Verification – Lecture 18
Symbolic Model Checking

Bernd Finkbeiner – Sven Schewe
Rayna Dimitrova – Lars Kuhtz – Anne Proetzsch

Wintersemester 2007/2008

Summary of CTL model checking (1)

- CTL is a logic for formalizing properties over computation trees
- The expressiveness of LTL and CTL is incomparable
- Fairness constraints cannot be expressed in CTL
  - but are incorporated by adapting the CTL semantics such that quantification is over fair paths
- CTL model checking is by a recursive descent over parse tree of Φ
  - $Sat(\exists(\Phi \, \mathsf{U} \, \Psi))$ is determined using a least fixed point computation
  - $Sat(\exists\Box\Phi)$ is determined by a greatest fixed point computation

Summary of CTL model checking (2)

- Time complexity of CTL model-checking $S \models \Phi$ is:
  - is linear in $|S|$ and $|\Phi|$ and linear in k for k fairness constraints
- Checking $S \models_{fair} \Phi$ is $S \models \Phi$ plus computing $Sat_{fair}(\exists \Box a)$
- CTL* is more expressive than both CTL and LTL
- The CTL* model-checking problem can be solved by an appropriate combination of the CTL and the LTL model-checking algorithm
- The CTL*-model checking problem is PSPACE-complete

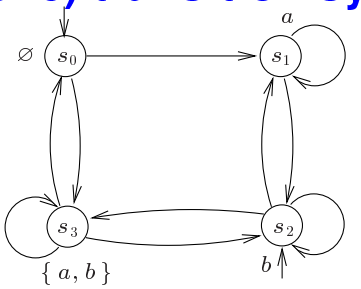
Review: Transition Systems

- finite set of variables: V
- initial condition θ: assertion over V
- finite set of transitions $\mathcal{T}$
  each $\tau \in \mathcal{T}$ represented by transition relation ρ_τ over $V \cup V'$
  - V: values in present state
  - V': values in next state
- Atomic propositions AP: assertions over V

Boolean Transition Systems

- finite set of boolean variables: V
- initial condition θ: boolean function over V
- transitions represented by transition relation: boolean function ρ over $V \cup V'$
  - V: values in present state
  - V': values in next state
- Atomic propositions $AP = V$.

(Explicit) state graphs vs. (symbolic) transition systems



- States:

state	*bit-vector*	*boolean function*
s_0	$\langle 0, 0 \rangle$	$\neg\, x_1 \wedge \neg\, x_2$
s_1	$\langle 0, 1 \rangle$	$\neg\, x_1 \wedge x_2$
s_2	$\langle 1, 0 \rangle$	$x_1 \wedge \neg\, x_2$
s_3	$\langle 1, 1 \rangle$	$x_1 \wedge x_2$

- Initial states: $\theta(x_1, x_2) = (\neg\, x_1 \wedge \neg\, x_2) \ \vee \ (x_1 \wedge \neg\, x_2)$

explicit vs. symbolic (cont'd)

- Edge relation:

E	$\langle 0,0\rangle$	$\langle 0,1\rangle$	$\langle 1,0\rangle$	$\langle 1,1\rangle$
$\langle 0,0\rangle$	0	1	0	1
$\langle 0,1\rangle$	0	1	1	0
$\langle 1,0\rangle$	0	1	1	1
$\langle 1,1\rangle$	1	0	1	1

- Alternatively: $\rho(\underbrace{x_1, x_2}_{q}, \underbrace{x_1', x_2'}_{q'}) = 1$ if and only if $(q, q') \in E$

$$\begin{array}{rll} \rho(x_1, x_2, x_1', x_2') = & & (\neg\, x_1 \ \wedge \ \neg\, x_2 \ \wedge \ \neg\, x_1' \ \wedge \ x_2') \\ & \vee & (\neg\, x_1 \ \wedge \ \neg\, x_2 \ \wedge \ x_1' \ \wedge \ x_2') \\ & \vee & (\neg\, x_1 \ \wedge \ x_2 \ \wedge \ x_1' \ \wedge \ \neg\, x_2') \\ & \vee & \ldots \\ & \vee & (x_1 \ \wedge \ x_2 \ \wedge \ x_1' \ \wedge \ x_2') \end{array}$$

