COMP6210: Automated Software Verification

Binary Decision Diagrams (BDDs)

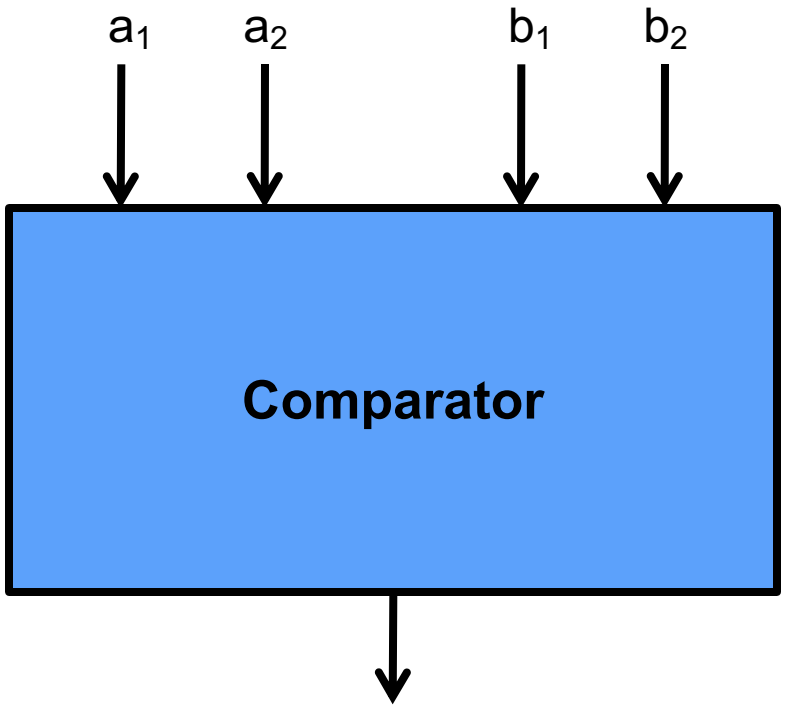
PavelNaumov

- Reduced Ordered Binary Decision Diagrams (ROBDDs)
- canonical representation of Boolean formulas
- can be manipulated efficiently
  - Conjunction, Disjunction, Negation, Existential Quantification

BDDs in a nutshell

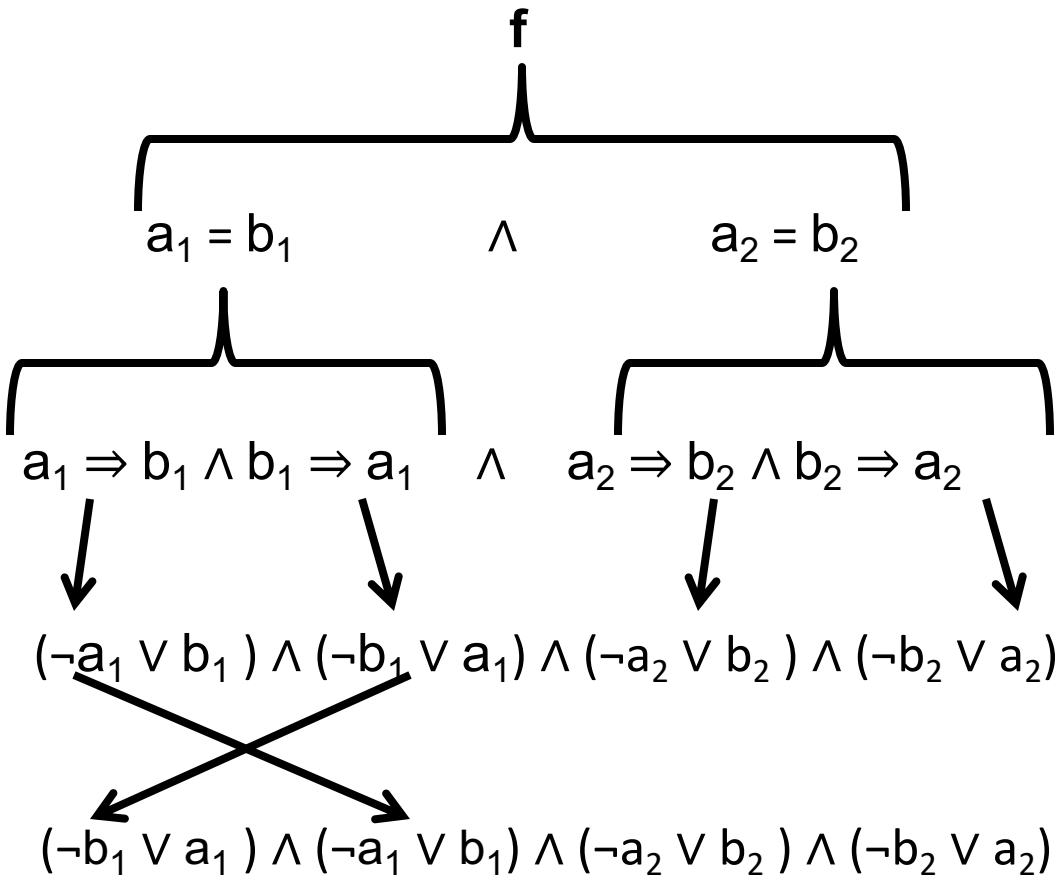
- *"one of the only really fundamental data structures that came out in the last twenty-five years"* (Donald Knuth, *Fun with Binary Decision Diagrams*, 2008 lecture)
- Typically mean Reduced Ordered Binary Decision Diagrams (ROBDDs)
- Canonical representation of Boolean formulas
- Often substantially more compact than a traditional normal form
- Can be manipulated very efficiently
  - Conjunction, Disjunction, Negation, Existential Quantification
- R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers, C-35(8), 1986.*

Running Example: Comparator



$\mathbf{f = 1} \Leftrightarrow \quad a_1 = b_1 \ \wedge \ a_2 = b_2$

Conjunctive Normal Form



Not Canonical

Truth Table (1)

a_1	b_1	a_2	b_2	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Still Not Canonical

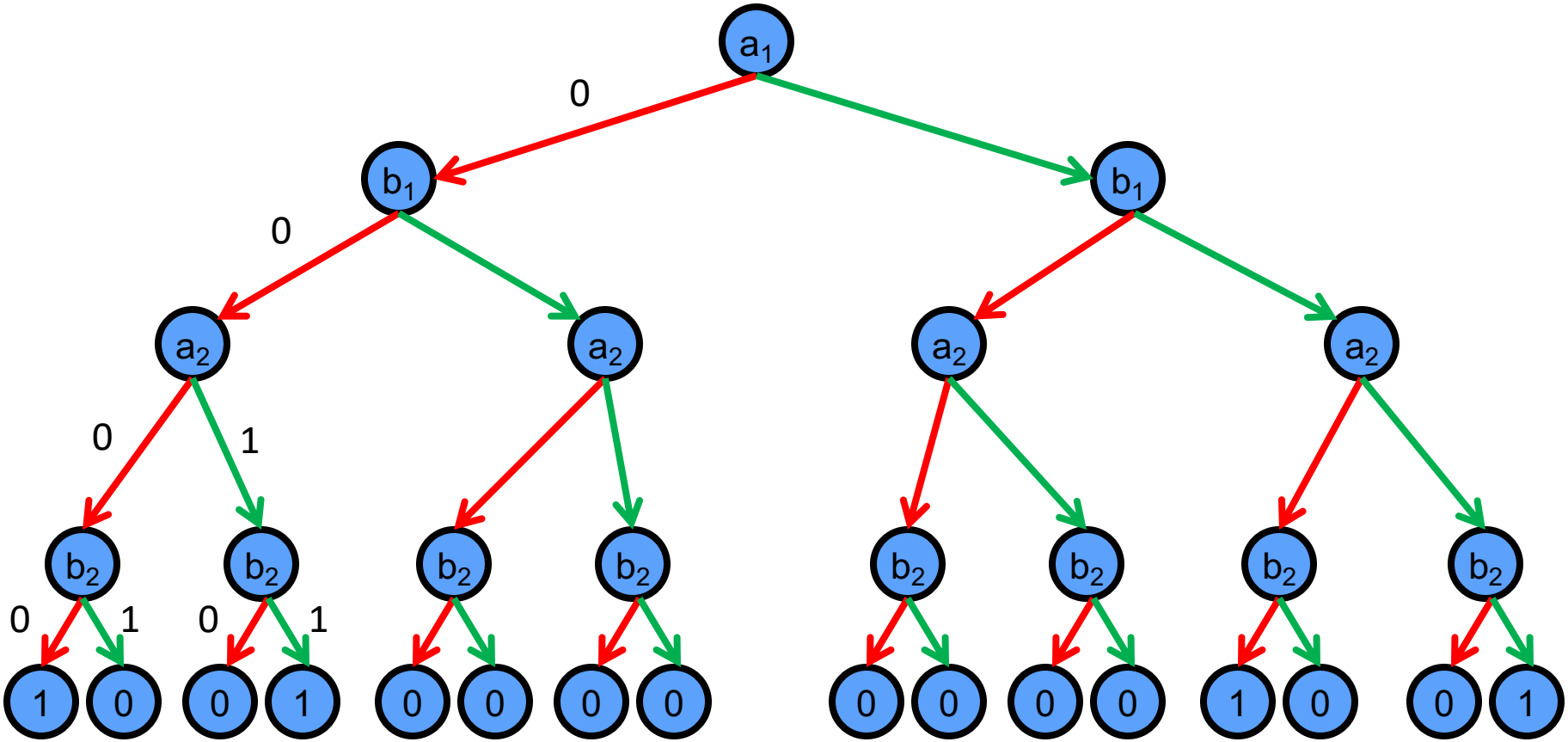
Truth Table (2)

a_1	a_2	b_1	b_2	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Canonical if you fix variable order.

