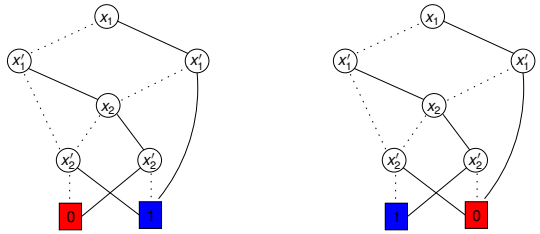
Operations on ROBDDs

Algorithm	Inputs	Output ROBDD
NOT	B_f	$B_{\neg f}$
APPLY	B_f, B_g, binary logical operator *op*	$B_{f\ op\ g}$
RESTRICT	B_f, variable x, boolean value b	$B_{f[x:=b]}$
RENAME	B_f, variables x and y	$B_{f[x:=y]}$
ABSTRACT	B_f, variable x	$B_{\exists x.\, f}$

Negation



negation amounts to interchanging the 0- and 1-leaf

APPLY

- Shannon expansion for binary operations:

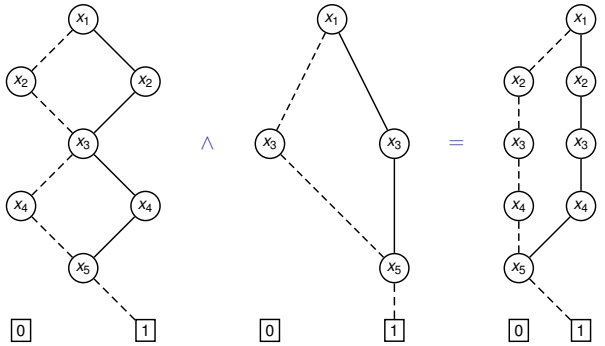
$$f\ op\ g \quad = \quad (x_1 \wedge (f[x_1 := 1]\ op\ g[x_1 := 1])) \vee (\neg x_1 \wedge (f[x_1 := 0]\ op\ g[x_1 := 0]))$$

- A top-down evaluation scheme using Shannon's expansion:
  - let v be the smallest variable in the ordering occurring in B_f or B_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 $B_{f\ op\ g}$ is in $\mathcal{O}(|B_f| \cdot |B_g|)$

Algorithm APPLY(op, B_f, B_g)

if $G(op, v_1, v_2) \neq$ empty **then return** $G(op, v_1, v_2)$ **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 := $MakeNode(Var(v_2), \text{APPLY}(op, v_1, left(v_2)), \text{APPLY}(op, v_1, right(v_2)))$;
else if (v_1 is nonterminal and v_2 is terminal)
  then res := $MakeNode(Var(v_1), \text{APPLY}(op, left(v_1), v_2)), \text{APPLY}(op, right(v_1), v_2))$;
else if ($Var(v_1) = Var(v_2)$)
  then res := $MakeNode(Var(v_1), \text{APPLY}(op, left(v_1), left(v_2))),$
    $\text{APPLY}(op, right(v_1), right(v_2)))$;
else if ($Var(v_1) < Var(v_2)$)
  then res := $MakeNode(Var(v_1), \text{APPLY}(op, left(v_1), v_2)), \text{APPLY}(op, right(v_1), v_2))$;
else {$Var(v_1) > Var(v_2)$}
  res := $MakeNode(Var(v_2), \text{APPLY}(op, v_1, left(v_2)), \text{APPLY}(op, v_1, right(v_2)))$;
$G(op, v_1, v_2)$:= res; {memoize result}
return res

Example

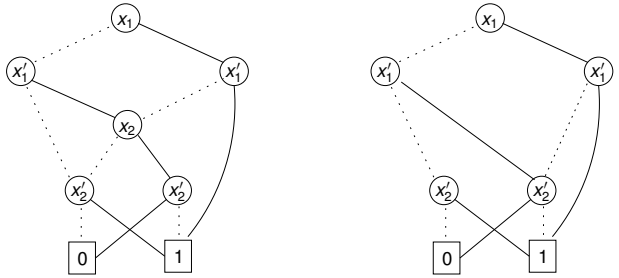


(edges to 0 are omitted.)

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 all vertices only reachable through v
  - (if necessary) reduce (only above v)

RESTRICT



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

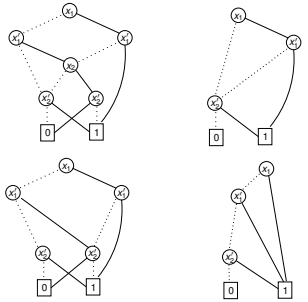
ABSTRACT

- 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

Example



ROBDDs B_f (left up), $B_{f[x_2:=0]}$ (right up), $B_{f[x_2:=1]}$ (left down), and $B_{\exists x_2.\ f}$ (right down)

