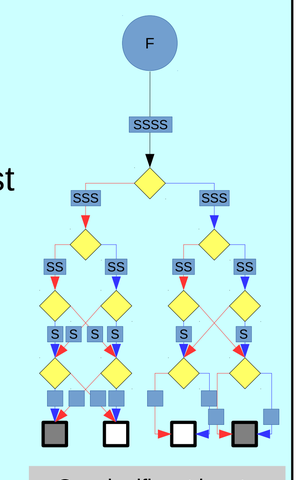
[6] Generalized Reduction of OBDD (GroBdd)

Labeling edges with transformation descriptor

A GroBdd differs from a ROBDDs in several aspects:

1) Every edge is labeled by a transformation (represented using a polynomial size transformation descriptor). A transformation can remove variable(s).
2) Case disjunction is always performed on the first variable.

Transformations must respect some properties (e.g, composability) in order to ensure canonicity.



S = significant inputs

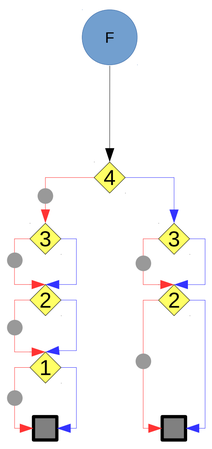
[9] Results

Average reduction of the {number of nodes / **estimated** memory cost} on four benchmarks.
Results are given relatively to the model N: ROBDD + « output negation »

	Circuits (dense functions)				CNF formulas (sparse functions)			
	lgsynth91		iscas99		uf20-91		uf50-218	
model	#node	mem	#node	mem	#node	mem	#node	mem
Z	+234%	+234%	+162%	+162%	-41%	-41%	-42%	-42%
NU	-26%	-21%	-25%	-20%	-3%	+7%	-3%	+22%
NNI	-59%	-53%	-56%	-49%	-30%	-10%	-39%	+5%
NU-X	-64%	-58%	-55%	-46%	-96%	-95%	-97%	-96%

[4] ROBDD + "output negation" => model N

Remove useless nodes



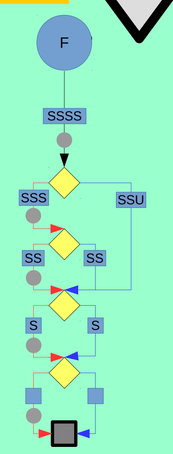
= Not

Therefore, several variants have been invented, such as: "output negation" (which allows to complement a function in constant time) and ZBDD (Zero supressed BDD, better suited for sparse functions, cf. [5]).

[7] « output Negation » (N) + « Useless inputs extraction » (U) => model NU

Extracting useless inputs

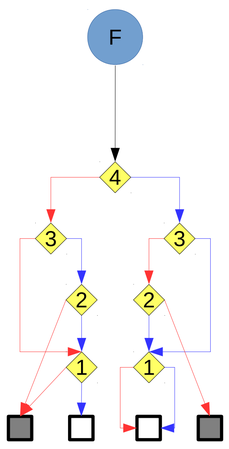
An input (or variable) is useless iff it does not influence the result of the function, e.g. x2 in (x1, x2, x3) → x1 /\ x3



U = useless inputs

[5] Zero supressed Binary Decision Diagram => model Z

Remove "if 1 then 0" nodes

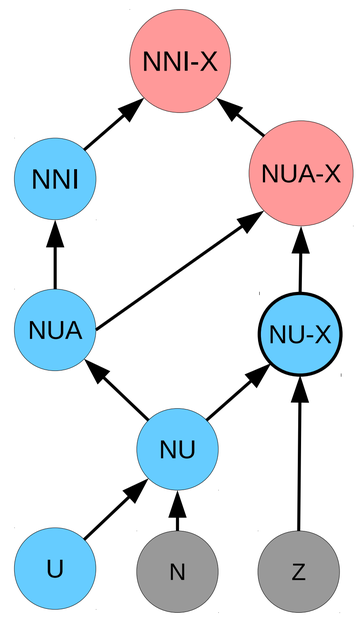


[10] Generalization Graph

"input/output negation" + "1-prediction extraction"

"inputs/output negation"

"output negation" + "useless inputs extraction" + "anti-variables extraction"



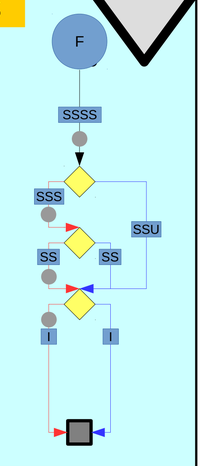
"output negation" + "useless inputs extraction" + "anti-variables extraction" + "1-prediction extraction"

"output negation" + "useless inputs extraction" + "1-prediction extraction"

= Future work
= New model
= State-Of-The-Art model

[8] model NU + « 1-prediction extraction » (X) => model NU-X

Extracting 1-predictions



1 = If 0 then 0
0 = If 1 then 0
I = If 0 then 1
O = If 1 then 1

However, ZBDD and "output negation" are not compatible, i.e. one cannot merge the reduction rules of ZBDD and "output negation" with a constant memory overhead and maintaining canonicity.
Our main contribution is to provide an abstraction of reduction rules, which allows to efficiently determine if two variants are compatible and simplifies the task of proving that a variant is canonical.
This abstraction layer was used to implement 5 new models.

Let f be a Boolean function, we say that f admits a 1-prediction (i, ifx, thy), (with i being a integer and thx/thy being Booleans),

Iff: $\forall (X_{1,}\ldots,X_{n-1}) \in B^n, f(X_{1,}\ldots,X_{i-1},ifx,X_i,..,X_{n-1}) = thy$

1-predictions are a generalization of ZBDD's reduction rule and (in average) allow a more compact representation (cf. [9]) while being compatible with « output negation ».