But always exponential in # of variables. Let's try to fix this.

Representing a Truth Table using a Graph

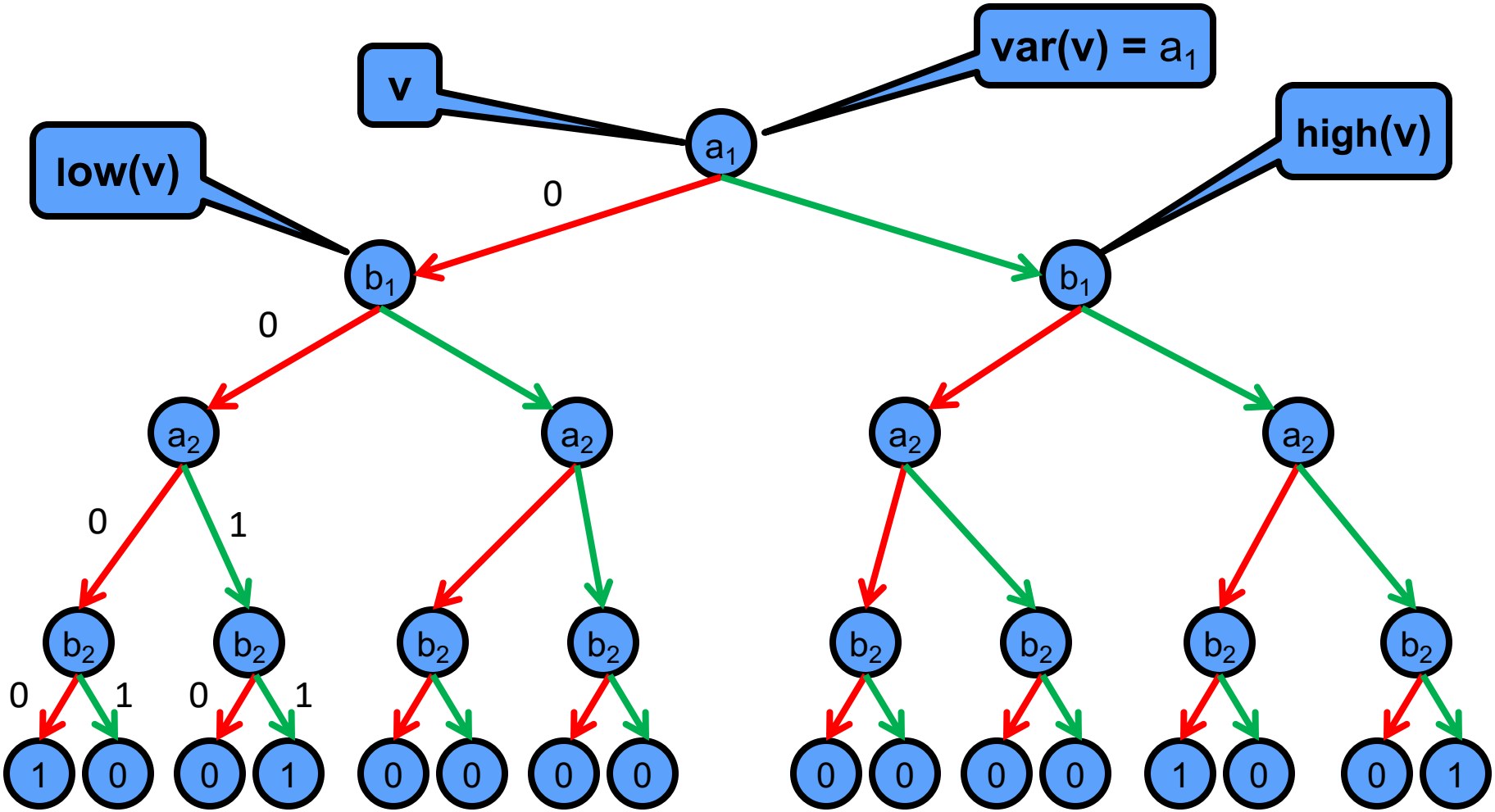


Binary Decision Tree (in this case ordered)

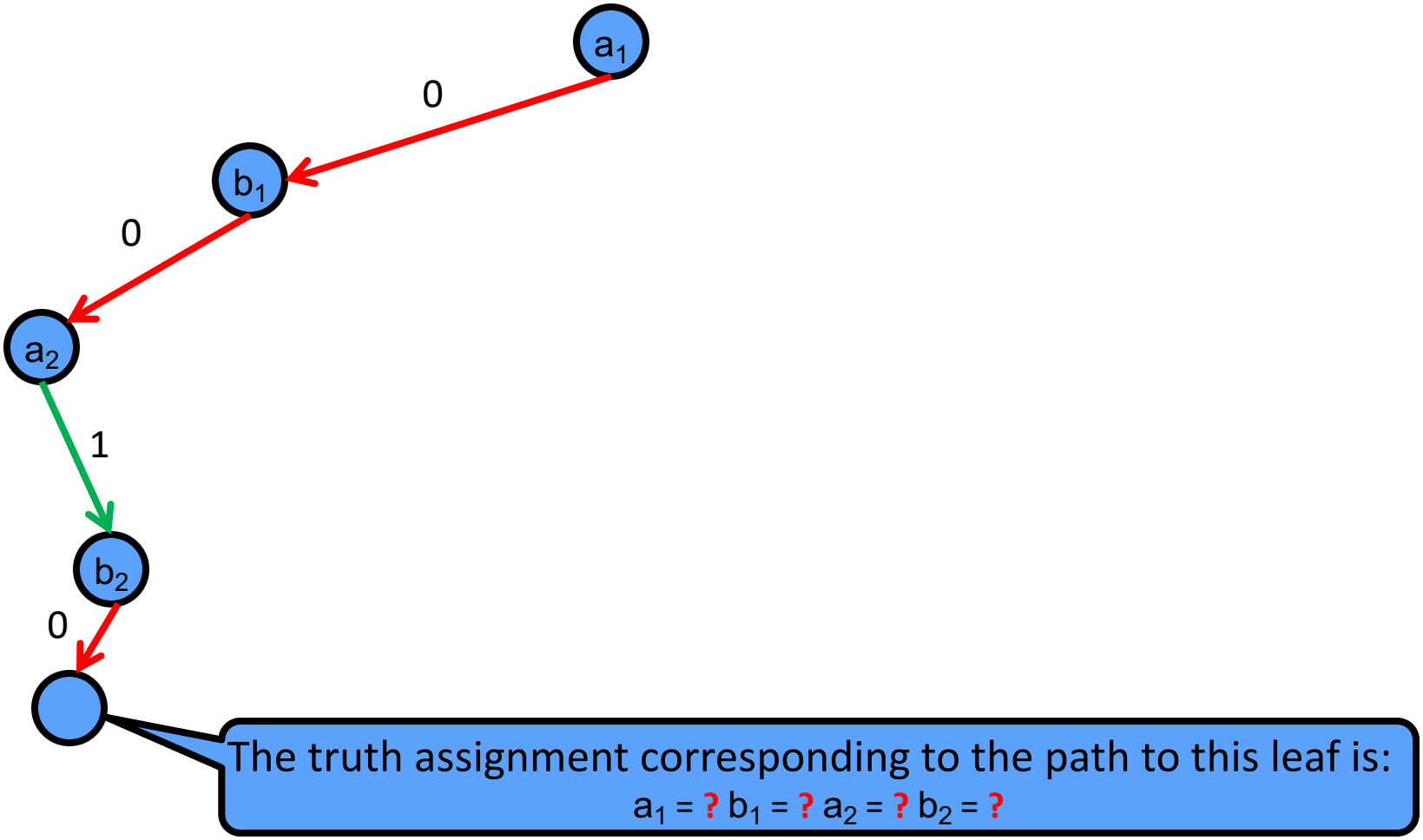
Binary Decision Tree (BDT): Formal Definition

- Balanced binary tree. Length of each path = # of variables
- Leaf nodes labeled with either 0 or 1
- Internal node v labeled with a Boolean variable var(v)
  - Every node on a path labeled with a different variable
- Internal node v has two children: low(v) and high(v)
- Each path corresponds to a (partial) truth assignment to variables
  - Assign 0 to var(v) if low(v) is in the path, and 1 if high(v) is in the path
- Value of a leaf is determined by:
  - Constructing the truth assignment for the path leading to it from the root
  - Looking up the truth table with this truth assignment
- A BDT is *ordered* if the variables appear in the same order along each path.

