Mạch tổ hợp nhiều mức

Trần Anh Điền	10070475
Hoàng Văn Long	10070485
Võ Bảo Hùng	09070441

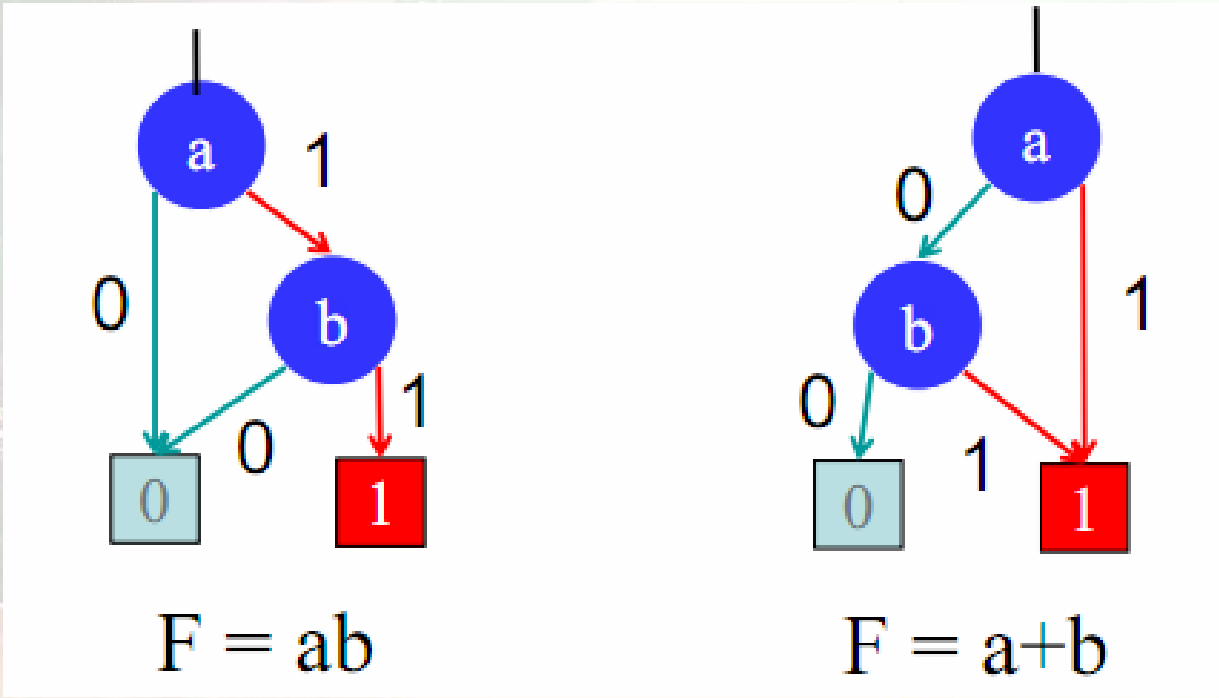
Outline

- Giới thiệu BDD
- Tính chất BDD, toán tử bù, cofactor, cạnh đảo trên BDD
- Toán tử APPLY, Multiplexor-based network

Cây quyết định nhị phân thứ tự

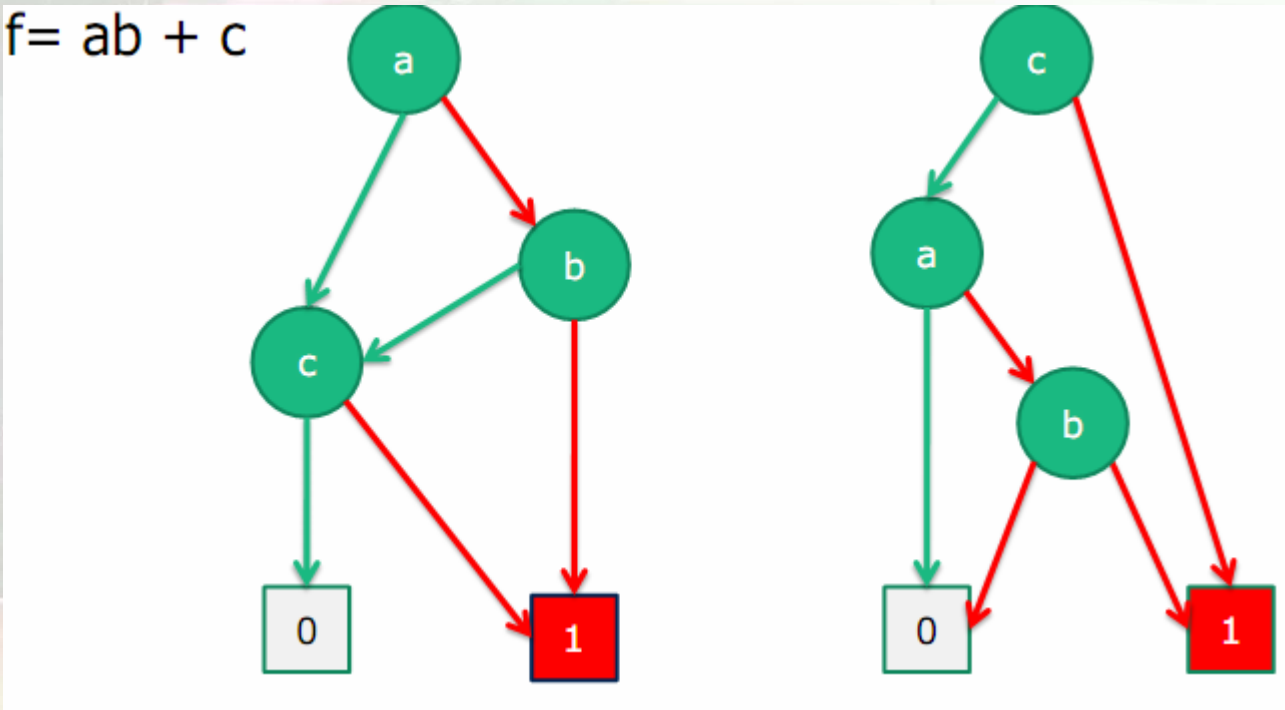
- OBDD (Order Binary Decision Diagram)
  - Là một đồ thị không có chu trình.
  - Đỉnh terminal biểu diễn bằng hình vuông.
  - Đỉnh nonterminal biểu diễn bằng hình tròn.
  - Đỉnh con low được trỏ đến bởi cạnh 0.
  - Đỉnh con high được trỏ đến bởi cạnh 1.
- OBDD biểu diễn OFF-set và ON-set của một hàm dưới dạng các cover tách biệt nhau.
  - Mỗi cube của cover: đường từ đỉnh gốc đến đỉnh terminal của cây.

Cây quyết định nhị phân thứ tự



- Thứ tự a=1, b=2

Cây quyết định nhị phân thứ tự



- Thứ tự a = 1, b = 2, c = 3 Thứ tự c = 1, a = 2, b =3

Khai triển Shannon

- Khai triển Shannon của hàm F

$$F(x, y, z) = x'.F_0 + x.F_1$$

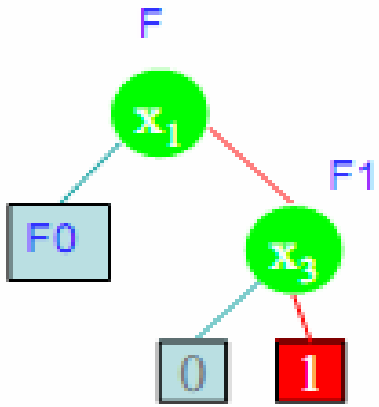
- Khai triển Shannon là chuẩn tắc
  - Cho một hàm F và một biến x trong hàm F, phần phụ đại số F_0 và F_1 là đơn định
- Định nghĩa không hình thức cho BDD
  - Khai triển Shannon được áp dụng đệ quy trên hàm và các phần phụ đại số của nó
  - Một node mới sẽ được thêm vào ở mỗi khai triển Shannon
  - Thứ tự các biến là cố định trên tất cả các nhánh
  - Rút gọn đồ thị

Xây dựng BDD dùng khai triển Shannon

F = x3 (x1+x2)

