

Reduced Ordered Binary Decision Diagrams

Lecture #13 of Advanced Model Checking

Joost-Pieter Katoen

Lehrstuhl 2: Software Modeling & Verification

E-mail: `katoen@cs.rwth-aachen.de`

December 13, 2010

Switching functions

- Let $Var = \{z_1, \dots, z_m\}$ be a finite set of Boolean variables
- An **evaluation** is a function $\eta : Var \rightarrow \{0, 1\}$
 - let $Eval(z_1, \dots, z_m)$ denote the set of evaluations for z_1, \dots, z_m
 - shorthand $[z_1 = b_1, \dots, z_m = b_m]$ for $\eta(z_1) = b_1, \dots, \eta(z_m) = b_m$
- $f : Eval(Var) \rightarrow \{0, 1\}$ is a **switching function** for $Var = \{z_1, \dots, z_m\}$
- Logical operations and quantification are defined by:

$$\begin{aligned} f_1(\cdot) \wedge f_2(\cdot) &= \min\{f_1(\cdot), f_2(\cdot)\} \\ f_1(\cdot) \vee f_2(\cdot) &= \max\{f_1(\cdot), f_2(\cdot)\} \\ \exists z. f(\cdot) &= f(\cdot)|_{z=0} \vee f(\cdot)|_{z=1}, \text{ and} \\ \forall z. f(\cdot) &= f(\cdot)|_{z=0} \wedge f(\cdot)|_{z=1} \end{aligned}$$

Ordered Binary Decision Diagram

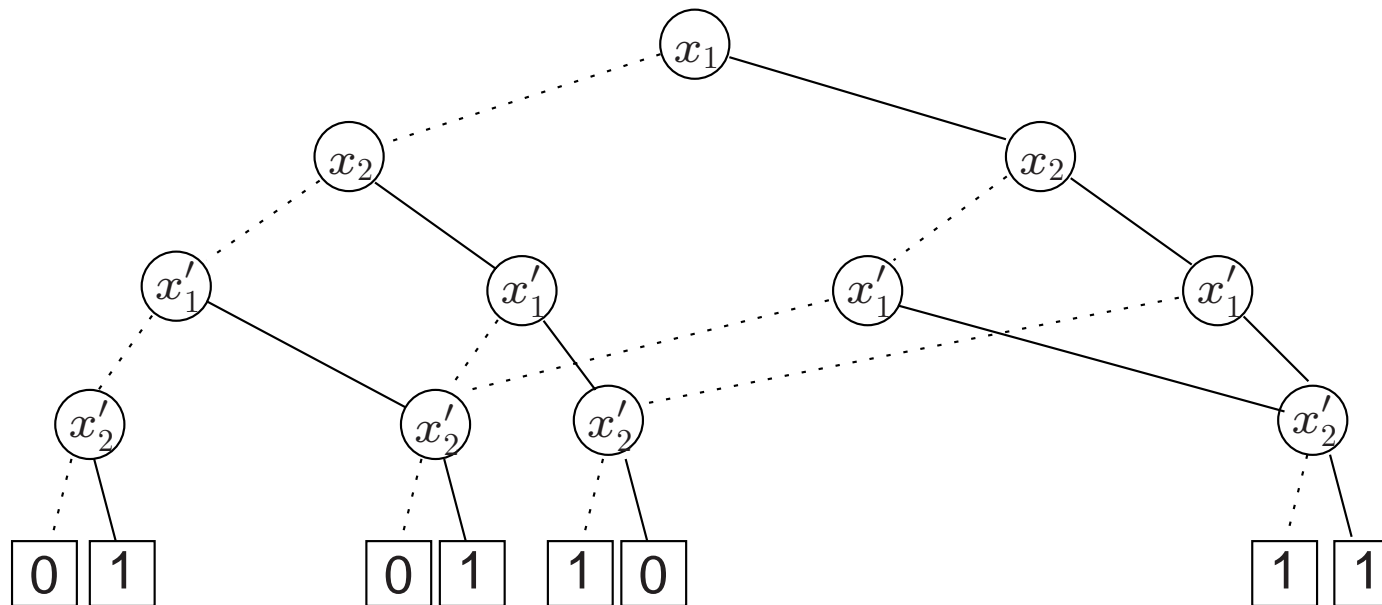
Let \wp be a **variable ordering** for Var where $z_1 <_{\wp} \dots <_{\wp} z_m$