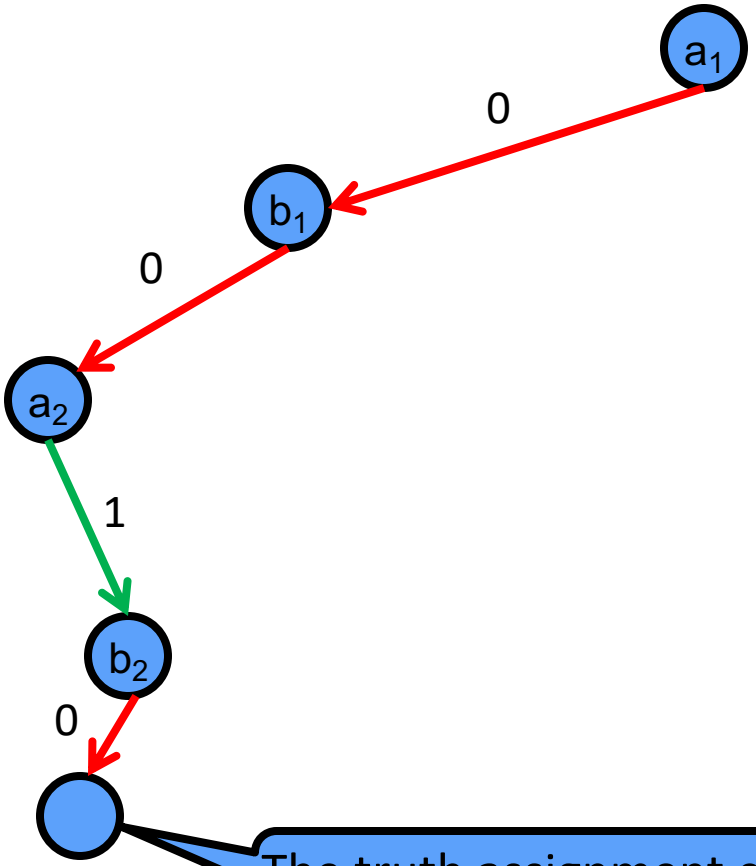
Binary Decision Tree



Binary Decision Tree



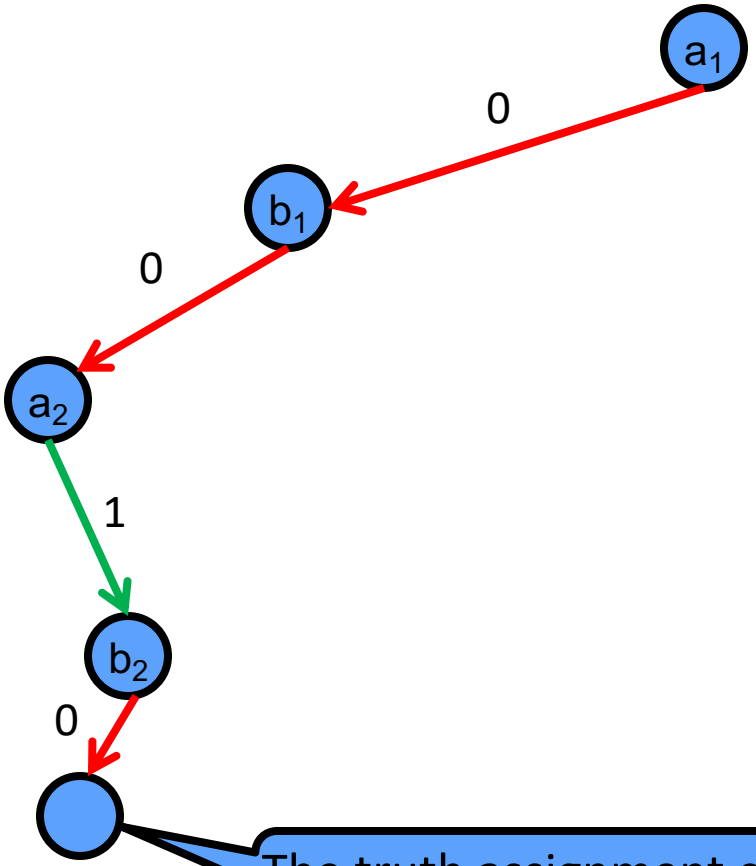
Binary Decision Tree



a_1	b_1	a_2	b_2	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

The truth assignment corresponding to the path to this leaf is:
$a_1 = 0\ b_1 = 0\ a_2 = 1\ b_2 = 0$

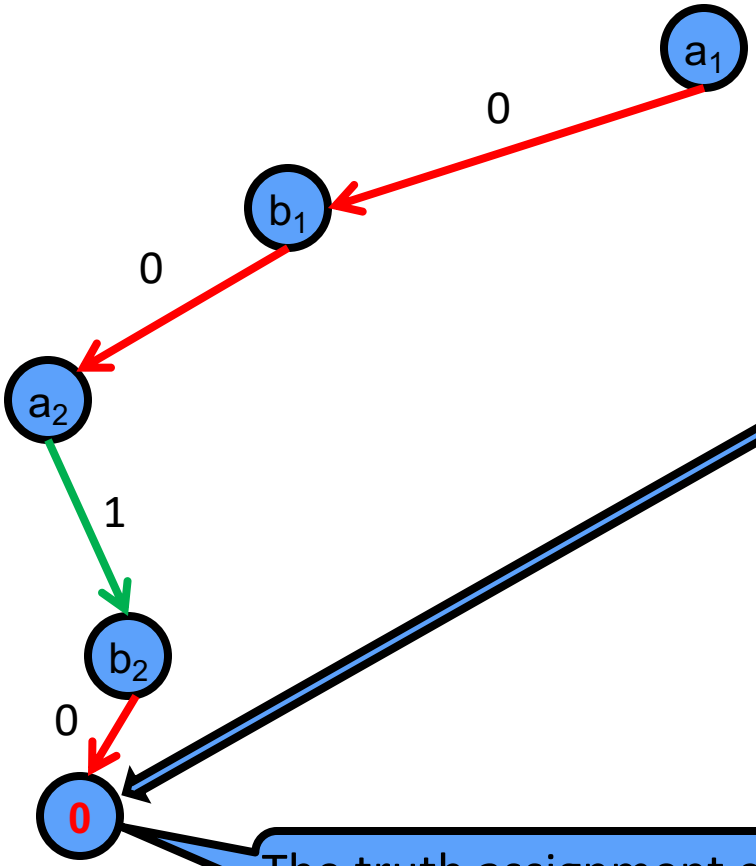
Binary Decision Tree



a_1	b_1	a_2	b_2	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

The truth assignment corresponding to the path to this leaf is:
$a_1 = 0\ b_1 = 0\ a_2 = 1\ b_2 = 0$