$F0 = F|_{x1=0} = x2x3$

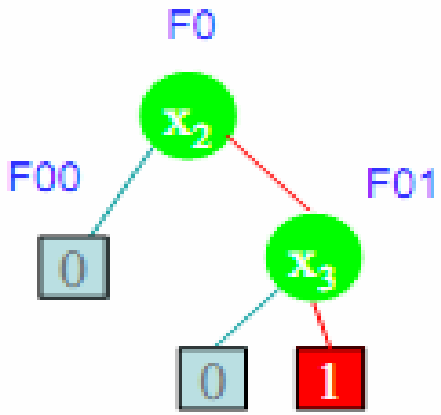
$F1 = F|_{x1=1} = x3$

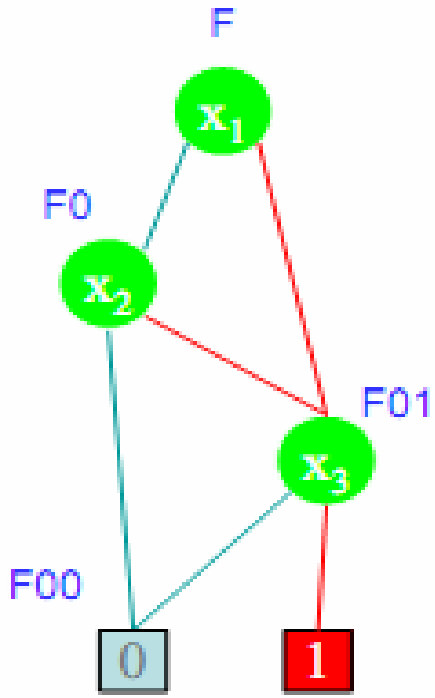


F0 = x2x3

$F00|_{x2=0} = 0$

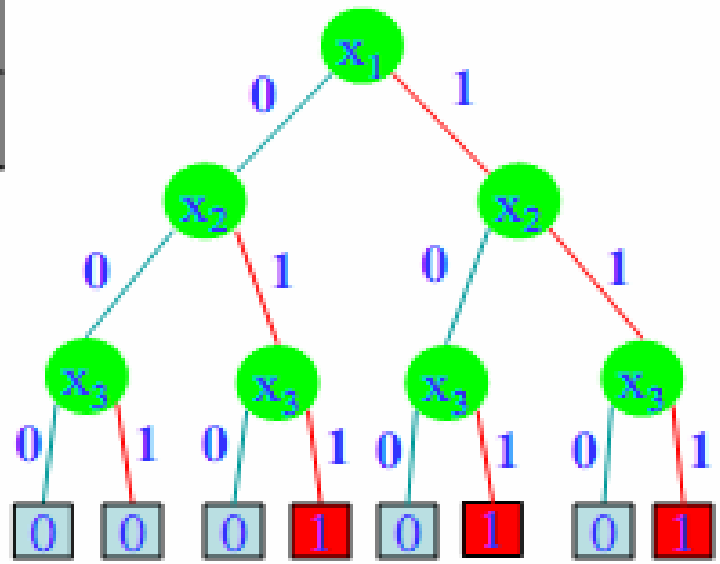
$F01|_{x2=1} = x3$





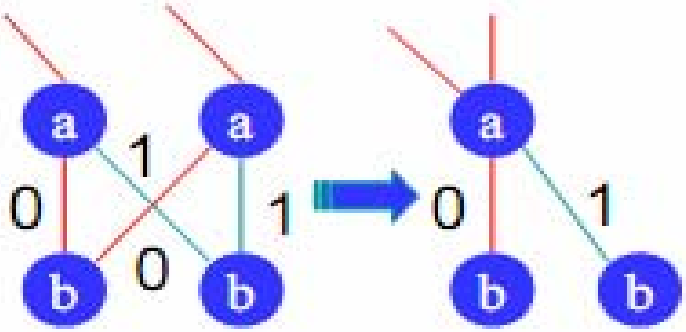
Xây dựng BDD từ bảng thực trị

x_1	0	0	0	0	1	1	1	1
x_2	0	0	1	1	0	0	1	1
x_3	0	1	0	1	0	1	0	1
F	0	0	0	1	0	1	0	1

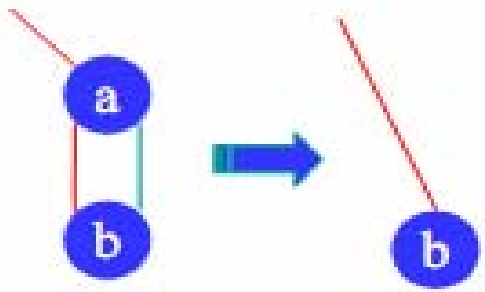


Rút gọn OBDD

Quy tắc 1: Nhập các đỉnh đẳng cấu



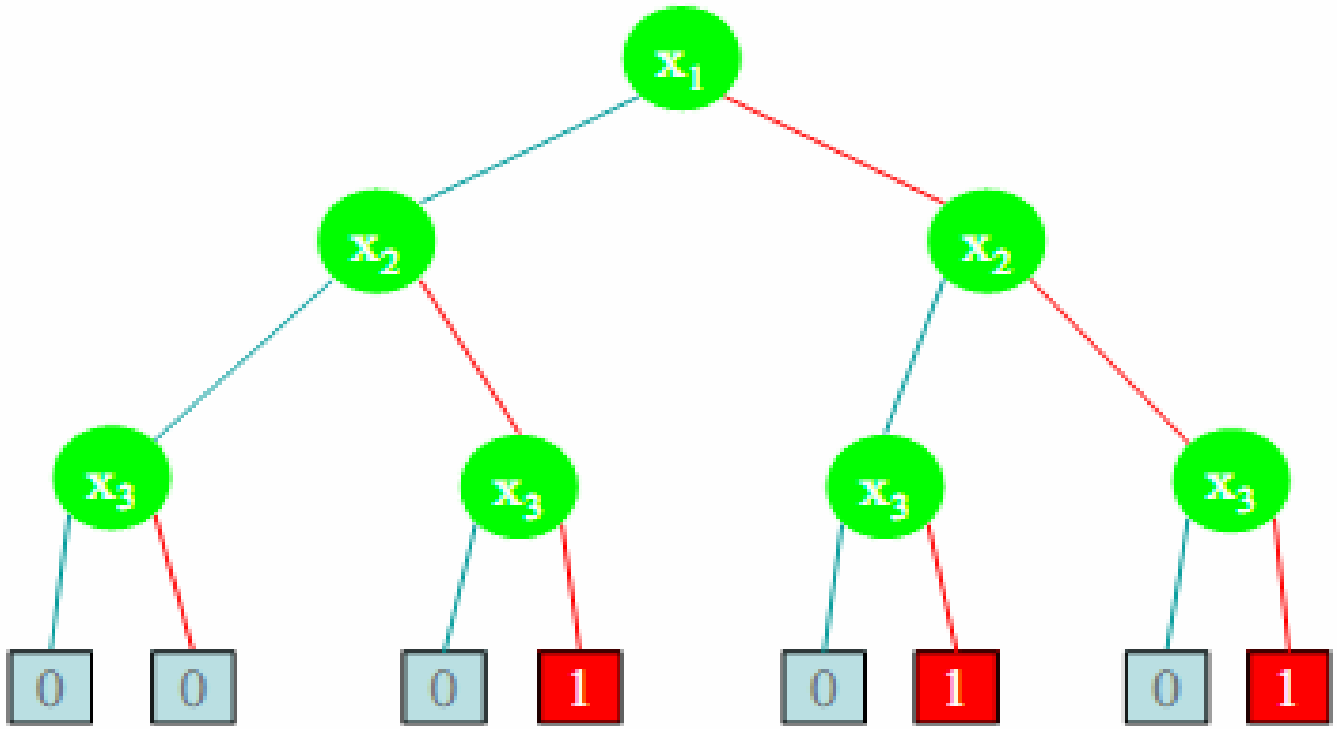
Quy tắc 2: Loại bỏ các node trung gian dư thừa