An \wp -OBDD is a tuple $\mathfrak{B} = (V, V_I, V_T, succ_0, succ_1, var, val, v_0)$ with

- a finite set V of nodes, partitioned into V_I (**inner**) and V_T (**terminals**)
 - and a distinguished **root** $v_0 \in V$
- **successor functions** $succ_0, succ_1 : V_I \rightarrow V$
 - such that each node $v \in V \setminus \{v_0\}$ has at least one predecessor
- **labeling functions** $var : V_I \rightarrow Var$ and $val : V_T \rightarrow \{0, 1\}$ satisfying

$$v \in V_I \wedge w \in \{succ_0(v), succ_1(v)\} \cap V_I \Rightarrow var(v) <_{\wp} var(w)$$

Transition relation as an OBDD



An example OBDD representing f_{\rightarrow} for our example using $x_1 < x_2 < x'_1 < x'_2$

Symbolic composition operators

Consistent co-factors in OBDDs

- Let f be a switching function for Var
- Let $\wp = (z_1, \dots, z_m)$ a variable ordering for Var , i.e., $z_1 <_{\wp} \dots <_{\wp} z_m$
- Switching function g is a *\wp -consistent cofactor* of f if

$$g = f|_{z_1=b_1, \dots, z_i=b_i} \quad \text{for some } i \in \{0, 1, \dots, m\}$$

- Then it holds that:
 1. for each node v of an \wp -OBDD \mathfrak{B} , f_v is a \wp -consistent cofactor of $f_{\mathfrak{B}}$
 2. for each \wp -consistent cofactor g of $f_{\mathfrak{B}}$ there is a node $v \in \mathfrak{B}$ with $f_v = g$

Reduced OBDDs

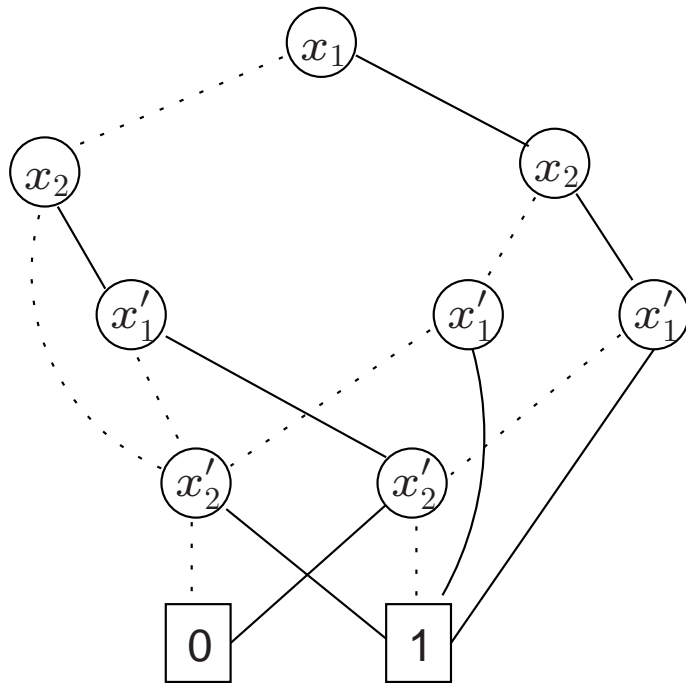
A \wp -OBDD \mathfrak{B} is *reduced* if for every pair (v, w) of nodes in \mathfrak{B} :