Operations on ROBDDs

Algorithm	Output	Time complexity	Space complexity
NOT	$B_{\neg f}$	$\mathcal{O}(\lvert B_f \rvert)$	$\mathcal{O}(\lvert B_f \rvert)$
APPLY	$B_{f\ op\ g}$	$\mathcal{O}(\lvert B_f \rvert \cdot \lvert B_g \rvert)$	$\mathcal{O}(\lvert B_f \rvert \cdot \lvert B_g \rvert)$
RESTRICT	$B_{f[x:=b]}$	$\mathcal{O}(\lvert B_f \rvert)$	$\mathcal{O}(\lvert B_f \rvert)$
RENAME	$B_{f[x:=y]}$	$\mathcal{O}(\lvert B_f \rvert)$	$\mathcal{O}(\lvert B_f \rvert)$
ABSTRACT	$B_{\exists x.\ f}$	$\mathcal{O}(\lvert B_f \rvert^2)$	$\mathcal{O}(\lvert B_f \rvert^2)$

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

Symbolic CTL model checking: Computing $Sat(\Phi)$

Require: CTL-formula Φ in ENF
Ensure: ROBDD $B_{Sat(\Phi)}$

---

switch(Φ):

true	:	**return** CONST(1);
false	:	**return** CONST(0);
x_i	:	**return** ROBDD B_f for $f(x_1, \ldots, x_n) = x_i$;
$\neg\Psi$	:	**return** NOT($bddSat(\Psi)$)
$\Phi_1 \wedge \Phi_2$	:	**return** APPLY($\wedge, bddSat(\Phi_1), bddSat(\Phi_2)$)
EX Ψ	:	**return** $bddEX(\Psi)$;
E (Φ_1 U Φ_2)	:	**return** $bddEU(\Phi_1, \Phi_2)$
EG Ψ	:	**return** $bddEG(\Psi)$

end switch

Symbolic model checking: bddEX

$$Sat(\mathrm{EX}\,\Phi) \;=\; \{\, q \in Q \mid \exists q'.\ (q, q') \in E \text{ and } q' \in Sat(\Phi) \,\}$$

Require: CTL-formula Φ in ENF
Ensure: ROBDD $B_{Sat(\bigcirc\Phi)}$

$B := bddSat(\Phi)$; $\{Sat(\Phi)\}$
$B :=$ RENAME$(B, x_1, \ldots, x_n, x'_1, \ldots, x'_n)$;
$B :=$ APPLY$(\wedge, B_{\rightarrow}, B)$; $\{Pre(Sat(\Phi))\}$
return ABSTRACT$(B, x'_1, \ldots, x'_n)$

Symbolic model checking: bddEU

Require: CTL-formulas Φ, Ψ in ENF
Ensure: ROBDD $B_{Sat(\mathsf{E}\,(\Phi\,\mathsf{U}\,\Psi))}$

---

var $\mathsf{N}, \mathsf{P}, \mathsf{B} : \mathit{ROBDD}$;
$\mathsf{N} := \mathit{bddSat}(\Psi)$;
$\mathsf{P} := \textsc{Const}(0)$;
$\mathsf{B} := \mathit{bddSat}(\Phi)$;
while $(\mathsf{N} \neq \mathsf{P})$ **do**
  $\mathsf{P} := \mathsf{N}$; $\{T_i\}$
  $\mathsf{N} := \textsc{Rename}(\mathsf{N}, x_1, \ldots, x_n, x'_1, \ldots, x'_n)$;
  $\mathsf{N} := \textsc{Apply}(\wedge, \mathsf{B}_{\rightarrow}, \mathsf{N})$; $\{\mathit{Pre}(T_i)\}$
  $\mathsf{N} := \textsc{Abstract}(\mathsf{N}, x'_1, \ldots, x'_n)$;
  $\mathsf{N} := \textsc{Apply}(\wedge, \mathsf{N}, \mathsf{B})$; $\{\mathit{Pre}(T_i) \cap \mathit{Sat}(\Phi)\}$
  $\mathsf{N} := \textsc{Apply}(\vee, \mathsf{P}, \mathsf{N})$; $\{T_{i+1} = T_i \cup \ldots\ldots\}$
end while
return N

Symbolic model checking: bddEG

Require: CTL-formula Φ in ENF
Ensure: ROBDD $B_{Sat(\mathsf{EG}\,\Phi)}$

---

var $\mathsf{N}, \mathsf{P}, \mathsf{B} : \mathit{ROBDD}$;
$\mathsf{B} := \mathit{bddSat}(\Phi)$;
$\mathsf{N} := \mathsf{B}$;
$\mathsf{P} := \textsc{Const}(0)$;
while $(\mathsf{N} \neq \mathsf{P})$ **do**
  $\mathsf{P} := \mathsf{N}$; $\{T_i\}$
  $\mathsf{N} := \textsc{Rename}(\mathsf{N}, x_1, \ldots, x_n, x'_1, \ldots, x'_n)$;
  $\mathsf{N} := \textsc{Apply}(\wedge, \mathsf{B}_{\rightarrow}, \mathsf{N})$; $\{\mathit{Pre}(T_i)\}$
  $\mathsf{N} := \textsc{Abstract}(\mathsf{N}, x'_1, \ldots, x'_n)$;
  $\mathsf{N} := \textsc{Apply}(\wedge, \mathsf{N}, \mathsf{B})$; $\{\mathit{Pre}(T_i) \cap \mathit{Sat}(\Phi)\}$
  $\mathsf{N} := \textsc{Apply}(\wedge, \mathsf{P}, \mathsf{N})$; $\{T_{i+1} = T_i \cap \ldots\ldots\}$
end while
return N