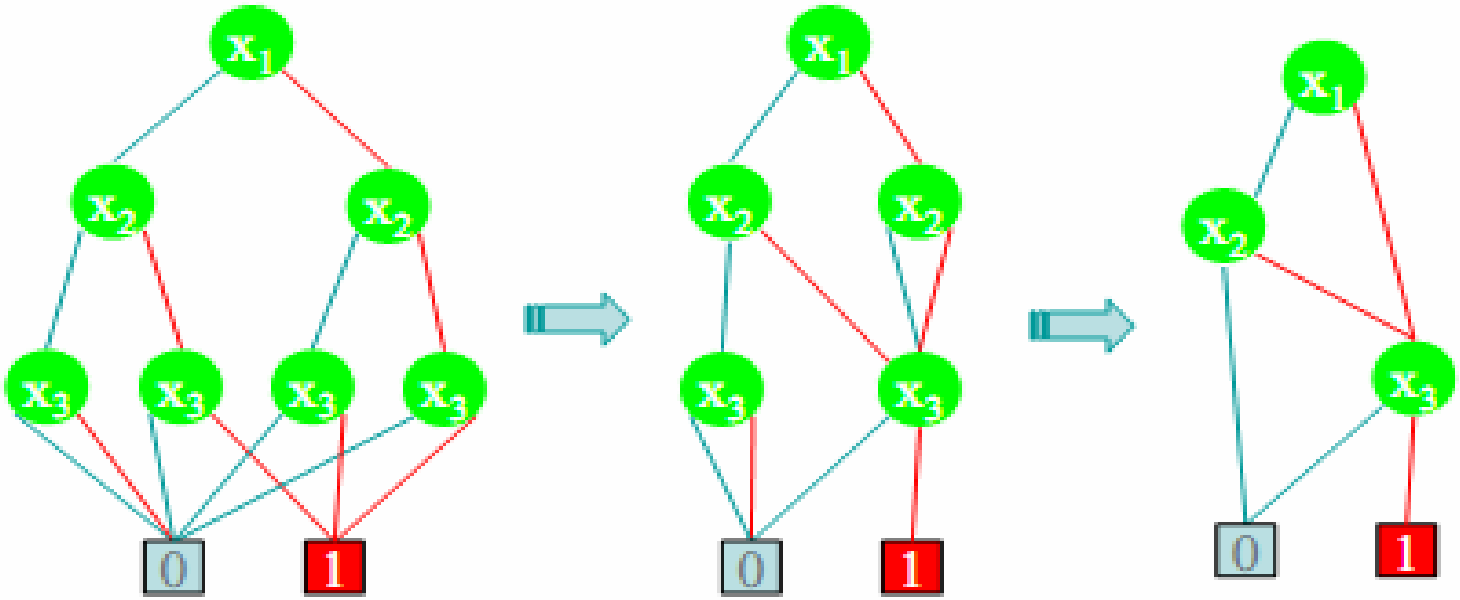
Các định nghĩa

- Đẳng cấu (isomorphism): 2 OBDD G1 và G2 gọi là đẳng cấu nếu tồn tại một hàm ánh xạ 1-1 σ từ tập các đỉnh của G1 sang tập các đỉnh của G2 sao cho với bất kỳ đỉnh v, nếu σ (v) = w thì
  - v và w là các đỉnh terminal với value(v) = value(w)
  - hoặc v và w là các đỉnh nonterminal với index(v) = index(w)
- σ (low(v)) = low(w)
- σ (high(v)) = high(w)

Ví dụ



Ví dụ(tt)

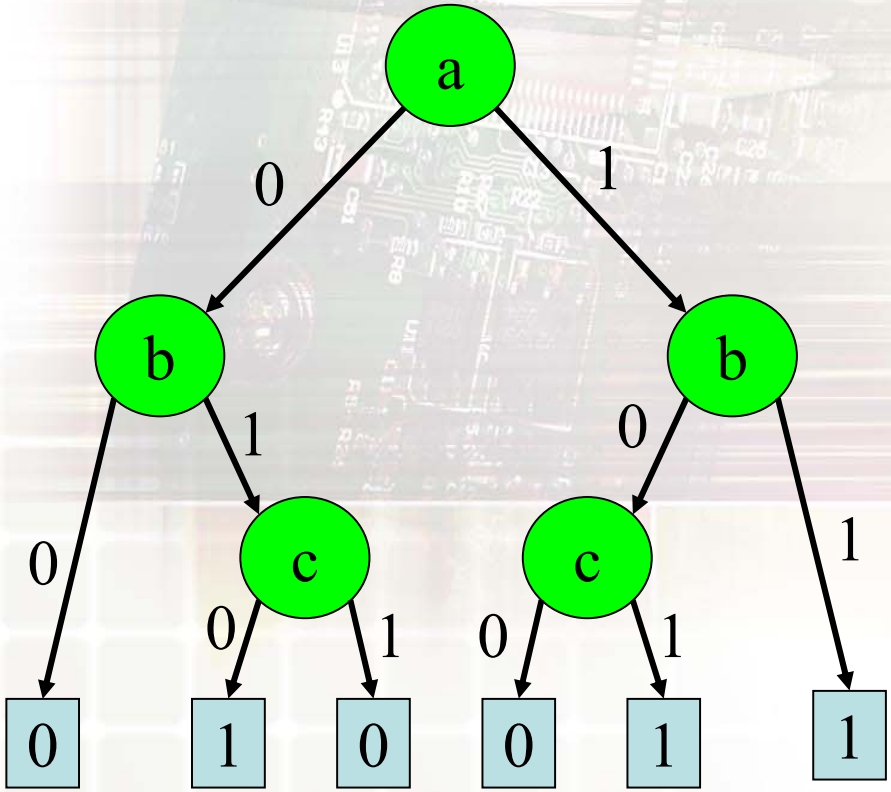


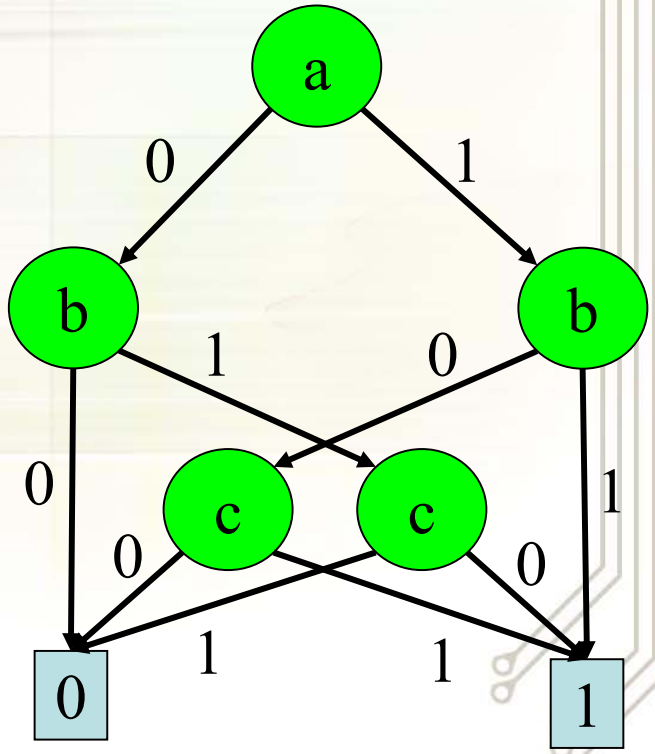
Cây quyết định Node đẳng cấu ROBDD

Các định nghĩa

- Một OBDD G là rút gọn (reduced) nếu nó
  - Không chứa đỉnh v nào với low(v) = high(v)
  - Không chứa hai đỉnh v, w nào mà đồ thị con có đỉnh gốc là v và w là đẳng cấu.

Reduction: OBDD → ROBDD





Mạch tương đương sử dụng ROBDD

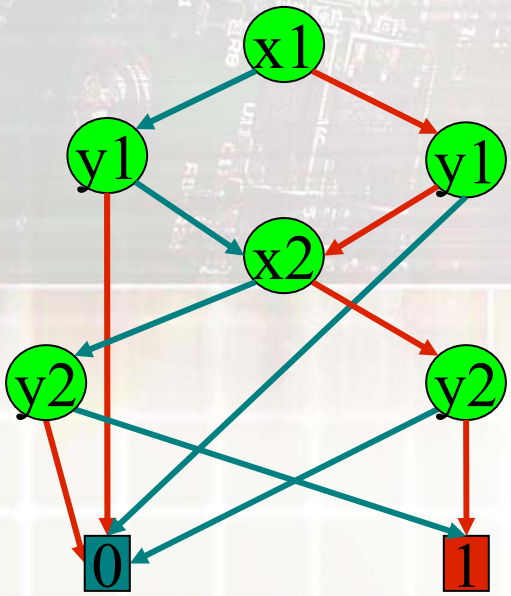
- Phương pháp kiểm tra sự tương đương giữa hai mạch luận lý 2 hay nhiều mức
  - Lựa chọn thứ tự giống nhau cho các ngõ nhập cơ bản của 2 mạch
  - Xây dựng ROBDD cho các ngõ xuất cơ bản của 2 mạch
  - Kiểm tra tính đẳng cấu của 2 ROBDD đã xây dựng để xác định sự tương đương của 2 mạch

Ảnh hưởng của thứ tự biến

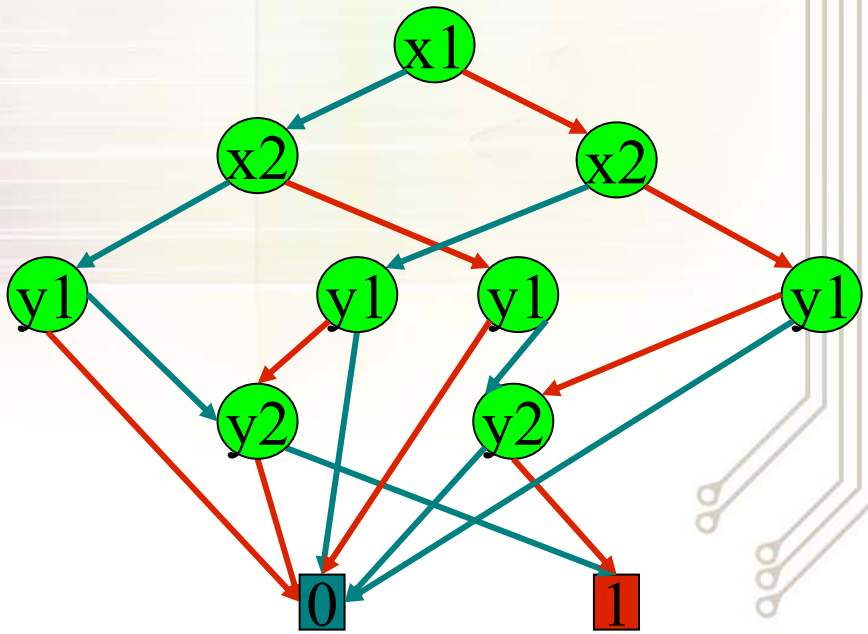
- Việc thay đổi thứ tự của các biên sẽ ảnh hưởng đến kích thước của ROBDD