Boolean functions

- Boolean functions $f : \mathbb{B}^n \to \mathbb{B}$ for $n \geqslant 0$ where $\mathbb{B} = \{0, 1\}$
  - examples: $f(x_1, x_2) = x_1 \wedge (x_2 \ \vee \ \neg x_1)$, and $f(x_1, x_2) = x_1 \leftrightarrow x_2$
- Finite sets are boolean functions
  - let $|Q| = N$ and $2^{n-1} < N \leqslant 2^n$
  - each state $q \in Q$ is a boolean vector of length n: $[\![\ ]\!] : Q \to \mathbb{B}^n$
  - $T \subseteq Q$ is represented by f_T such that:

$$f_T([\![\, q \,]\!]) = 1 \quad \text{iff} \quad q \in T$$

  - this is the characteristic function of T
- Relations are boolean functions
  - $\mathcal{R} \subseteq Q \times Q$ is represented by $f_{\mathcal{R}}$ such that:

$$f_R([\![\, s \,]\!], [\![\, t \,]\!]) = 1 \quad \text{iff} \quad (s, t) \in \mathcal{R}$$

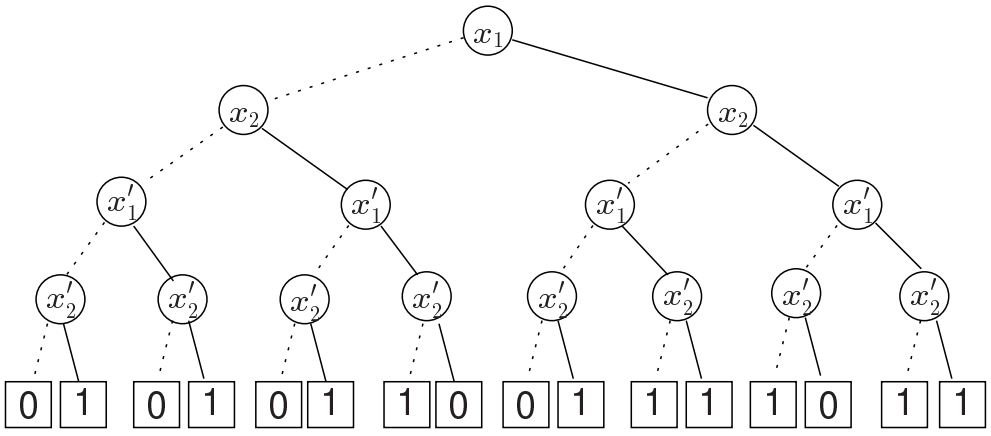
Binary decision trees

- Let X be a set of boolean variables and $<$ a total order on X
- Binary decision tree (BDT) is a complete binary tree over $\langle X, < \rangle$
  - each leaf v is labeled with a boolean value $\mathit{val}(v) \in \mathbb{B}$
  - non-leaf v is labeled by a boolean variable $\mathit{Var}(v) \in X$
  - such that for each non-leaf v and vertex w:

$$w \in \{\, \mathit{left}(v), \mathit{right}(v) \,\} \;\Rightarrow\; (\mathit{Var}(v) < \mathit{Var}(w) \;\vee\; w \text{ is a leaf})$$

$\Rightarrow$ On each path from root to leaf, variables occur in the same order

Transition relation as a BDT



A BDT representing ρ for our example using $x_1 < x_2 < x_1' < x_2'$

Shannon expansion

- Each boolean function $f : \mathbb{B}^n \longrightarrow \mathbb{B}$ can be written as:

$$f(x_1, \ldots, x_n) = (x_i \ \wedge \ f[x_i := 1]) \ \vee \ (\neg\, x_i \ \wedge \ f[x_i := 0])$$

  - where $f[x_i := 1]$ stands for $f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$
  - and $f[x_i := 0]$ is a shorthand for $f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$

- The boolean function $f_B(v)$ represented by vertex v in BDT B is:
  - for v a leaf: $f_B(v) = \mathit{val}(v)$
  - otherwise:

$$f_B(v) = (\mathit{Var}(v) \ \wedge \ f_B(\mathit{right}(v))) \ \vee \ (\neg\, \mathit{Var}(v) \ \wedge \ f_B(\mathit{left}(v)))$$

- $f_B = f_B(v)$ where v is the root of B

Considerations on BDTs

- BDTs are not compact
  - a BDT for boolean function $f : \mathbb{B}^b \to \mathbb{B}$ has 2^n leafs

  $\Rightarrow$ they are as space inefficient as truth tables!