$$v \neq w \text{ implies } f_v \neq f_w$$

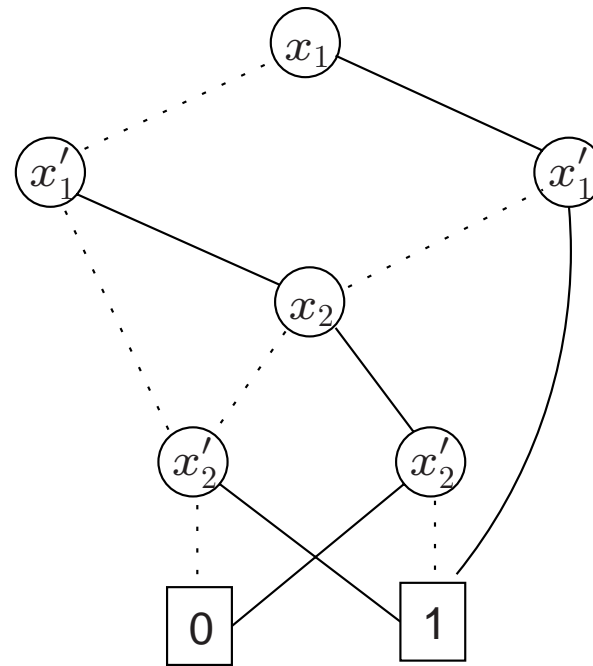
(A *reduced* \wp -OBDD is abbreviated as \wp -ROBDD)

\Rightarrow \wp -ROBDDs any \wp -consistent cofactor is represented by *exactly one node*

Transition relation as an ROBDD



(a) ordering $x_1 < x_2 < x'_1 < x'_2$



(b) ordering $x_1 <' x'_1 <' x_2 <' x'_2$

Universality and canonicity theorem

[Fortune, Hopcroft & Schmidt, 1978]

Let Var be a finite set of Boolean variables and \wp a variable ordering for Var . Then:

- (a) For each switching function f for Var there **exists** a \wp -ROBDD \mathfrak{B} with $f_{\mathfrak{B}} = f$
- (b) Any \wp -ROBDDs \mathfrak{B} and \mathfrak{C} with $f_{\mathfrak{B}} = f_{\mathfrak{C}}$ are **isomorphic**

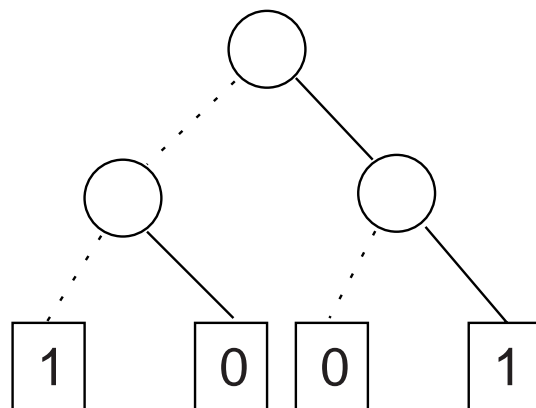
Any \wp -OBDD \mathfrak{B} for f is reduced iff $size(\mathfrak{B}) \leq size(\mathfrak{C})$ for each \wp -OBDD \mathfrak{C} for f

Reducing OBDDs

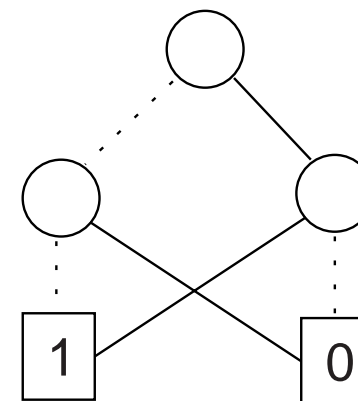
- Generate an OBDD (or BDT) for a switching function, then **reduce**
 - by means of a recursive descent over the OBDD
- **Elimination of duplicate leafs**
 - for a duplicate 0-leaf (or 1-leaf), redirect all incoming edges to just one of them
- **Elimination of “don’t care” (non-leaf) vertices**
 - if $\text{succ}_0(v) = \text{succ}_1(v) = w$, delete v and redirect all its incoming edges to w
- **Elimination of isomorphic subtrees**
 - if $v \neq w$ are roots of isomorphic subtrees, remove w and redirect all incoming edges to w to v

note that the first reduction is a special case of the latter

How to reduce an OBDD?