$$\varphi = (x1 \Leftrightarrow y1) \wedge (x2 \Leftrightarrow y2)$$

x1<y1<x2<y2

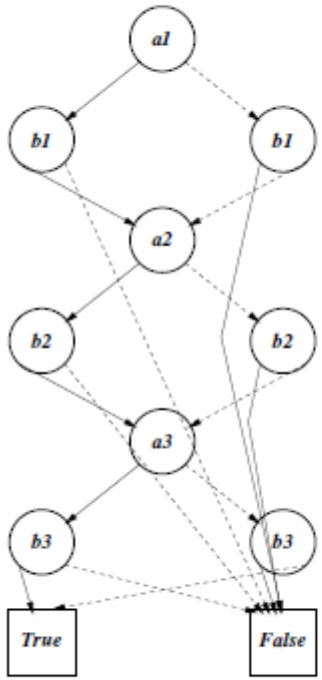


x1<x2<y1<y2

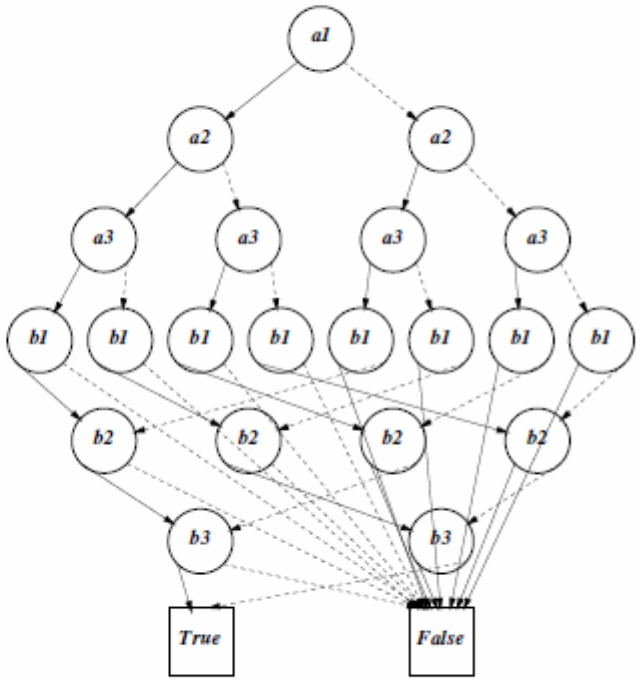


Ảnh hưởng của thứ tự biến

$$\varphi = (a1 \Leftrightarrow b1) \wedge (a2 \Leftrightarrow b2) \wedge (a3 \Leftrightarrow b3)$$



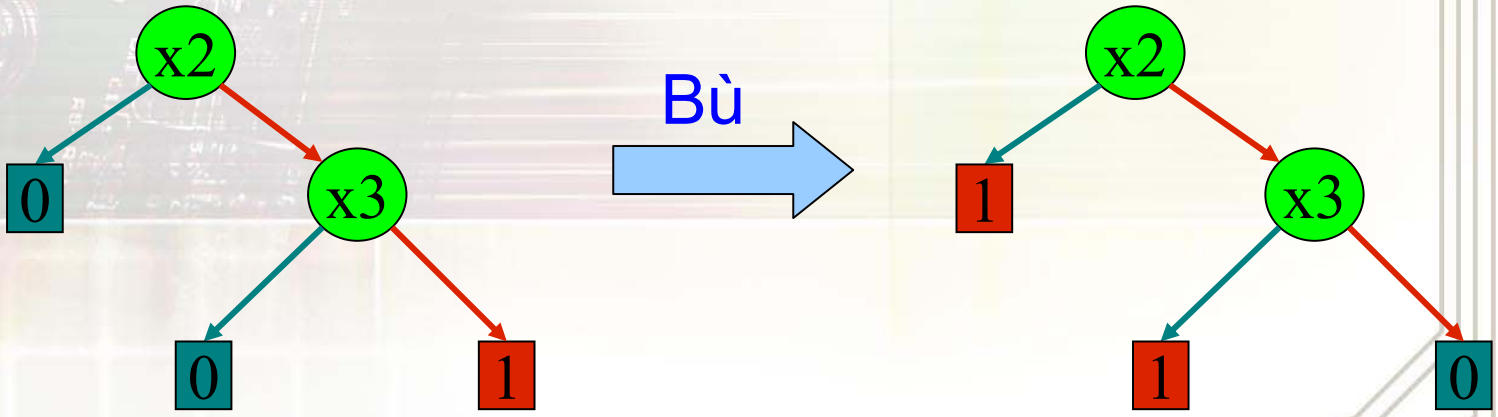
Linear size



Exponential size

Bù (Complement)

- Bù của một OBDD: thay thế các đỉnh terminal bởi các đỉnh có giá trị ngược
  - Đỉnh 0 → đỉnh 1
  - Đỉnh 1 → đỉnh 0



F=x2.x3

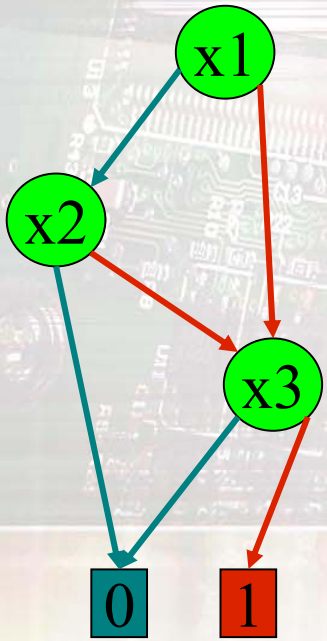
F'=(x2.x3)'

Cofactor

- Cofactor của ROBDD đối với biến x_i
  - Thay thế tất cả các đỉnh v với index(v) = i bằng high(v)
- Cofactor của ROBDD đối với biến x'_i
  - Thay thế tất cả các đỉnh v với index(v) = i bằng low(v)

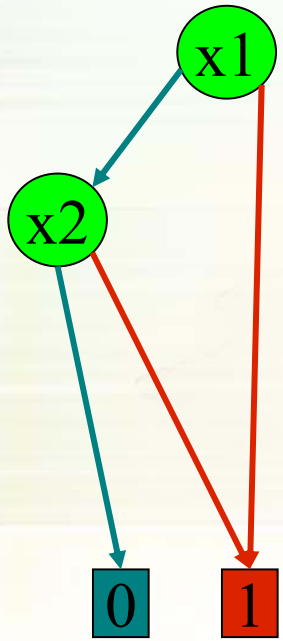
Cofactor

- Thay thế tất cả các đỉnh v với index(v) = i bằng high(v)



F=x3(x1+x2)