$\Rightarrow$ BDTs contain quite some redundancy

  - all leafs with value one (zero) could be collapsed into a single leaf
  - a similar scheme could be adopted for isomorphic subtrees

- The size of a BDT does not change if the variable order changes

Ordered Binary Decision Diagram

share equivalent expressions [Akers 76, Lee 59]

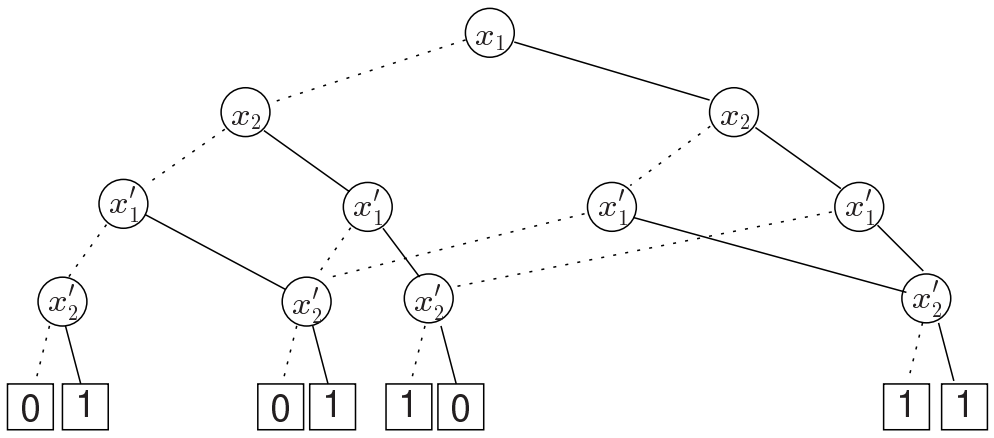
- Binary decision diagram (OBDD) is a directed graph over $\langle X, < \rangle$ with:
  - each leaf v is labeled with a boolean value *val*$(v) \in \{ 0, 1 \}$
  - non-leaf v is labeled by a boolean variable *Var*$(v) \in X$
  - such that for each non-leaf v and vertex w:

$$w \in \{ \mathit{left}(v), \mathit{right}(v) \} \;\Rightarrow\; (\mathit{Var}(v) < \mathit{Var}(w) \;\vee\; w \text{ is a leaf})$$

$\Rightarrow$ An OBDD is acyclic

- f_B for OBDD B is obtained as for BDTs

Transition relation as an OBDD



An example OBDD representing ρ for our example using $x_1 < x_2 < x_1' < x_2'$

Isomorphism

- B and B′ over $\langle X, < \rangle$ are *isomorphic* iff their roots are isomorphic
- Vertices v in B and w in B′ are isomorphic, denoted $v \cong w$, iff there exists a bijection H between the vertices of B and B′ such that:
  1. if v is a leaf, then $H(v) = w$ is a leaf with $\mathit{val}(v) = \mathit{val}(H(v))$
  2. if v is a non-leaf, then $H(v) = w$ is a non-leaf such that

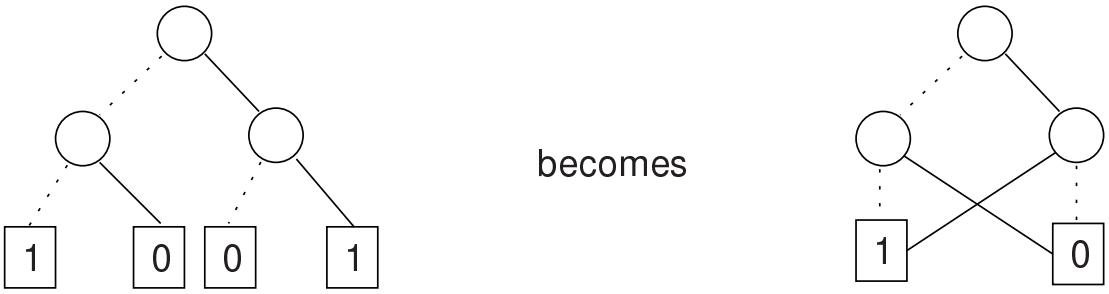
$$\mathit{Var}(v) = \mathit{Var}(w) \;\wedge\; H(\mathit{left}(v)) = \mathit{left}(H(v)) \;\wedge\; H(\mathit{right}(v)) = \mathit{right}(H(v))$$

- Testing B $\cong$ B′ can be done in linear time
  - due to the labels (0 and 1) of the edges.