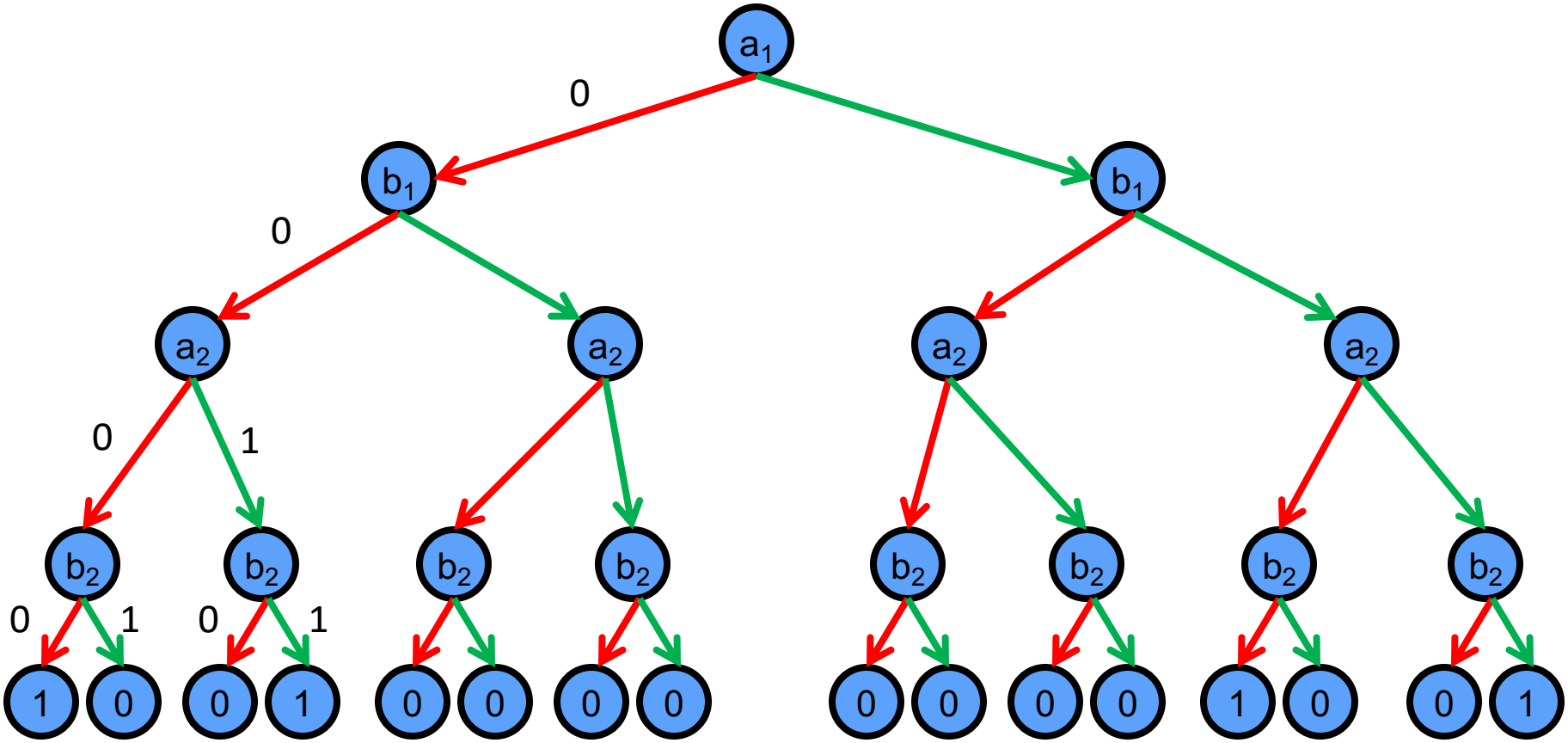
Binary Decision Tree



a_1	b_1	a_2	b_2	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

The truth assignment corresponding to the path to this leaf is:
$a_1 = 0\ b_1 = 0\ a_2 = 1\ b_2 = 0$

Binary Decision Tree (BDT)



Canonical if you fix variable order (i.e., use ordered BDT)

But still exponential in # of variables. Let's try to fix this.

Reduced Ordered BDD

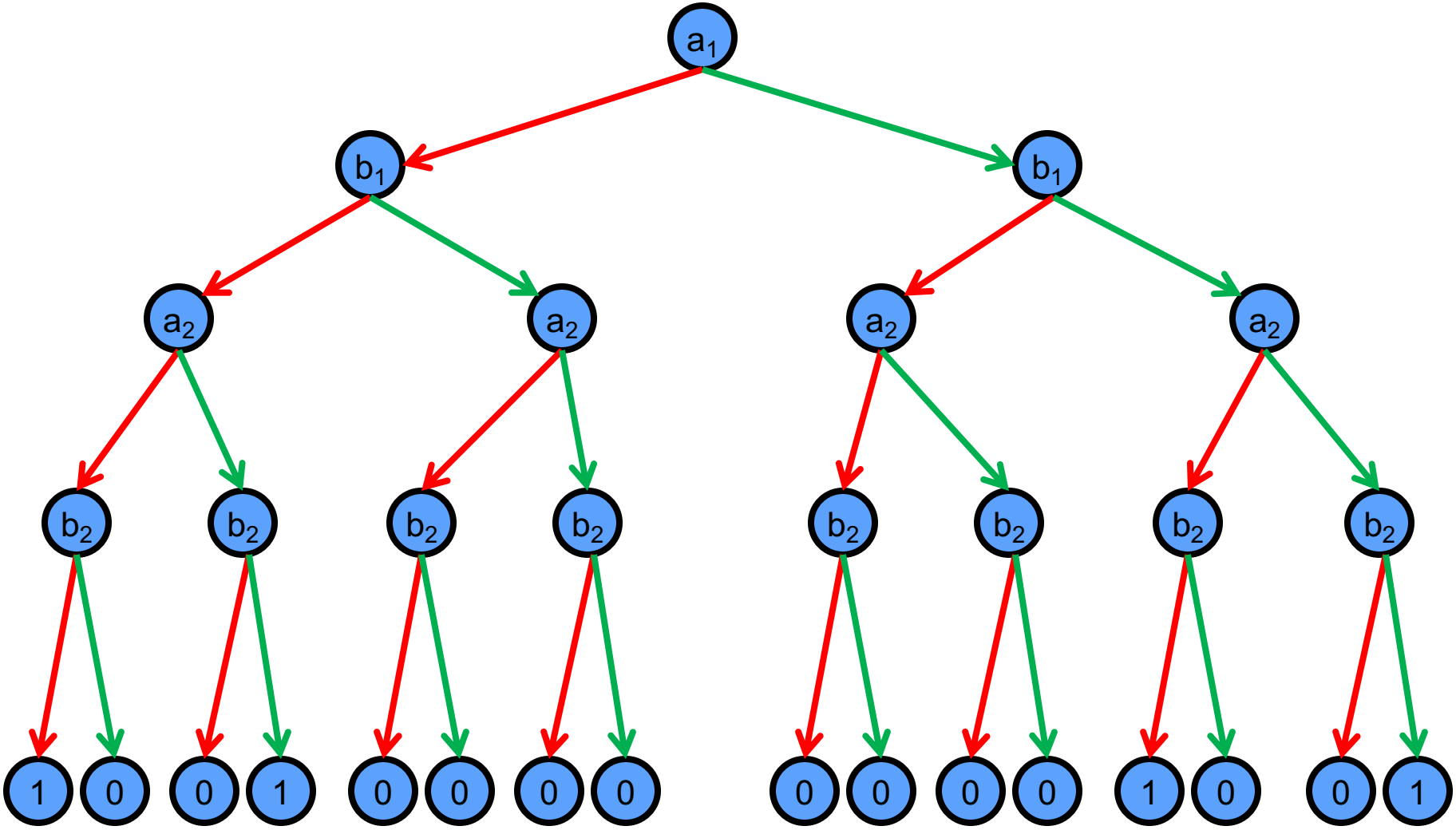
Reduced Ordered BDD

- Conceptually, a ROBDD is obtained from an ordered BDT (OBDT) by eliminating redundant nodes/sub-diagrams

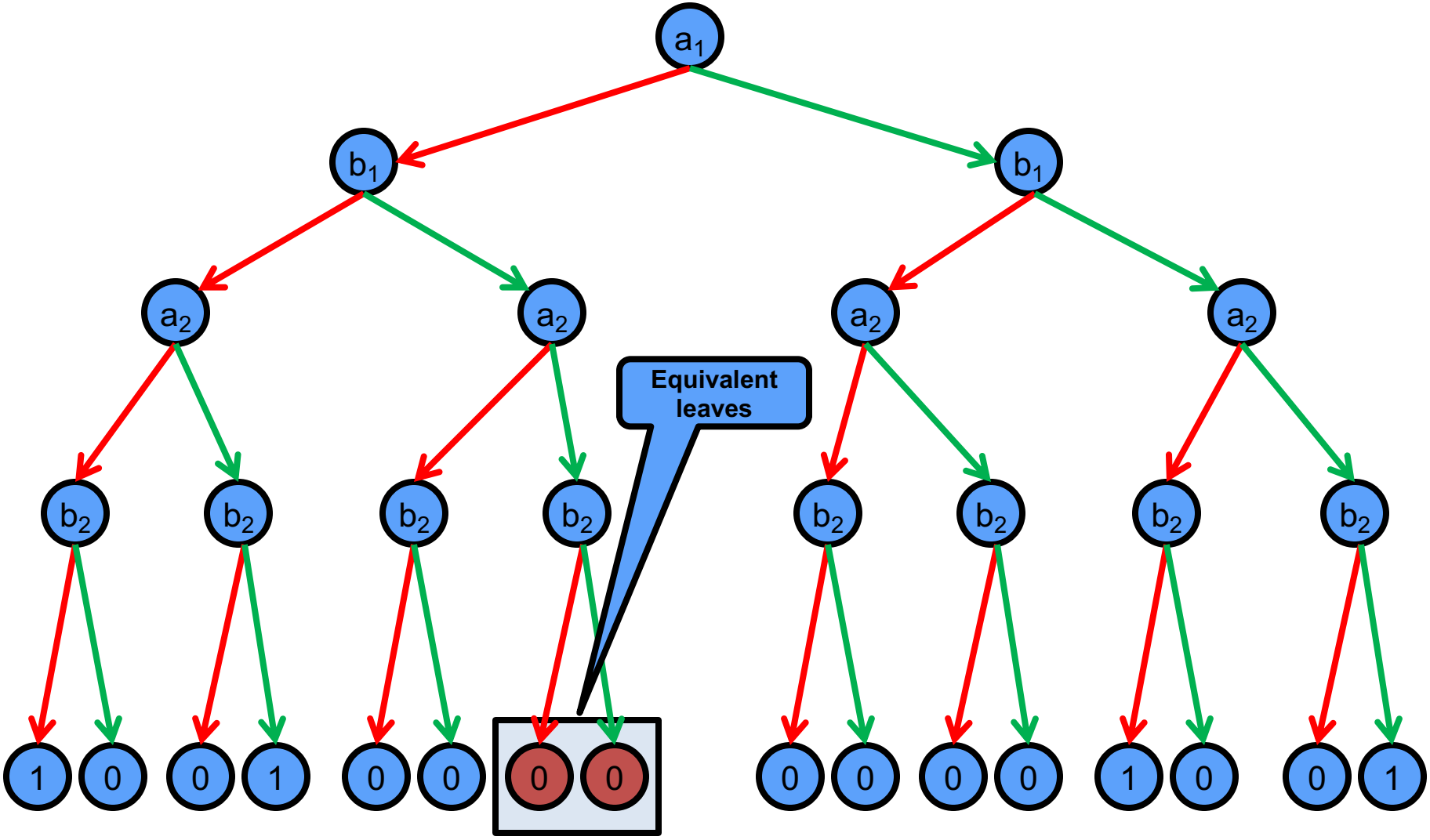
- Start with OBDT and repeatedly apply the following three operations as long as possible:
  1. Merge equivalent leaves.
  2. Merge isomorphic nodes.
  3. Eliminate redundant tests. Whenever low(v) = high(v), remove v and redirect edges into v to low(v).
  - Why does this terminate?