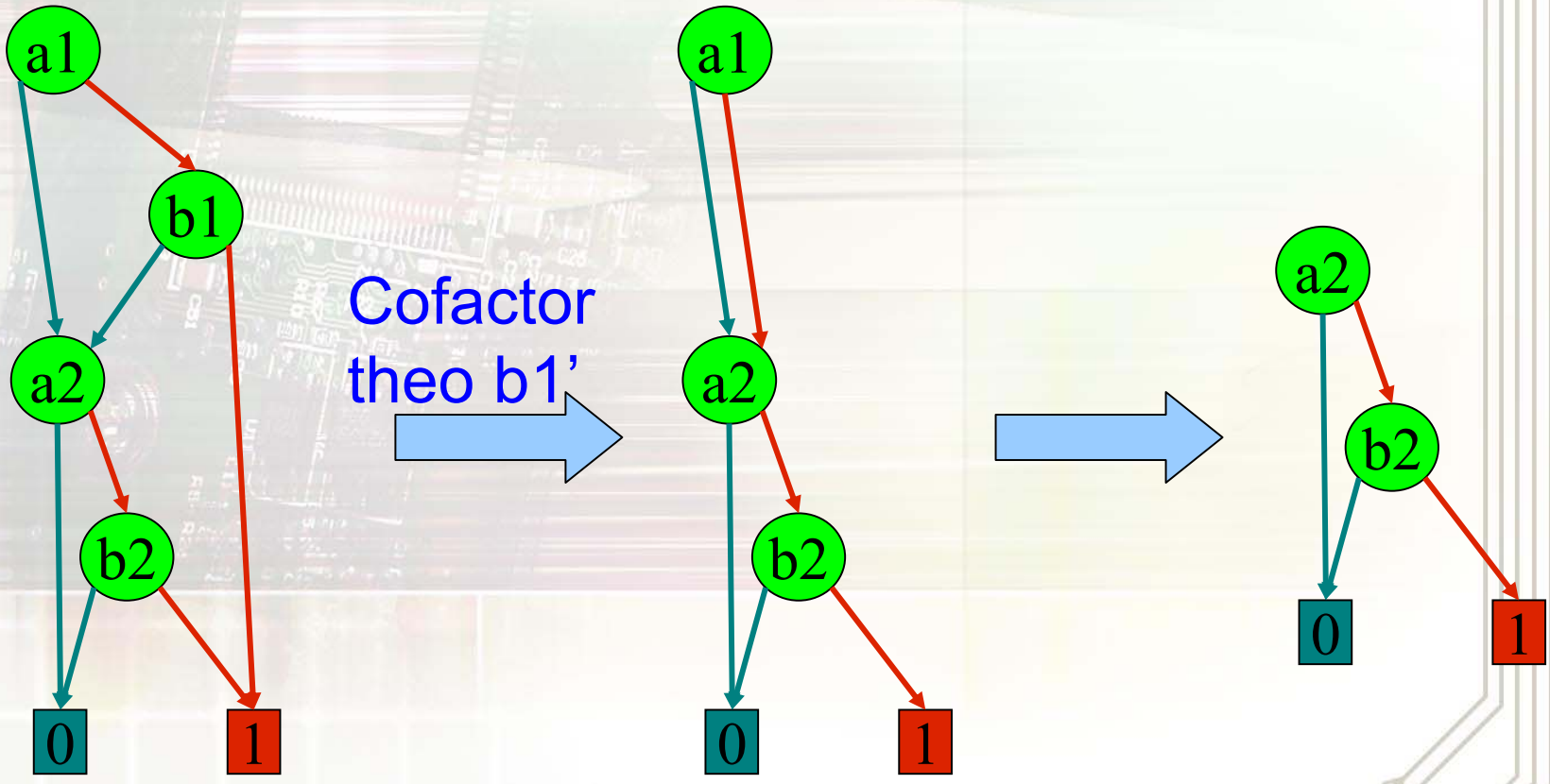
Cofactor theo x3



F_{x3}=x1+x2

Cofactor

- Thay thế tất cả các đỉnh v với index(v) = i bằng low(v)



$F = a_1b_1 + a_2b_2$

$F_{b'1} = a_2b_2$

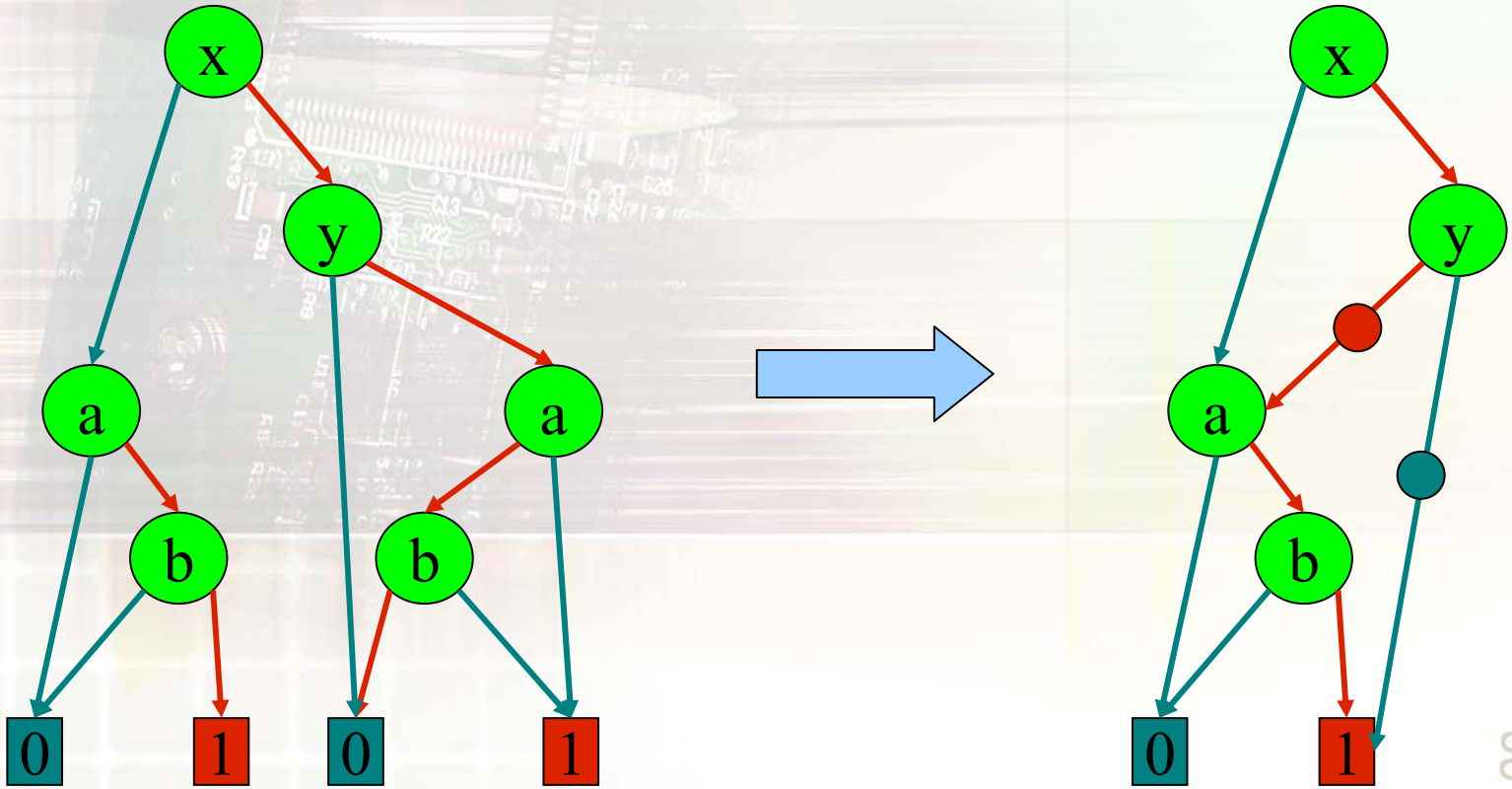
Cải tiến ROBDD

- Cạnh đảo
  - Sử dụng khi hàm F và các thành phần bù của hàm F được dùng để hình thành một hàm mới
  - Tiết kiệm bộ nhớ, thời gian thao tác trên đồ thị
  - Tránh dư thừa các đỉnh trong cấu trúc ROBDD

Cạnh đảo

$$F = a.b.x + a'.y + b'.y$$

$$= a.b.x + (a.b)'.y$$



Toán tử APPLY

- Toán tử APPLY(F,G)
  - Đại diện cho toán tử 2 ngôi bất kì (AND, OR, XOR…)
  - Toán tử cơ bản hỗ trợ thao tác trên cấu trúc ROBDD
  - Áp dụng trực tiếp trên ROBDD
- Toán tử APPLY được tính một cách đệ qui

$$APPLY(F_v, G_w) = x_i.APPLY(F_{high(v)}, G_{high(v)}) + x'_i.APPLY(F_{low(v)}, G_{low(v)})$$

Toán tử APPLY – Giải thuật thực hiện

- Độ phức tạp hàm mũ theo số biến ngõ nhập
  - Mỗi lần gọi APPLY cho đỉnh nonterminal: gọi 2 lần đệ qui
  - 2 phương pháp tinh chế
    - Áp dụng các trường hợp đặc biệt
    - Tạo bảng lưu trữ các node đã tạo trước đó
    - Độ phức tạp: O(|G1|.|G2|)