Reducing OBDDs

- Generate an OBDD (or BDT) for a boolean expression, 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 $\mathit{left}(v) = \mathit{right}(v) = w$, eliminate 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

How to reduce an OBDD?

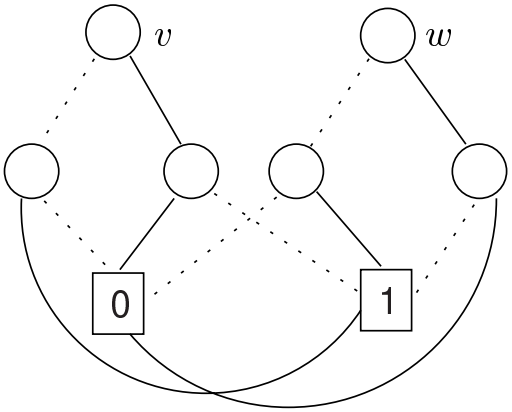


eliminating identical leafs

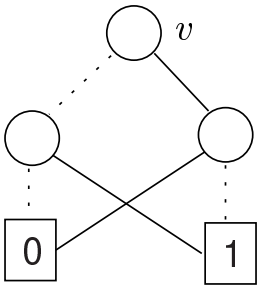
How to reduce an OBDD?

eliminating "don't care" vertices

How to reduce a BDD?



becomes



eliminating isomorphic subtrees

Reduced OBDDs

OBDD B over $\langle X, < \rangle$ is called *reduced* iff:

1. for each leaf v, w: $(\mathit{val}(v) = \mathit{val}(w)) \Rightarrow v = w$

   $\Rightarrow$ identical terminal vertices are forbidden

2. for each non-leaf v: $\mathit{left}(v) \neq \mathit{right}(v)$

   $\Rightarrow$ non-leafs may not have identical children

3. for each non-leaf v, w:

$$(\mathit{Var}(v) = \mathit{Var}(w) \ \wedge \ \mathit{right}(v) \cong \mathit{right}(w) \ \wedge \ \mathit{left}(v) \cong \mathit{left}(w)) \ \Rightarrow \ v = w$$

   $\Rightarrow$ vertices may not have isomorphic sub-dags

this is what is mostly called BDD; in fact it is an ROBDD!

Dynamic generation of ROBDDs

Main idea:

- Construct directly an ROBDD from a boolean expression
- Create vertices in depth-first search order
- On-the-fly reduction by applying hashing
  - on encountering a new vertex v, check whether:
  - an equivalent vertex w has been created (same label and children)
  - $\mathit{left}(v) = \mathit{right}(v)$, i.e., vertex v is a "don't care" vertex

ROBDDs are canonical

[Fortune, Hopcroft & Schmidt, 1978]

For ROBDDs B and B′ over $\langle X, < \rangle$ we have:

$(f_{\mathsf{B}} = f_{\mathsf{B}'})$ implies B and B′ are isomorphic

$\Rightarrow$ for a fixed variable ordering, any boolean function can be uniquely represented by an ROBDD (up to isomorphism)

The importance of canonicity

- Absence of redundant vertices
  - if f_{B} does not depend on x_i, ROBDD B does not contain an x_i vertex
- Test for equivalence: $f(x_1, \ldots, x_n) \equiv g(x_1, \ldots, x_n)$?
  - generate ROBDDs B_f and B_g, and check isomorphism
- Test for validity: $f(x_1, \ldots, x_n) = 1$?
  - generate ROBDD B_f and check whether it only consists of a 1-leaf
- Test for implication: $f(x_1, \ldots, x_n) \rightarrow g(x_1, \ldots, x_n)$?
  - generate ROBDD $\mathsf{B}_f \wedge \neg \mathsf{B}_g$ and check if it just consist of a 0-leaf
- Test for satisfiability
  - f is satisfiable if and only if B_f is not just the 0-leaf