- ROBDD is often exponentially smaller than the corresponding OBDT

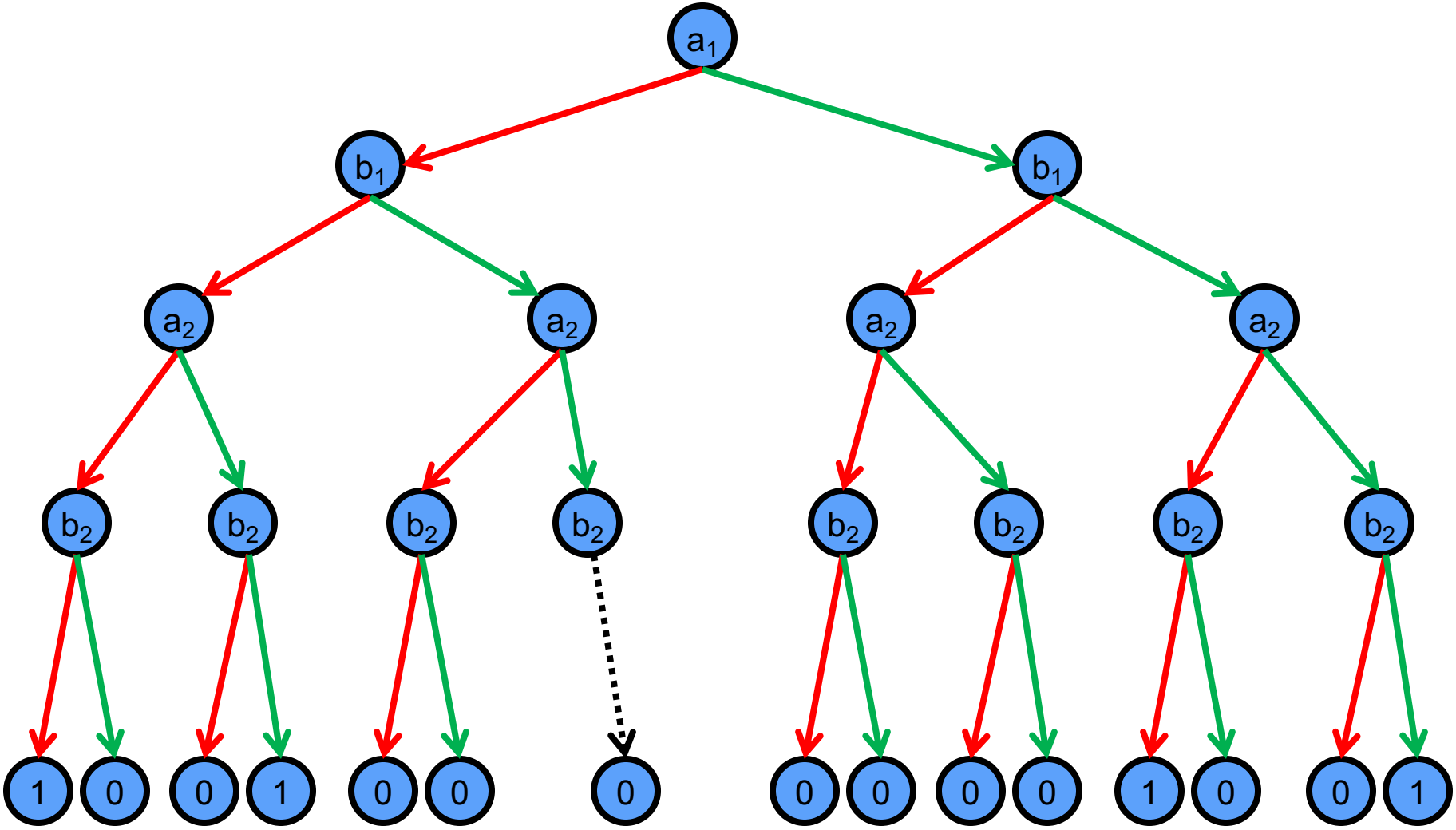
OBDT to ROBDD



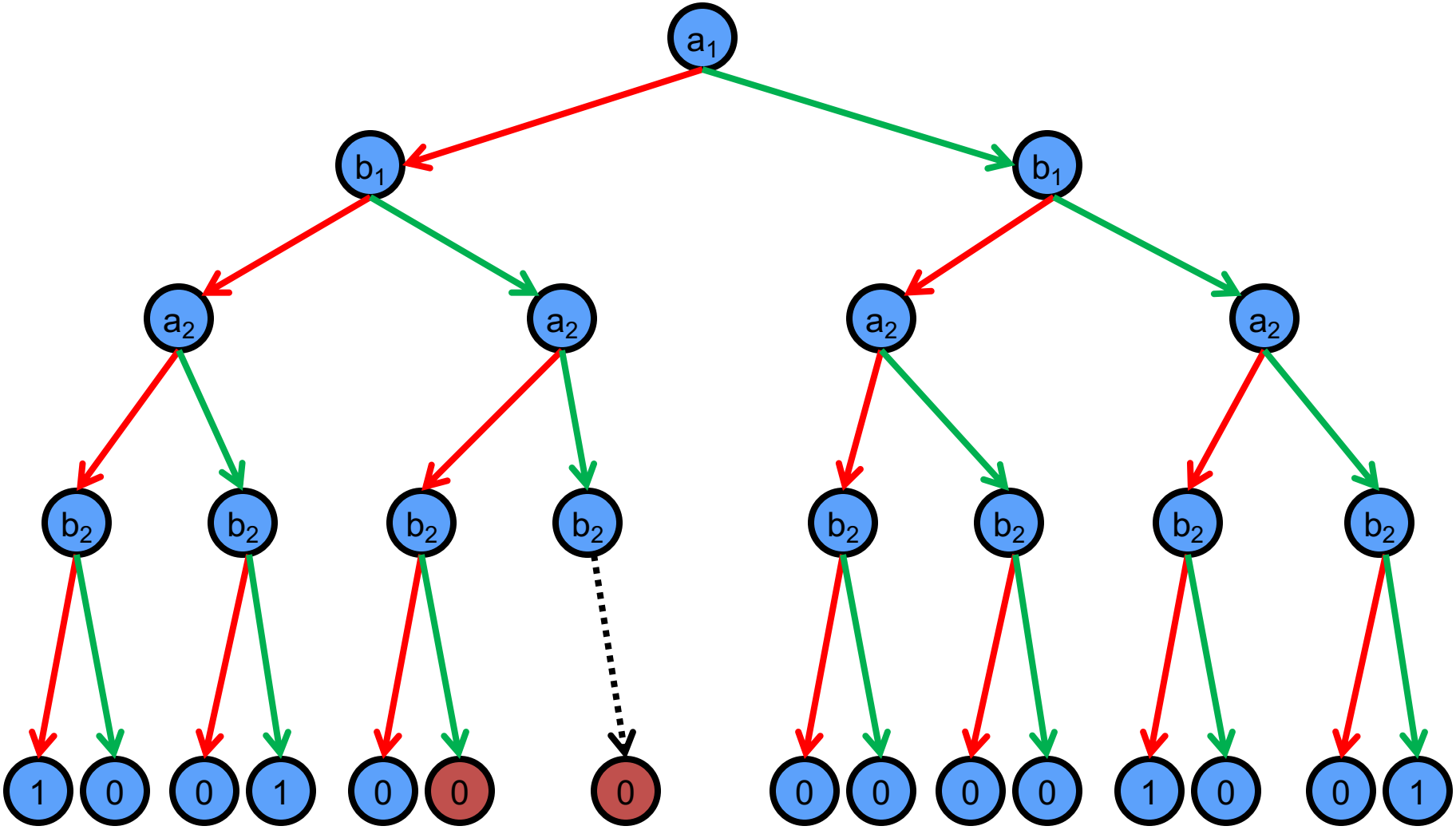