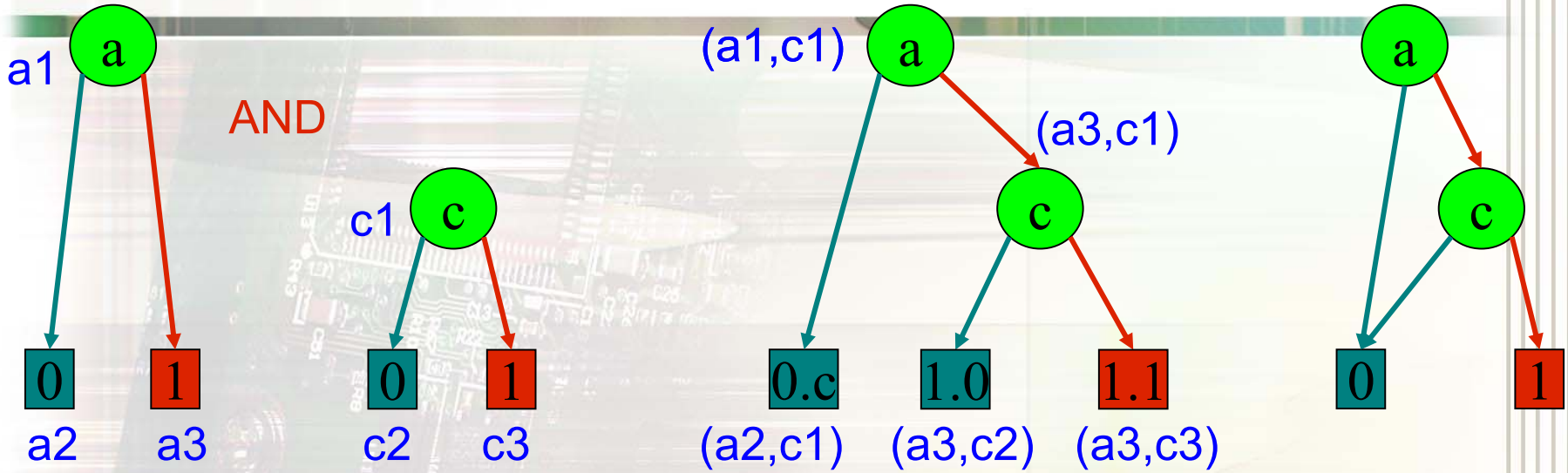
Toán tử APPLY – Giải thuật thực hiện

```
APPLY(Fv, Gw) {
   if ( IsAlreadyComputed( Fv, Gw) ) return result

   if ( value(v)∈{0,1} && value(w)∈{0,1} )
         u= newNode( value(u) = value(v) <OP> value(w) )
   if ( index(v) = index(w)= i)
         u= newNode(index(u) = i, APPLY( low(v),low(w) ), APPLY(
                                                  high(v),high(w) ) )
   else if ( index(v) = i && index(w) > i)
         u= newNode( index(u) = i, APPLY(low(v),w), APPLY(high(v), w)))
   else if ( index(v) > i && index(w) = i)
         u= newNode( index(u) = i, APPLY(v,low(w)), APPLY(v,high(w)))
   InsertComputed( Fv, Gw, u )
   return u
}
```

Toán tử APPLY – Ví dụ

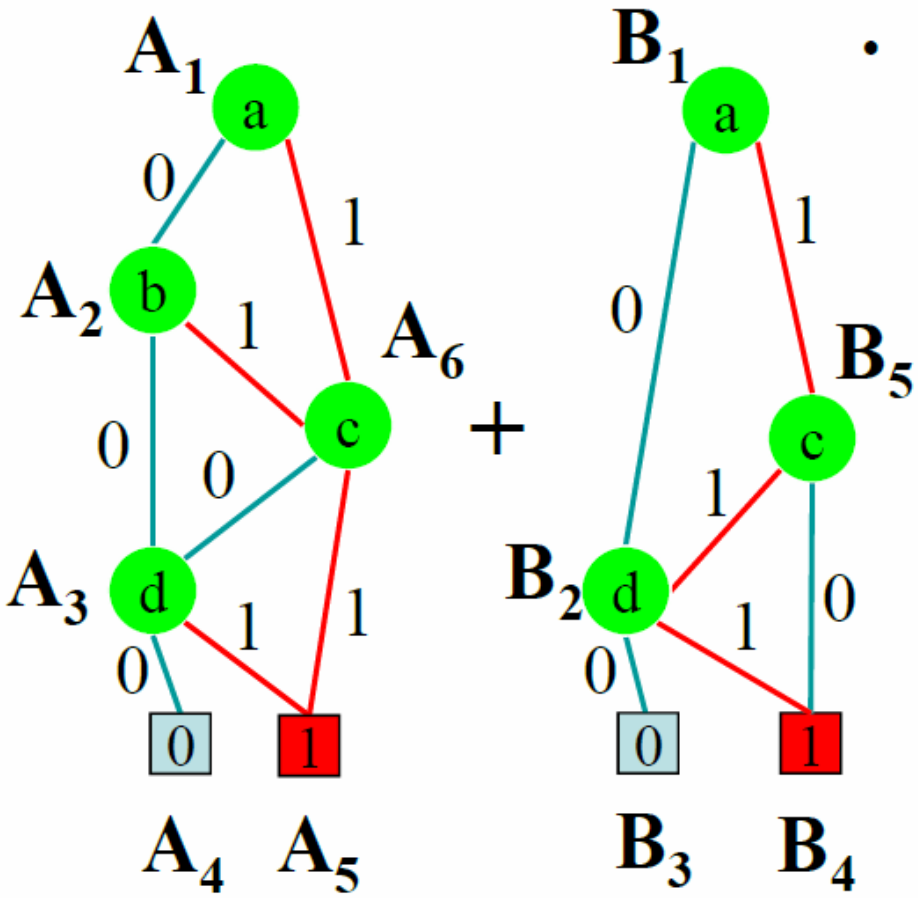
F = a AND c



- APPLY(a1,c1)
  - APPLY(a2,c1)
    - APPLY(a2,c2)=0, APPLY(a2,c3)=0
  - APPLY(a3,c1)
    - APPLY(a3,c2)=0, APPLY(a3,c3)=1