Variable ordering

- The size of the ROBDD depends on the variable ordering
- For some functions, very compact ROBDDs may be obtained
  - e.g., the even parity function
- Some boolean functions have linear and exponential ROBDDs
  - e.g., the addition function, or the stable function
- Some boolean functions only have polynomial ROBDDs
  - this holds, e.g., for symmetric functions (see next)
  - examples $f(\ldots) = x_1 \oplus \ldots \oplus x_n$, or $f(\ldots) = 1$ iff $\geqslant k$ variables x_i are true
- Some boolean functions only have exponential ROBDDs
  - this holds, e.g., for the multiplication function, cf. (Bryant, 1986)

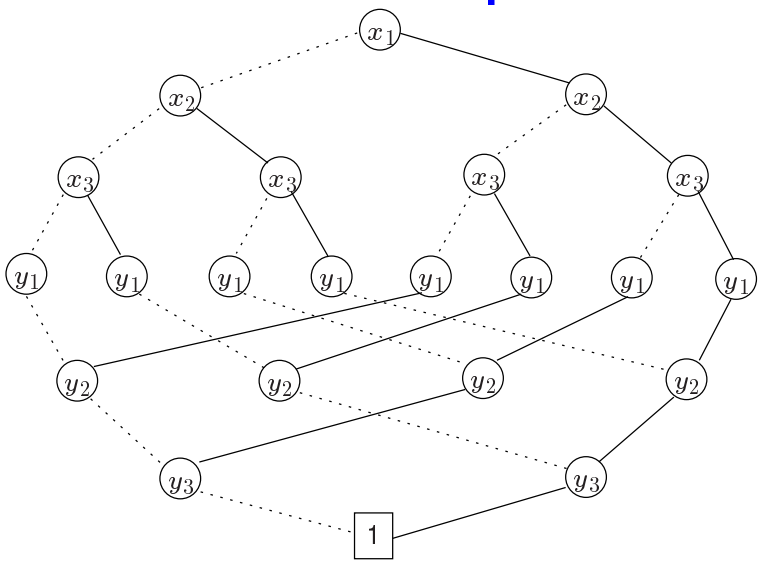
The even parity function

$f_{even}(x_1, \ldots, x_n) = 1$ 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

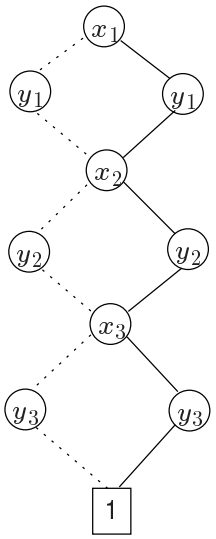
The function stable with exponential ROBDD



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

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

The function stable with linear ROBDD



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

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

Symmetric functions

$$f[x_1 := b_1, \ldots x_n := b_n] = f[x_1 := b_{i_1}, \ldots, x_{i_n} := b_{i_n}]$$

for each permutation $(i_1, \ldots, i_n)$ of $(1, \ldots, n)$

symmetric boolean functions have ROBDDs of size in $\mathcal{O}(n^2)$

The multiplication function

- Consider two n-bit integers
  - let $b_{n-1}b_{n-2}\ldots b_0$ and $c_{n-1}c_{n-2}\ldots c_0$
  - where b_{n-1} is the most significant bit, and b_0 the least significant bit
- Multiplication yields a $2n$-bit integer
  - the ROBDD $B_{f_{n-1}}$ has at least 1.09^n vertices
  - where f_{n-1} denotes the 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 $<$ such that the ROBDD has minimal size?
  - the optimal variable ordering problem for ROBDDs is NP-complete
  - polynomial reduction from the 3SAT problem (Bollig & Wegener, 1996)
- There are many boolean functions with large ROBDDs
  - for almost all boolean 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 manipulations of ROBDDs