OBDT to ROBDD



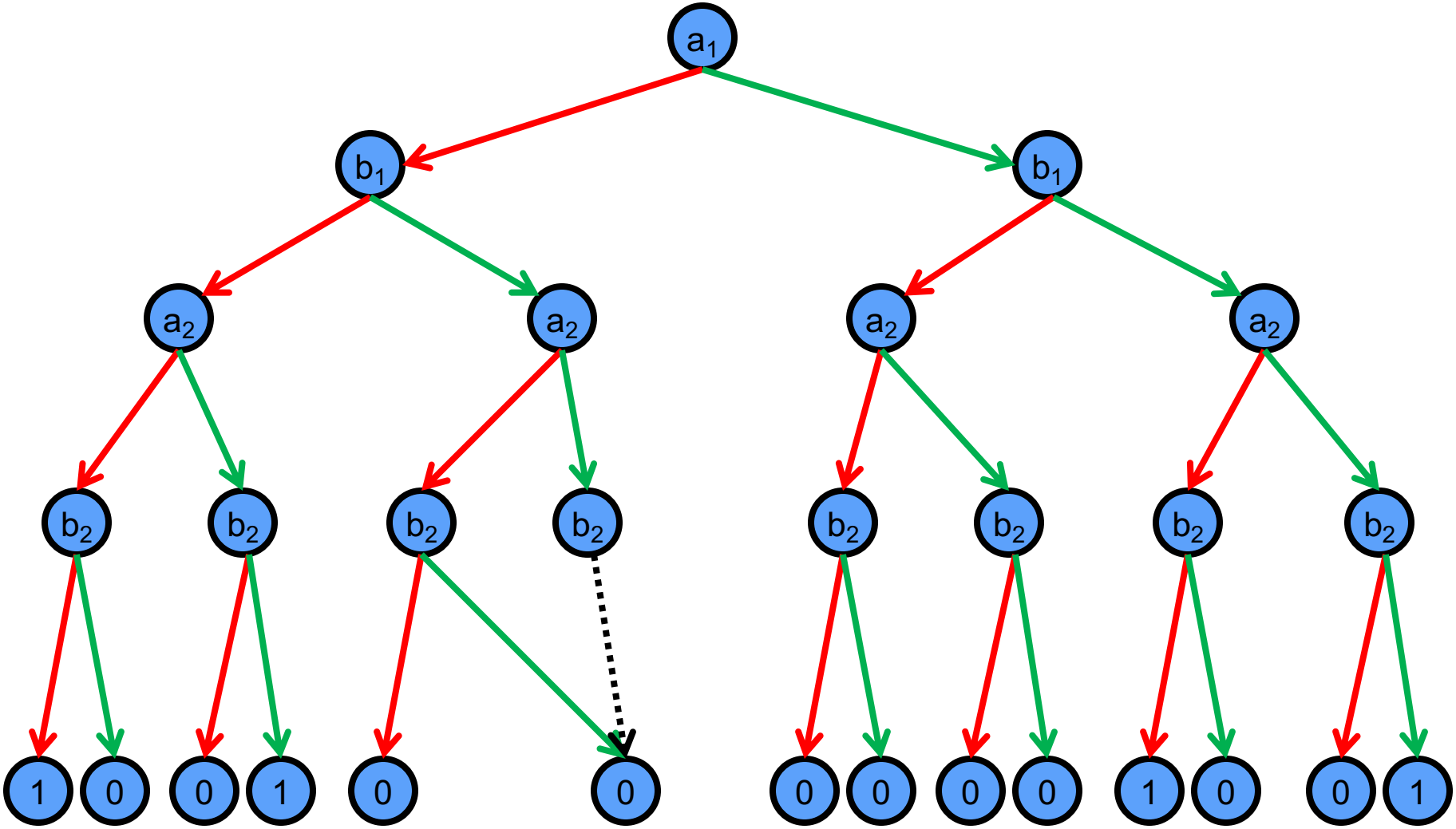
OBDT to ROBDD



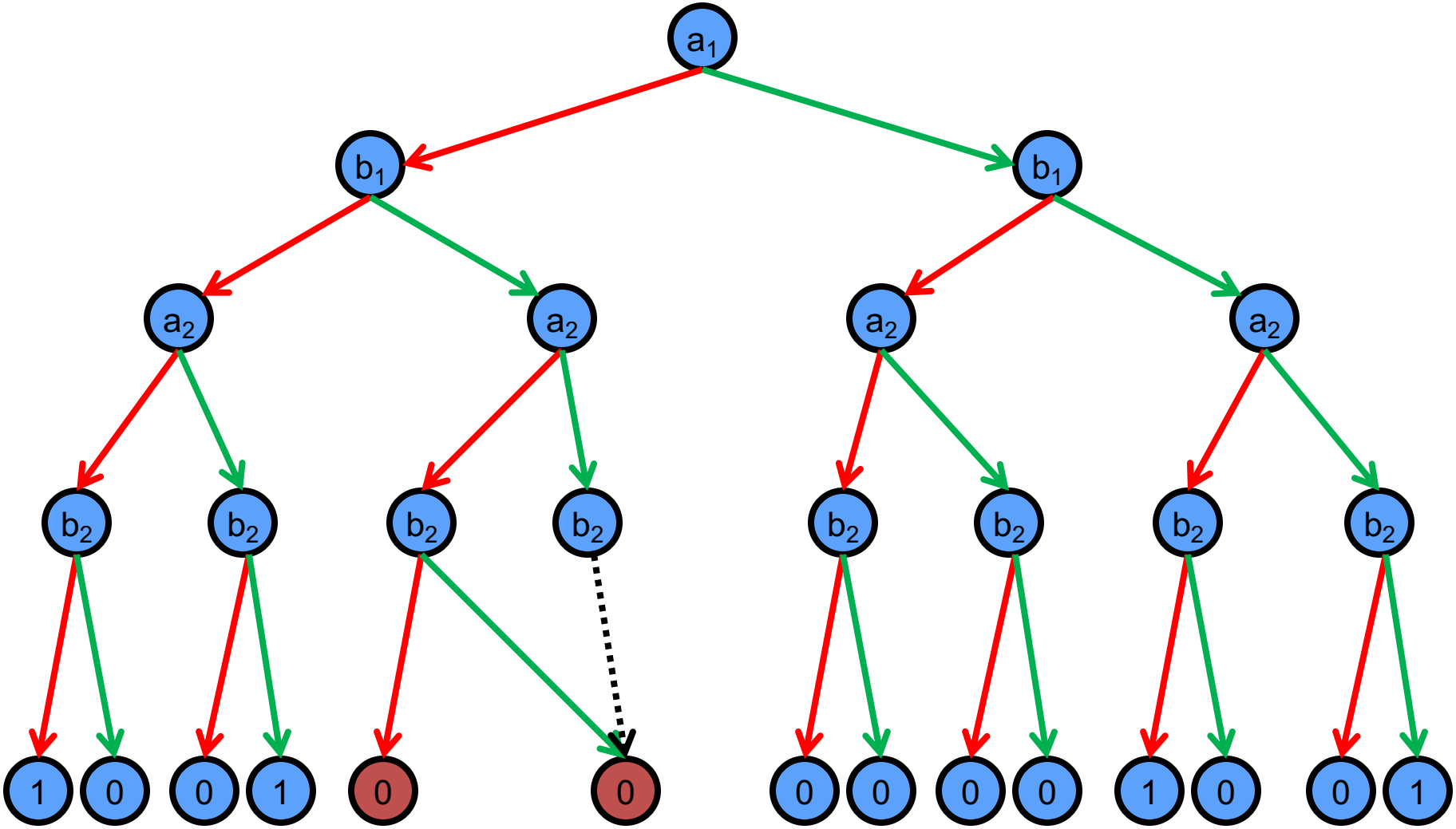
OBDT to ROBDD