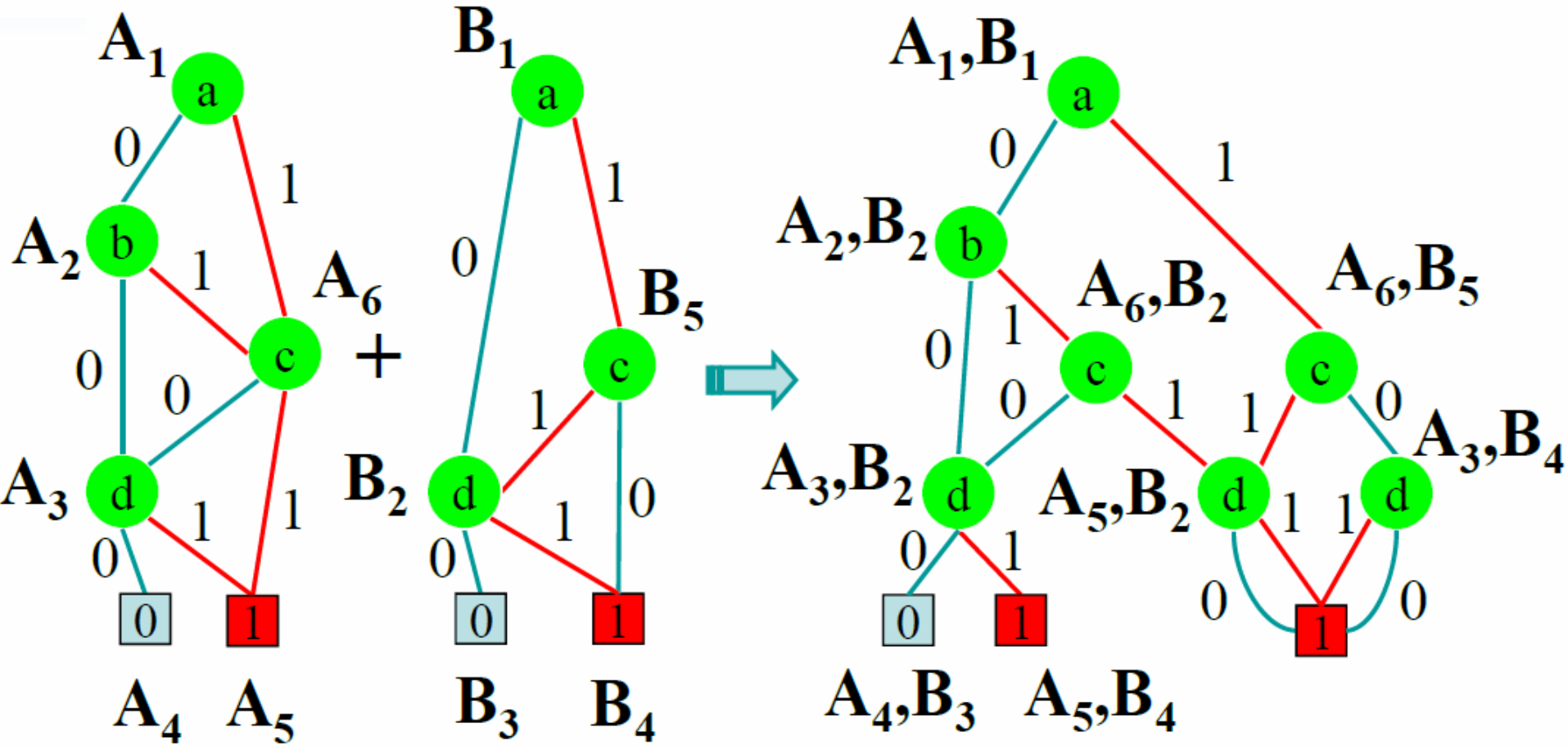
Toán tử APPLY – Ví dụ

F=ac+bc+d *G=ac'+d* *F+G = ?*

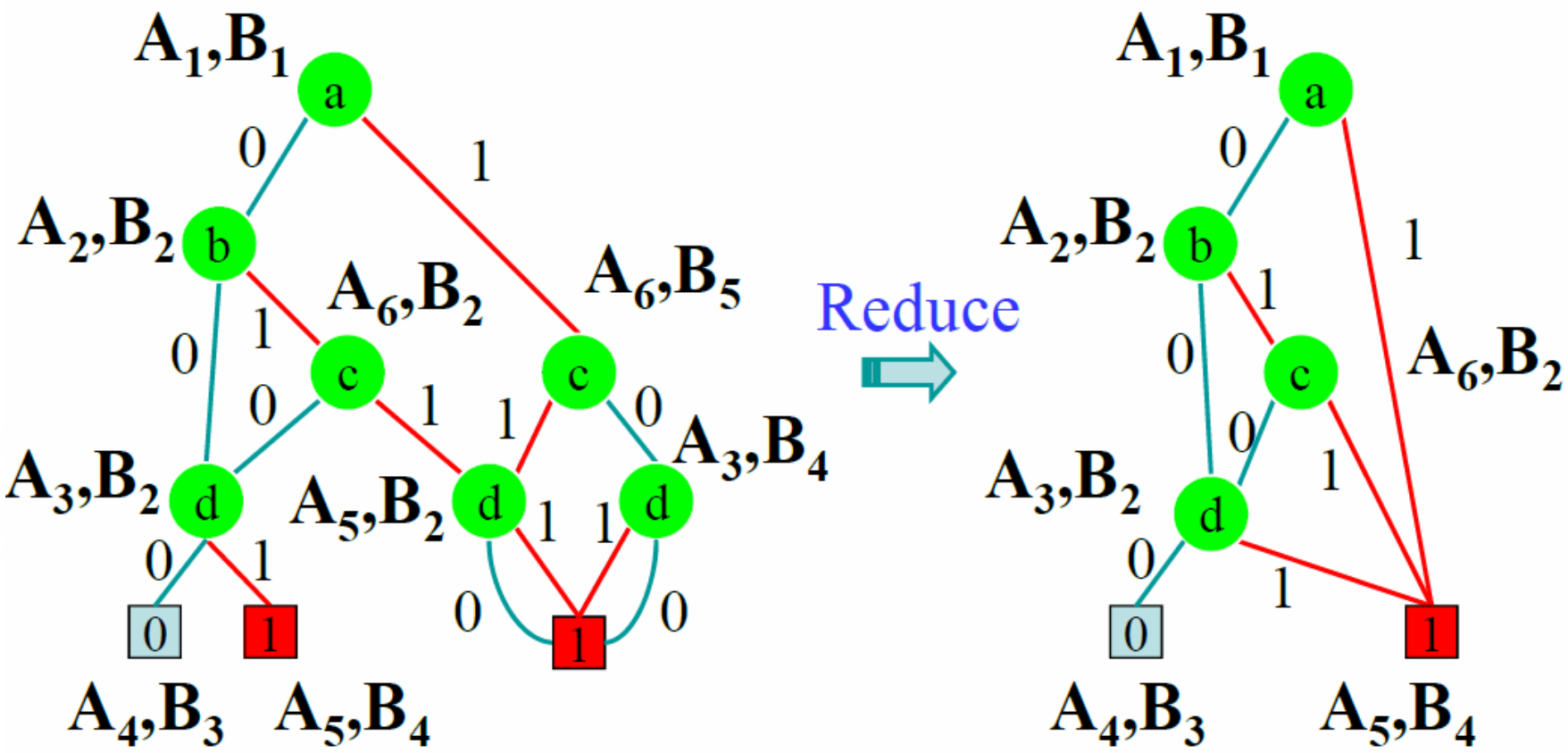


- APPLY(A1, B1)
  - APPLY(A2, B2)
    - APPLY(A3, B2)
      - APPLY(A4, B3) = 0
      - APPLY(A5, B4) = 1
    - APPLY(A6, B2)
      - APPLY(A3, B2)
      - APPLY(A5, B2)
        - APPLY(A5, B3) = 1
        - APLLY(A5, B4) = 1
  - APPLY(A6, B5)
    - APPLY(A3, B4)
    - APPLY(A5, B2)

Toán tử APPLY – Ví dụ



Toán tử APPLY – Ví dụ



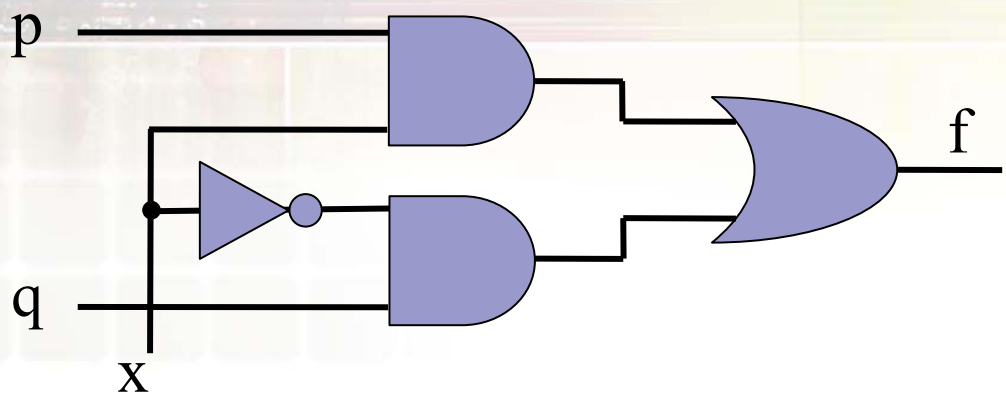
$F=ac+bc+d$ $G=ac'+d$ $F+G = a + bc + d$

Mạng Multiplexor-based

- Mỗi multiplexor hiện thực cho hàm luận lý

$$f = x.p + x'.q$$

- Mạng multiplexor-based
  - Thành phần cơ bản là các mulitplexor
  - liên hệ trực tiếp với BDD và tương đương với hàm luận lý biểu diễn bởi BDD



Mạng Multiplexor-based

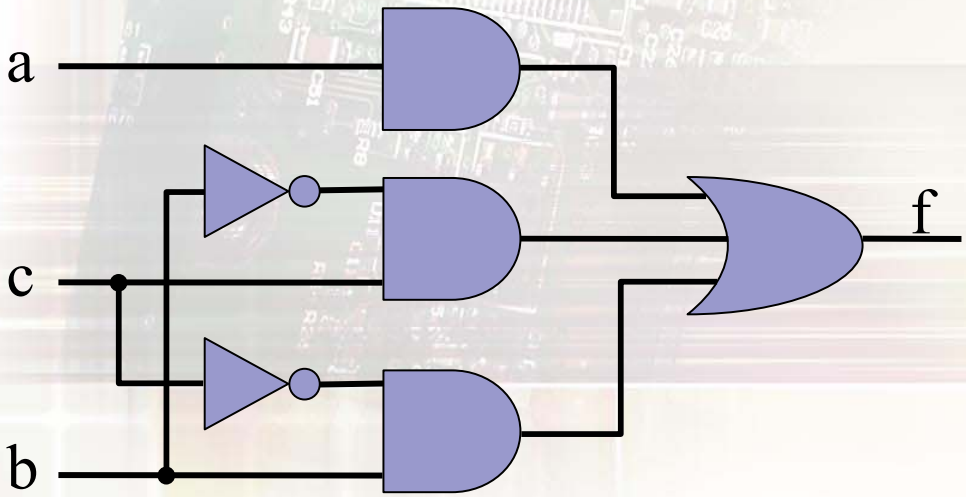
- Mạng Multiplexor-based *η* được suy ra từ BDD G dựa trên phép biến đổi sau:
  - Nếu low(v) và high(v) là các đỉnh *nonterminal*: thay thế v bằng multiplexor hai ngõ nhập tương ứng với hàm

$$f_v = x'_{index(v)} \cdot f_{low(v)} + x_{index(v)} \cdot f_{high(v)}$$

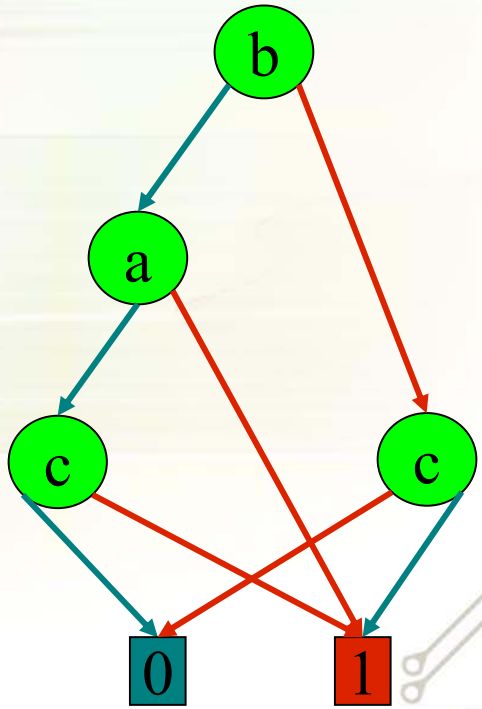
  - Nếu low(v) hoặc high(v) là các đỉnh *terminal*: thay thế v bằng các hàm đơn giản hoá tương ứng