Sifting algorithm

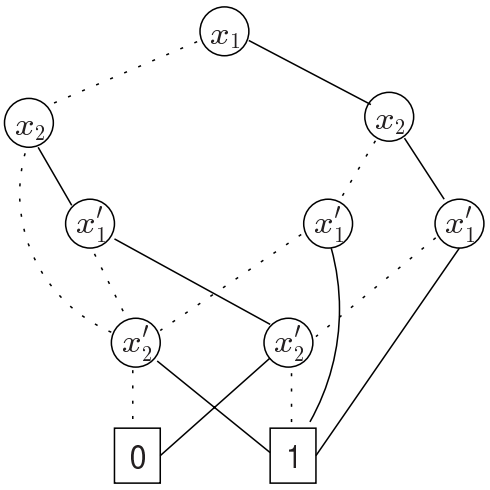
(Rudell, 1993)

Dynamic variable ordering using variable swapping:

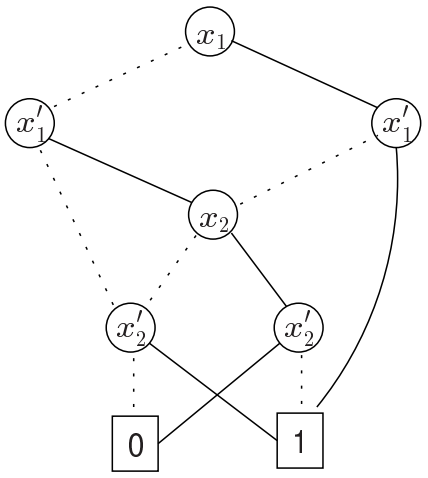
1. Select a variable x_i
2. By successive swapping of x_i, determine $|\mathrm{B}|$ at any position for x_i
3. Shift x_i to its optimal position
4. Go back to the first step until no improvement is made

- Characteristics:
  - a variable may change position several times during a single sifting iteration
  - often yields a local optimum, but works well in practice

Transition relation as an ROBDD



(a) ordering $x_1 < x_2 < x_1' < x_2'$



(b) ordering $x_1 <' x_1' <' x_2 <' x_2'$

Interleaved variable ordering

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

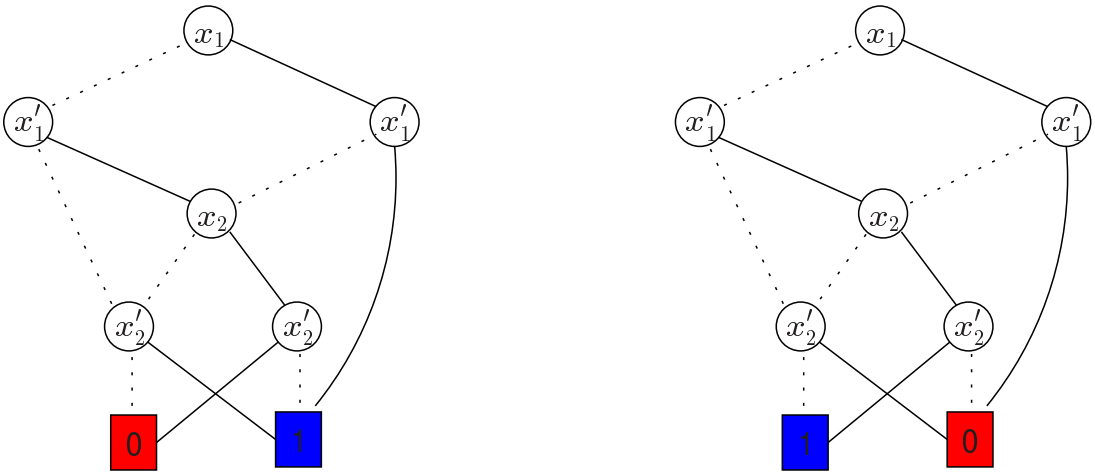
$$x_1 < y_1 < x_2 < y_2 < \ldots < x_n < y_n$$

- This variable ordering yields compact ROBDDs for binary relations

Operations on ROBDDs

Algorithm	Inputs	Output ROBDD
REDUCE	B (not reduced)	B′ (reduced) with $f_{\mathsf{B}} = f_{\mathsf{B}'}$
NOT	B_f	$\mathsf{B}_{\neg f}$
APPLY	B_f, B_g, binary logical operator *op*	$\mathsf{B}_{f\ op\ g}$
RESTRICT	B_f, variable x, boolean value b	$\mathsf{B}_{f[x:=b]}$
RENAME	B_f, variables x and y	$\mathsf{B}_{f[x:=y]}$
EXISTS	B_f, variable x	$\mathsf{B}_{\exists x.\, f}$