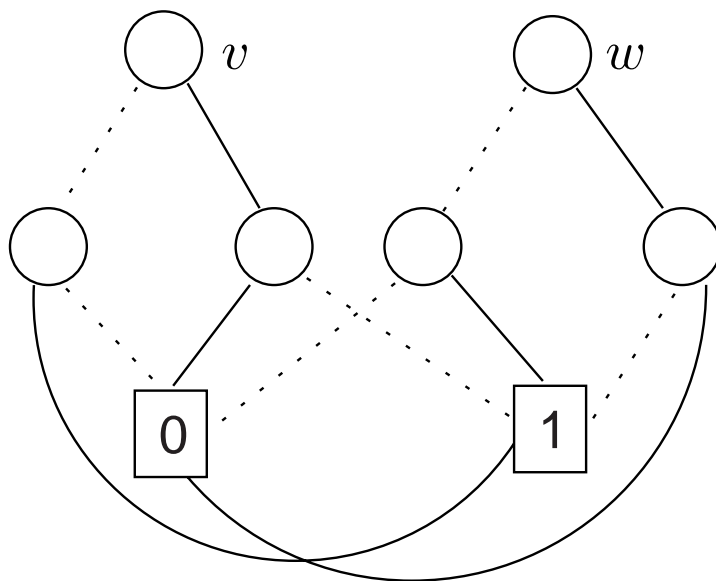


becomes

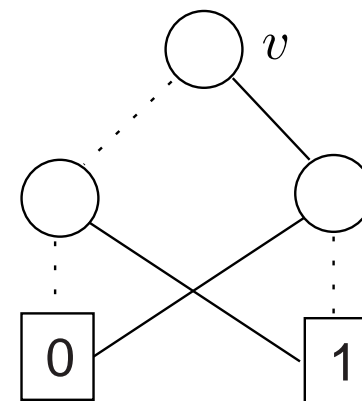


elimination of duplicated leaves

How to reduce an OBDD?

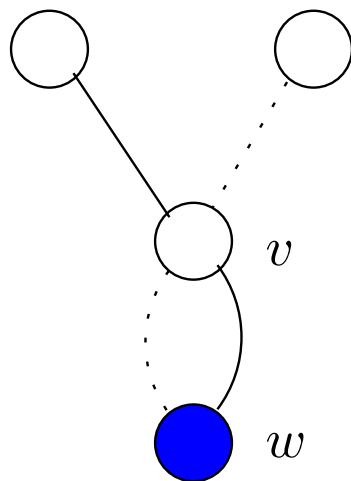


becomes

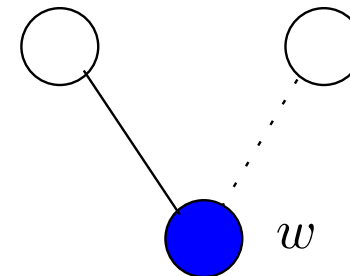


isomorphism rule

How to reduce an OBDD?



becomes



elimination rule

Soundness and completeness

if \mathcal{C} arises from a \wp -OBDD \mathfrak{B} by applying
the elimination or isomorphism rule, then:

\mathcal{C} is a \wp -OBDD with $f_{\mathfrak{B}} = f_{\mathcal{C}}$

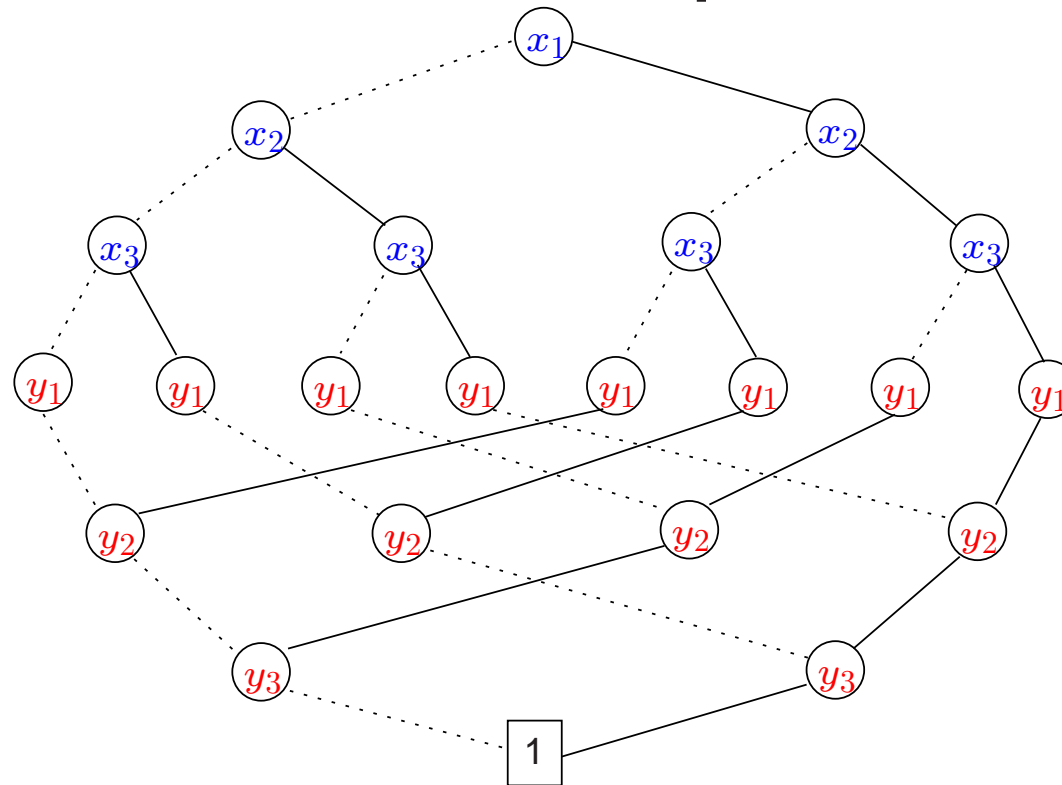
\wp -OBDD \mathfrak{B} is reduced if and only if
no reduction rule is applicable to \mathfrak{B}

Proof

Variable ordering

- ROBDDs are canonical for a **fixed** variable ordering
 - the size of the ROBDD crucially depends on the variable ordering
 - $\#$ nodes in ROBDD $\mathfrak{B} = \#$ of \wp -consistent co-factors of f
- Some switching functions have **linear and exponential** ROBDDs
 - e.g., the addition function, or the stable function
- Some switching functions only have **polynomial** ROBDDs
 - this holds, e.g., for symmetric functions (see next)
 - examples $f(\dots) = x_1 \oplus \dots \oplus x_n$, or $f(\dots) = 1$ iff $\geq k$ variables x_i are true
- Some switching functions only have **exponential** ROBDDs
 - this holds, e.g., for the middle bit of the multiplication function

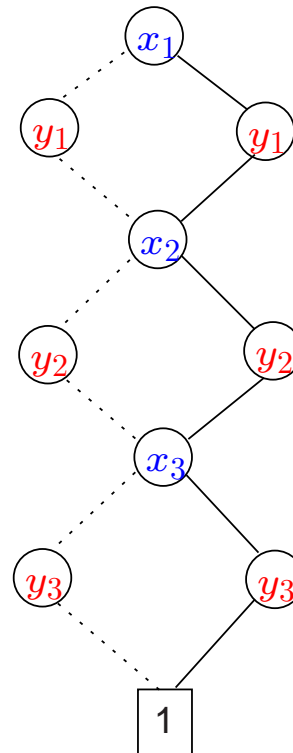
The function stable with exponential ROBDD



The ROBDD of $f_{stab}(\overline{x}, \overline{y}) = (x_1 \leftrightarrow y_1) \wedge \dots \wedge (x_n \leftrightarrow y_n)$

has $3 \cdot 2^n - 1$ vertices under ordering $x_1 < \dots < x_n < y_1 < \dots < y_n$