Mạng Multiplexor-based

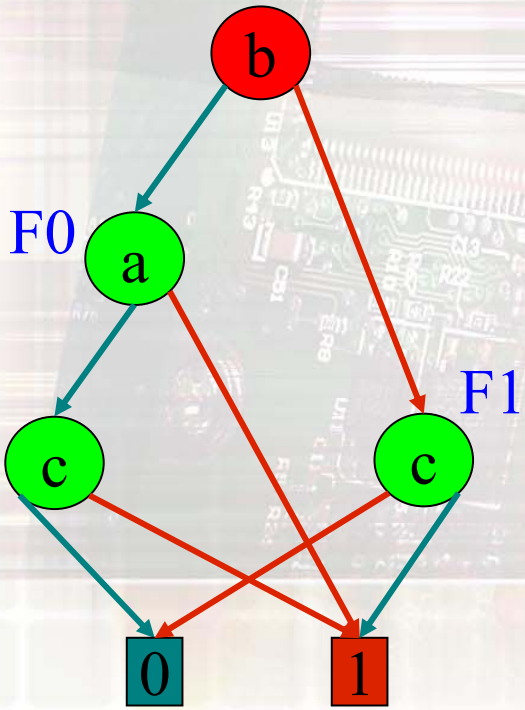
Mạch 2 mức

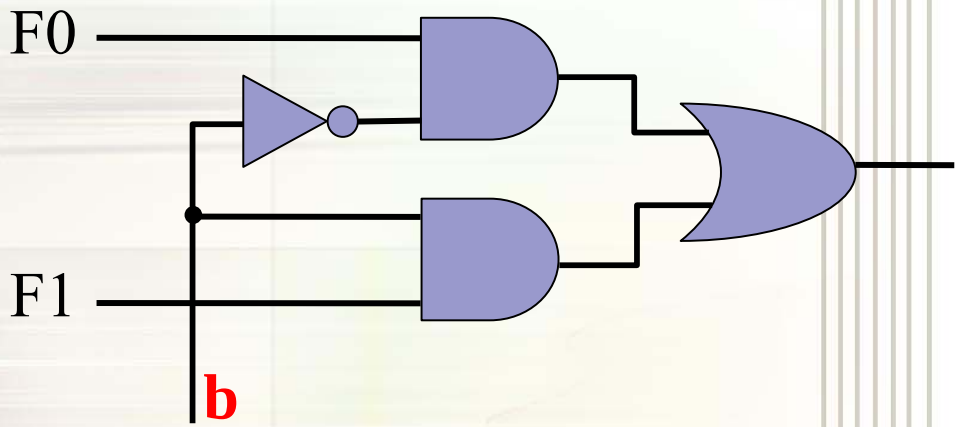


BDD tương ứng



Mạng Multiplexor-based



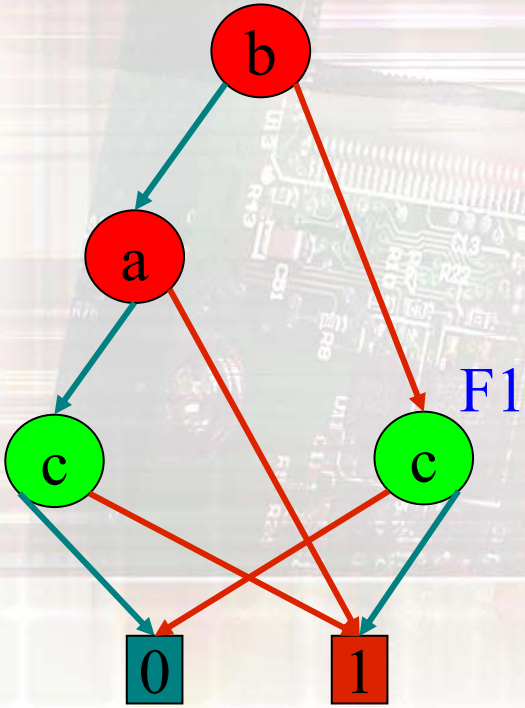


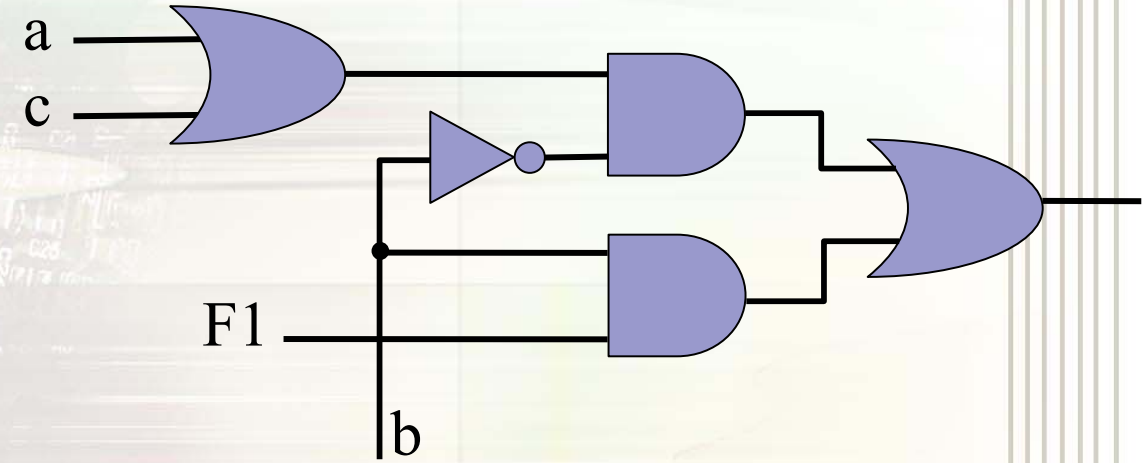
- Đỉnh b
  - low(b)= F0
  - high(b) = F1

  Multiplexor thay thế cho đỉnh b

  f = b'.F0 + b.F1

Mạng Multiplexor-based



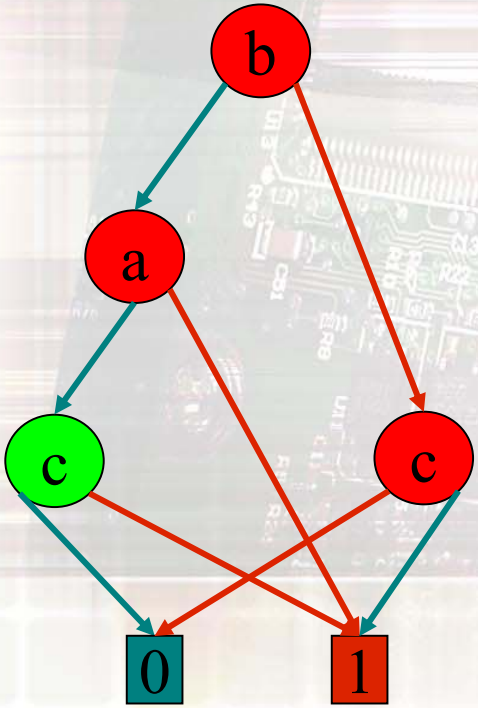


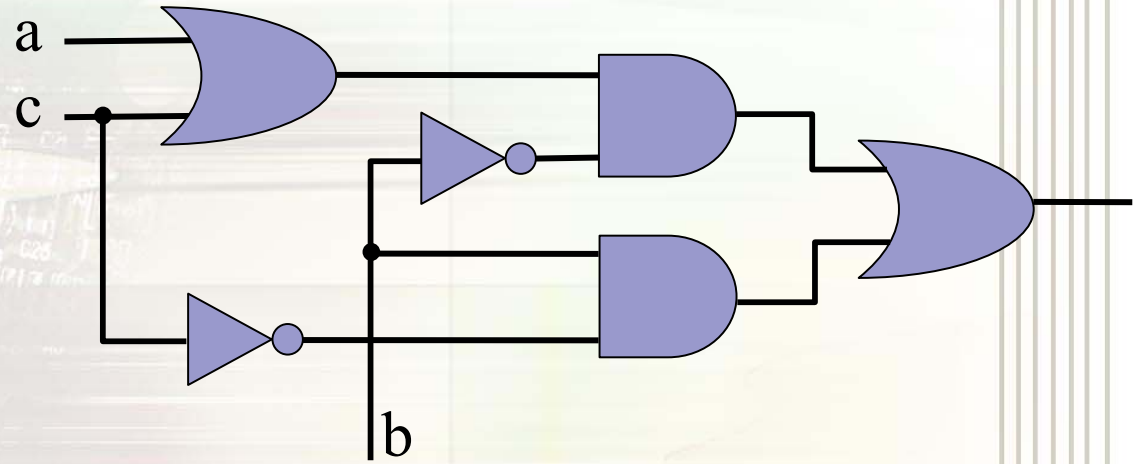
- Đỉnh a
  - low(a)= c
  - high(a) = 1

  Multiplexor thay thế cho đỉnh a

  $f = a'.f_{low(a)} + a.f_{high(a)} = c + a$

Mạng Multiplexor-based





- Đỉnh c
  - low(c) và high(c) là terminal

  Multiplexor thay thế cho đỉnh c là c’

Thank you