Negation



negation amounts to interchange the 0- and 1-leaf

APPLY

- Shannon expansion for binary operations:

$$f \ op \ g \quad = \quad (x_1 \ \wedge \ (f[x_1 := 1] \ op \ g[x_1 := 1])) \\ \vee \ (\neg\, x_1 \ \wedge \ (f[x_1 := 0] \ op \ g[x_1 := 0]))$$

- A top-down evaluation scheme using the Shannon's expansion:
  - let v be the variable highest in the ordering occurring in B_f orB_g
  - split the problem into subproblems for $v := 0$ and $v := 1$, and solve recursively
  - at the leaves, apply the boolean operator *op* directly
  - reduce afterwards to turn the resulting OBDD into an ROBDD
- Efficiency gain is obtained by *dynamic programming*
  - the time complexity of constructing the ROBDD of $B_f \ op \ B_g$ is in $\mathcal{O}(|B_f| \cdot |B_g|)$

Algorithm APPLY(op, B_f, B_g)

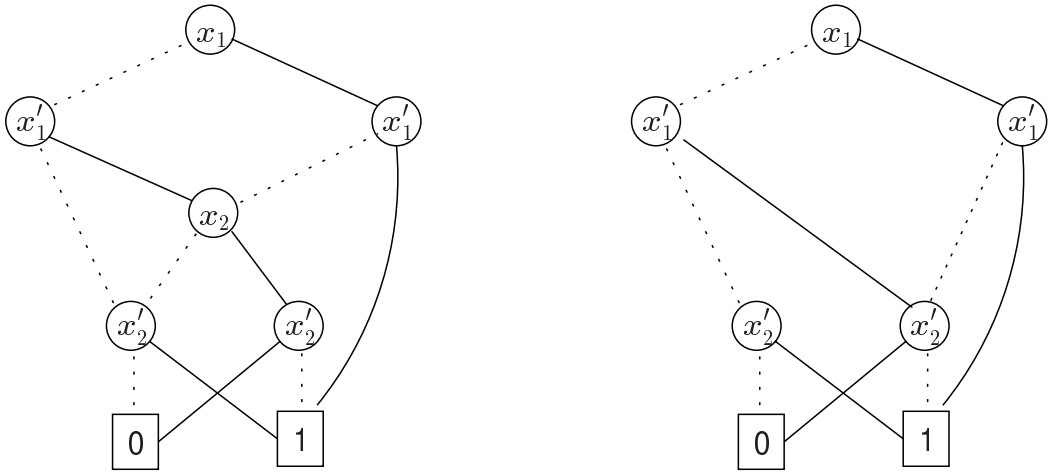
B.root := APPLY(*op*, B_f.root, B_g.root);

if $G(v_1, v_2) \neq$ empty **then return** $G(v_1, v_2)$ **fi**; (* lookup in hashtable *)
if (v_1 and v_2 are terminals) **then** *res* := *val*(v_1) *op val*(v_2) **fi**;
else if (v_1 is terminal and v_2 is nonterminal)
  then *res* := $MakeNode$(*Var*(v_2), APPLY(*op*, v_1, *left*(v_2)), APPLY(*op*, v_1, *right*(v_2)));
else if (v_1 is nonterminal and v_2 is terminal)
  then *res* := $MakeNode$(*Var*(v_1), APPLY(*op*, *left*(v_1), v_2), APPLY(*op*, *right*(v_1), v_2));
else if (*Var*(v_1) = *Var*(v_2))
  then *res* := $MakeNode$(*Var*(v_1), APPLY(*op*, *left*(v_1), *left*(v_2)), APPLY(*op*, *right*(v_1), *right*(v_2)));
else if (*Var*(v_1) < *Var*(v_2))
  then *res* := $MakeNode$(*Var*(v_1), APPLY(*op*, *left*(v_1), v_2), APPLY(*op*, *right*(v_1), v_2));
else (* *Var*(v_1) > *Var*(v_2) *)
  res := $MakeNode$(*Var*(v_2), APPLY(*op*, v_1, *left*(v_2)), APPLY(*op*, v_1, *right*(v_2)));
$G(v_1, v_2)$:= *res*; (* memoize result *)
return *res*