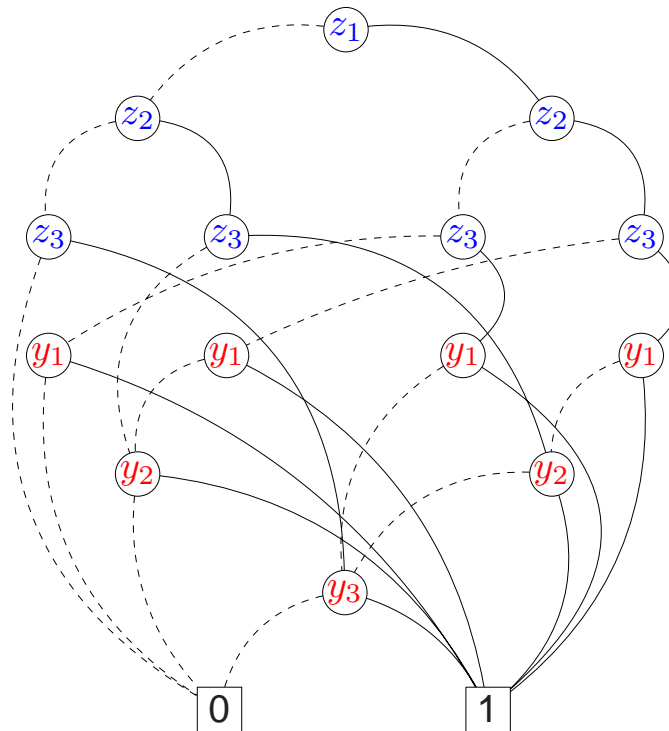
The function stable with linear ROBDD



The ROBDD of $f_{stab}(\overline{x}, \overline{y}) = (x_1 \leftrightarrow y_1) \wedge \dots \wedge (x_n \leftrightarrow y_n)$

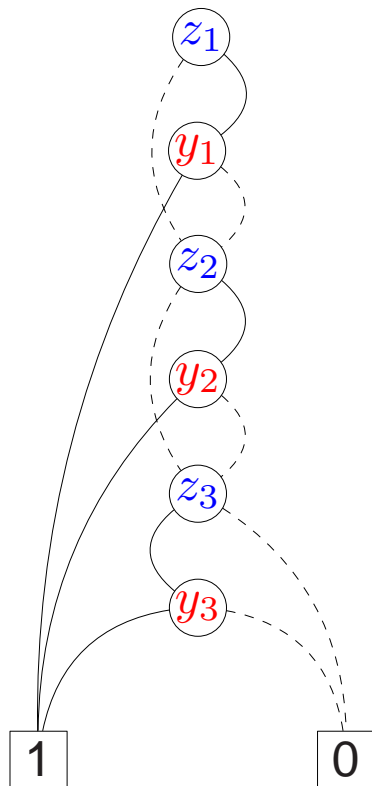
has $3 \cdot n + 2$ vertices under ordering $x_1 < y_1 < \dots < x_n < y_n$

Another function with an exponential ROBDD



ROBDD for $f_3(\bar{z}, \bar{y}) = (z_1 \wedge y_1) \vee (z_2 \wedge y_2) \vee (z_3 \wedge y_3)$
 for the variable ordering $z_1 < z_2 < z_3 < y_1 < y_2 < y_3$

And an optimal linear ROBDD



- ROBDD for $f_3(\cdot) = (z_1 \wedge y_1) \vee (z_2 \wedge y_2) \vee (z_3 \wedge y_3)$
- for ordering $z_1 < y_1 < z_2 < y_2 < z_3 < y_3$
- as all variables are essential for f , this ROBDD is **optimal**
- that is, for no variable ordering a smaller ROBDD exists

Symmetric functions

$f \in Eval(z_1, \dots, z_m)$ is **symmetric** if and only if

$$f([z_1 = b_1, \dots, z_m = b_m]) = f([z_1 = b_{i_1}, \dots, z_m = b_{i_m}])$$

for each permutation (i_1, \dots, i_m) of $(1, \dots, m)$

E.g.: $z_1 \vee z_2 \vee \dots \vee z_m$, $z_1 \wedge z_2 \wedge \dots \wedge z_m$, the parity function, and the majority function

If f is a symmetric function with m essential variables, then
for each variable ordering \wp the \wp -ROBDD has size $\mathcal{O}(m^2)$

The even parity function

$f_{\text{even}}(x_1, \dots, x_n) = 1$ iff the number of variables x_i with value 1 is even

truth table or propositional formula for f_{even} has exponential size

but an ROBDD of linear size is possible

The multiplication function

- Consider two n -bit integers
 - let $b_{n-1}b_{n-2} \dots b_0$ and $c_{n-1}c_{n-2} \dots c_0$
 - where b_{n-1} is the most significant bit, and b_0 the least significant bit
- Multiplication yields a $2n$ -bit integer
 - the ROBDD $\mathfrak{B}_{f_{n-1}}$ has at least 1.09^n vertices
 - where f_{n-1} denotes the $(n-1)$ -st output bit of the multiplication