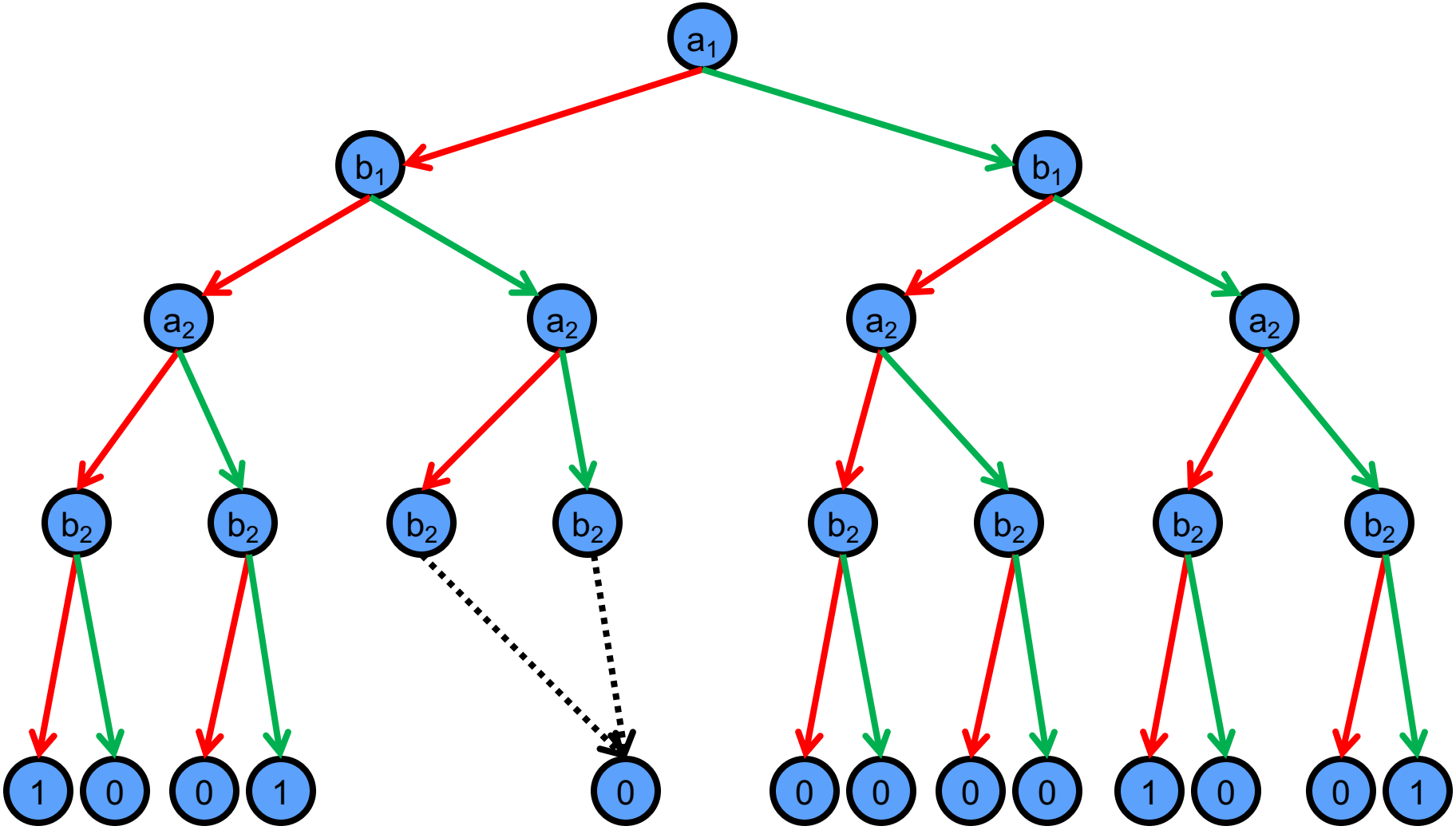
OBDT to ROBDD



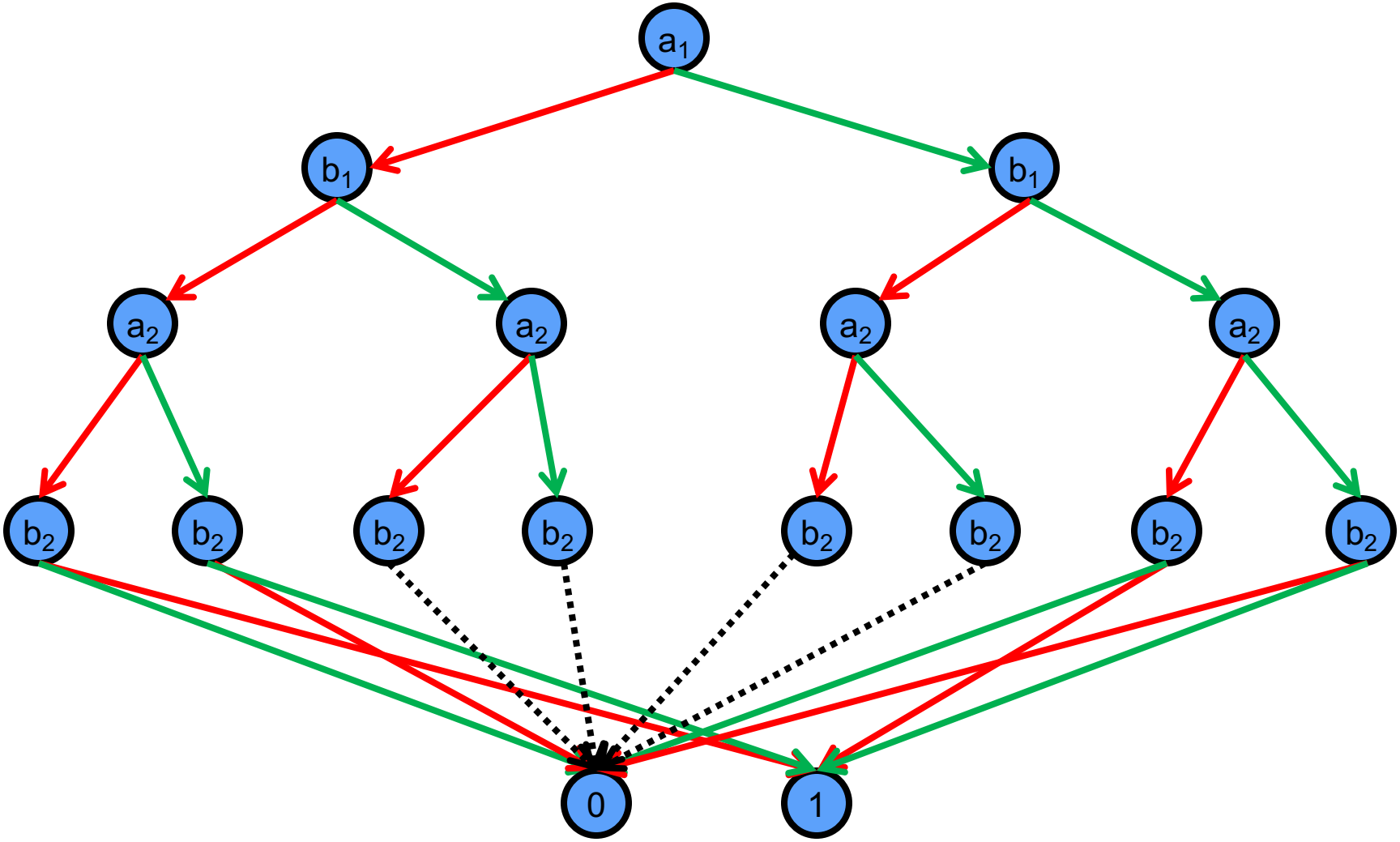
OBDT to ROBDD



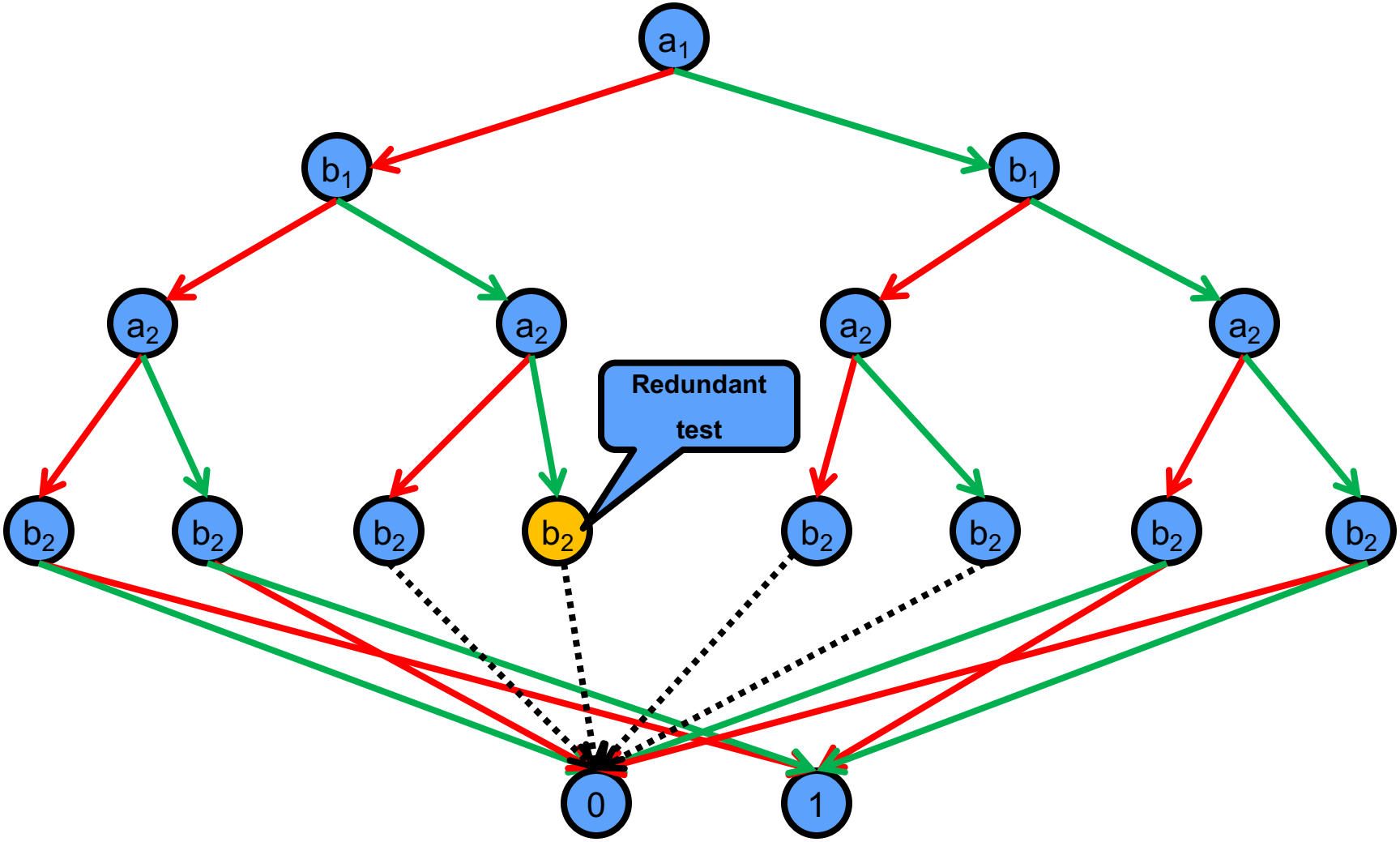