Algorithm RESTRICT(B, x, b)

- For each vertex v labeled with variable x:
  - if $b = 1$ then redirect incoming edges to *right*(v)
  - if $b = 0$ then redirect incoming edges to *left*(v)
  - remove vertex v, and (if necessary) reduce (only above v)

RESTRICT



performing RESTRICT(B, x_2, 1): replace x_2 by constant 1

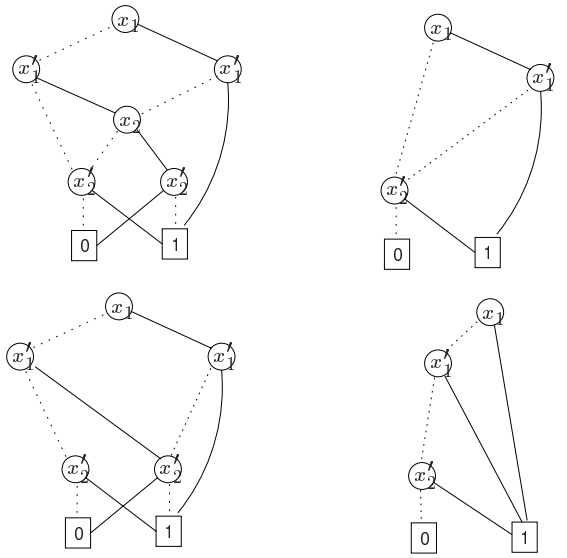
EXISTS

- Existential quantification over x_i:

$$\exists x_i.\, f(x_1, \ldots, x_n) = f[x_i := 1] \;\vee\; f[x_i := 0]$$

- Naive realization: APPLY($\vee$, RESTRICT(B$_f$, x_i, 1), RESTRICT(B$_f$, x_i, 0))
- Efficiency gain:
  - observe that RESTRICT(B$_f$, x_i, 1) and RESTRICT(B$_f$, x_i, 0) are equal up to x_i
  - . . . the resulting ROBDD also has the same structure up to x_i
  - replace each node labeled with x_i by the result of applying $\vee$ on its children
- This can easily be generalized to $\exists x_1. \ldots \exists x_k.\, f(x_1, \ldots x_n)$

A more involved example



ROBBDs B_f (left up), $\mathsf{B}_{f[x_2:=0]}$ (right up), $\mathsf{B}_{f[x_2:=1]}$ (left down), and $\mathsf{B}_{\exists x_2.\ f}$ (right down)

Operations on ROBDDs

Algorithm	Output	Time complexity	Space complexity
REDUCE	B' (reduced) with $f_{\mathsf{B}} = f_{\mathsf{B}'}$	$\mathcal{O}(\lvert B_f\rvert \cdot \log \lvert\mathsf{B}_f\rvert)$	$\mathcal{O}(\lvert\mathsf{B}_f\rvert)$
NOT	$\mathsf{B}_{\neg f}$	$\mathcal{O}(\lvert\mathsf{B}_f\rvert)$	$\mathcal{O}(\lvert\mathsf{B}_f\rvert)$
APPLY	B_f *op* g	$\mathcal{O}(\lvert\mathsf{B}_f\rvert \cdot \lvert\mathsf{B}_g\rvert)$	$\mathcal{O}(\lvert\mathsf{B}_f\rvert \cdot \lvert\mathsf{B}_g\rvert)$
RESTRICT	$\mathsf{B}_{f[x:=b]}$	$\mathcal{O}(\lvert\mathsf{B}_f\rvert)$	$\mathcal{O}(\lvert\mathsf{B}_f\rvert)$
RENAME	$\mathsf{B}_{f[x:=y]}$	$\mathcal{O}(\lvert\mathsf{B}_f\rvert)$	$\mathcal{O}(\lvert\mathsf{B}_f\rvert)$
EXISTS	$\mathsf{B}_{\exists x.\ f}$	$\mathcal{O}(\lvert\mathsf{B}_f\rvert^2)$	$\mathcal{O}(\lvert\mathsf{B}_f\rvert^2)$

operations are only efficient if f and g have compact ROBDD representations