Optimal variable ordering

- The size of ROBDDs is dependent on the variable ordering
- Is it possible to determine \wp such that the ROBDD has minimal size?
 - to check whether a variable ordering is optimal is NP-hard
 - polynomial reduction from the 3SAT problem [Bollig & Wegener, 1996]
- There are many switching functions with large ROBDDs
 - for almost all switching functions the minimal size is in $\Omega(\frac{2^n}{n})$
- How to deal with this problem in practice?
 - guess a variable ordering in advance
 - rearrange the variable ordering during the ROBDD manipulations
 - not necessary to test all $n!$ orderings, best known algorithm in $\mathcal{O}(3^n \cdot n^2)$

Variable swapping

Sifting algorithm

[Rudell, 1993]

Dynamic variable ordering using variable swapping:

1. Select a variable x_i in OBDD at hand
 2. By successive swapping of x_i , determine $size(\mathfrak{B})$ at any position for x_i
 3. Shift x_i to position for which $size(\mathfrak{B})$ is minimal
 4. Go back to the first step until no improvement is made
- Characteristics:
 - a variable may change position several times during a single sifting iteration
 - often yields a local optimum, but works well in practice

Interleaved variable ordering

- Which variable ordering to use for transition relations?
- The **interleaved** variable ordering:
 - for encodings x_1, \dots, x_n and y_1, \dots, y_n of state s and t respectively:

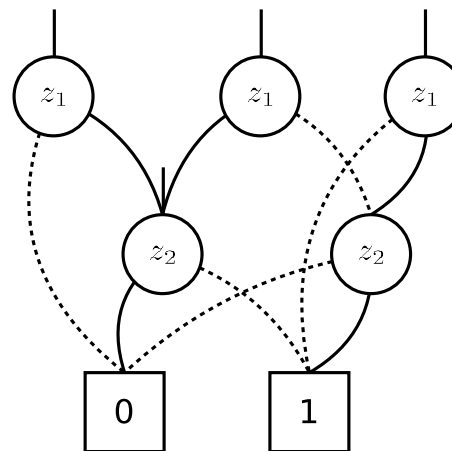
$$x_1 < y_1 < x_2 < y_2 < \dots < x_n < y_n$$

- This variable ordering yields compact ROBDDs for binary relations
 - for transition relation with $z_1 \dots z_m$ be the encoding of action α , take:

$$\underbrace{z_1 < z_2 < \dots < z_m}_{\text{encoding of } \alpha} < \underbrace{x_1 < y_1 < x_2 < y_2 < \dots < x_n < y_n}_{\text{interleaved order of states}}$$

Implementation: shared OBDDs

A **shared** \wp -OBDD is an OBDD with **multiple** roots



Shared OBDD representing $\underbrace{z_1 \wedge \neg z_2}_{f_1}$, $\underbrace{\neg z_2}_{f_2}$, $\underbrace{z_1 \oplus z_2}_{f_3}$ and $\underbrace{\neg z_1 \vee z_2}_{f_4}$

Main underlying idea: combine several OBDDs with same variable ordering
such that common \wp -consistent co-factors are shared

Synthesizing shared ROBDDs

Relies on the use of two tables

- The **unique** table
 - keeps track of ROBDD nodes that already have been created
 - table entry $\langle \text{var}(v), \text{succ}_1(v), \text{succ}_0(v) \rangle$ for each inner node v
 - main operation: *find_or_add*(z, v_1, v_0) with $v_1 \neq v_0$
 - * return v if there exists a node $v = \langle z, v_1, v_0 \rangle$ in the ROBDD
 - * if not, create a new z -node v with $\text{succ}_0(v) = v_0$ and $\text{succ}_1(v) = v_1$
 - implemented using hash functions (expected access time is $\mathcal{O}(1)$)
- The **computed** table
 - keeps track of tuples for which ITE has been executed (memoization) \Rightarrow realizes a kind of dynamic programming

ITE normal form

The **ITE** (if-then-else) operator: $ITE(g, f_1, f_2) = (g \wedge f_1) \vee (\neg g \wedge f_2)$

The ITE operator and the representation of the SOBDD nodes in the unique table:

$$f_v = ITE\left(z, f_{succ_1(v)}, f_{succ_0(v)}\right)$$

Then:

$$\begin{aligned}\neg f &= ITE(f, 0, 1) \\ f_1 \vee f_2 &= ITE(f_1, 1, f_2) \\ f_1 \wedge f_2 &= ITE(f_1, f_2, 0) \\ f_1 \oplus f_2 &= ITE(f_1, \neg f_2, f_2) = ITE(f_1, ITE(f_2, 0, 1), f_2)\end{aligned}$$

If g, f_1, f_2 are switching functions for Var , $z \in Var$ and $b \in \{0, 1\}$, then

$$ITE(g, f_1, f_2)|_{z=b} = ITE(g|_{z=b}, f_1|_{z=b}, f_2|_{z=b})$$

ITE-operator on shared OBDDs

- A node in a \wp -SOBDD for representing $ITE(g, f_1, f_2)$ is a node w with $info\langle z, w_1, w_0 \rangle$ where:
 - z is the minimal (wrt. \wp) essential variable of $ITE(g, f_1, f_2)$
 - w_b is an SOBDD-node with $f_{w_b} = ITE(g|_{z=b}, f_1|_{z=b}, f_2|_{z=b})$
- This suggests a recursive algorithm:
 - determine z
 - recursively compute the nodes for ITE for the cofactors of g, f_1 and f_2

$ITE(u, v_1, v_2)$ on shared OBDDs (initial version)

if u is terminal **then**

if $val(u) = 1$ **then**

$w := v_1$

else

$w := v_2$

fi

else

$z := \min\{var(u), var(v_1), var(v_2)\};$

$w_1 := ITE(u|_{z=1}, v_1|_{z=1}, v_2|_{z=1});$

$w_0 := ITE(u|_{z=0}, v_1|_{z=0}, v_2|_{z=0});$

if $w_0 = w_1$ **then**

$w := w_1;$

else

$w := find_or_add(z, w_1, w_0);$

fi

fi

return w

(* $ITE(1, f_{v_1}, f_{v_2}) = f_{v_1}$ *)

(* $ITE(0, f_{v_1}, f_{v_2}) = f_{v_2}$ *)

(* minimal essential variable *)

(* elimination rule *)

(* isomorphism rule *)

ROBDD size under ITE

The size of the \wp -ROBDD for $ITE(g, f_1, f_2)$ is bounded by $N_g \cdot N_{f_1} \cdot N_{f_2}$
where N_f denotes the size of the \wp -ROBDD for f

ROBDD size under ITE

The size of the \wp -ROBDD for $ITE(g, f_1, f_2)$ is bounded by $N_g \cdot N_{f_1} \cdot N_{f_2}$
where N_f denotes the size of the \wp -ROBDD for f

But how to avoid multiple invocations to ITE?

\Rightarrow Store triples (u, v_1, v_2) for which ITE already has been computed

Efficiency improvement by memoization

```

if there is an entry for  $(u, v_1, v_2, w)$  in the computed table then
  return node  $w$ 
else
  if  $u$  is terminal then
    if  $val(u) = 1$  then  $w := v_1$  else  $w := v_2$  fi
  else
     $z := \min\{var(u), var(v_1), var(v_2)\};$ 
     $w_1 := ITE(u|_{z=1}, v_1|_{z=1}, v_2|_{z=1});$ 
     $w_0 := ITE(u|_{z=0}, v_1|_{z=0}, v_2|_{z=0});$ 
    if  $w_0 = w_1$  then  $w := w_1$  else  $w := find\_or\_add(z, w_1, w_0)$  fi;
    insert  $(u, v_1, v_2, w)$  in the computed table;
    return node  $w$ 
  fi
fi

```

The number of recursive calls for the nodes u, v_1, v_2 equals the \wp -ROBDD size of $ITE(f_u, f_{v_1}, f_{v_2})$, which is bounded by $N_u \cdot N_{v_1} \cdot N_{v_2}$