OBDT to ROBDD



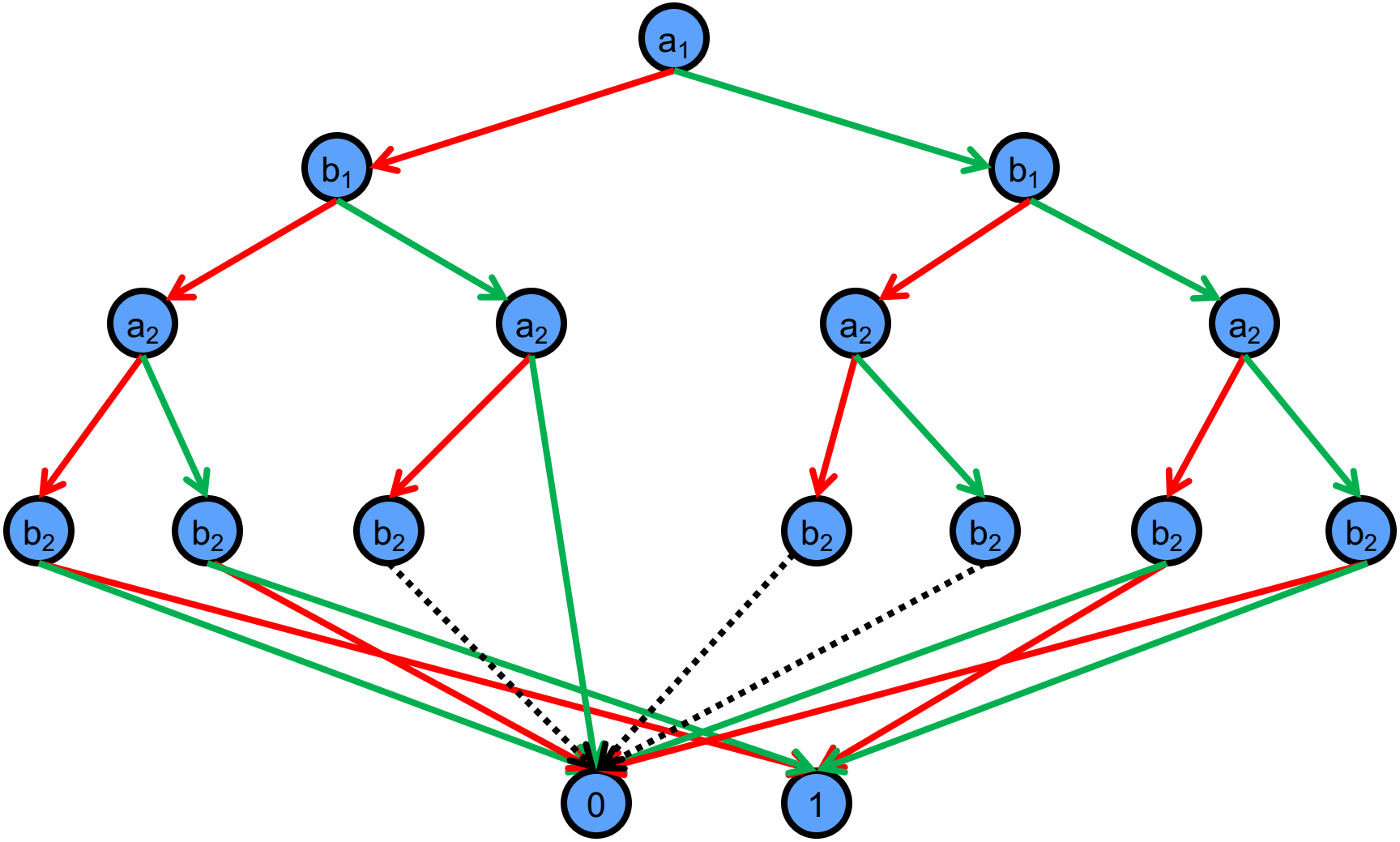
OBDT to ROBDD



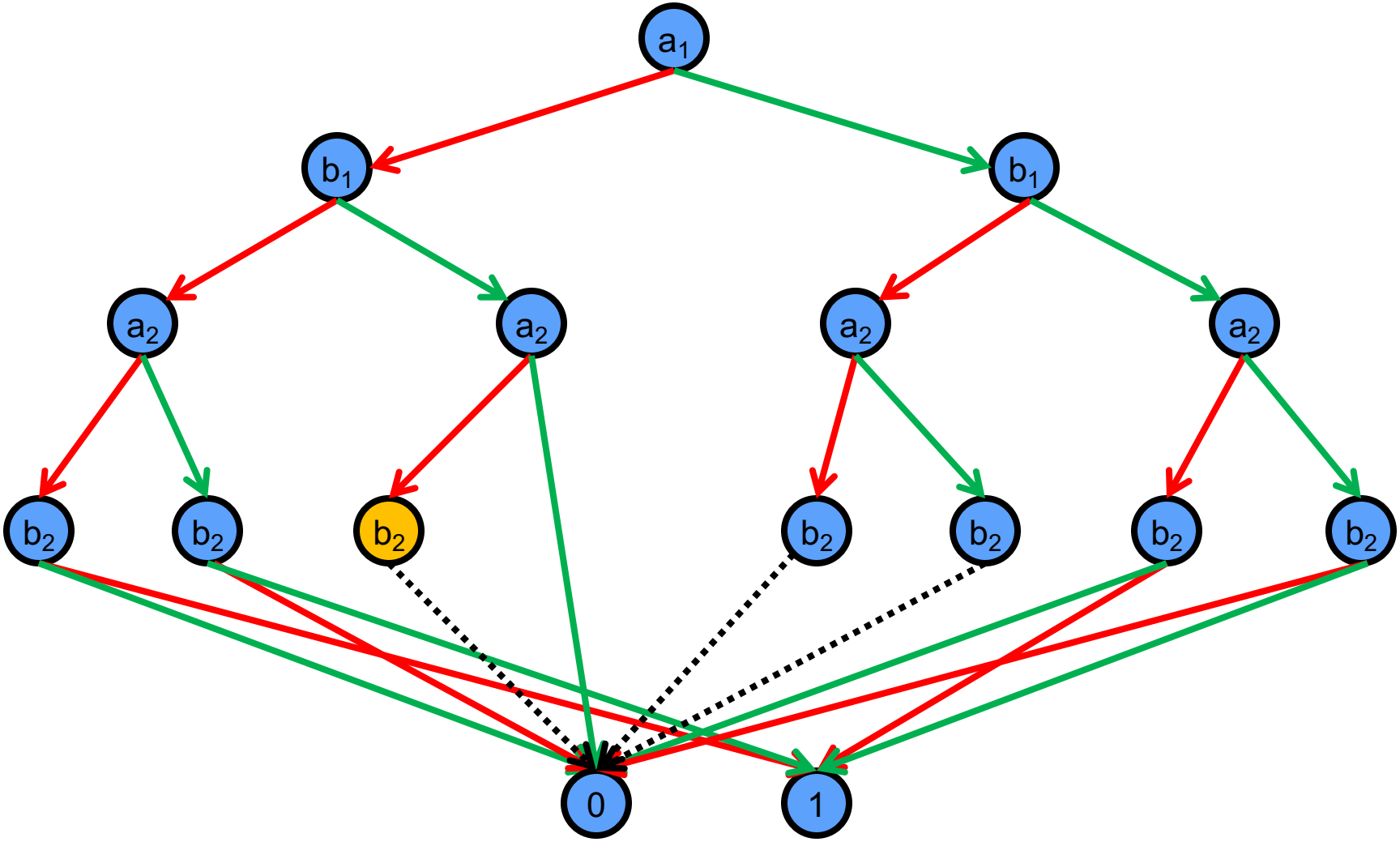
OBDT to ROBDD



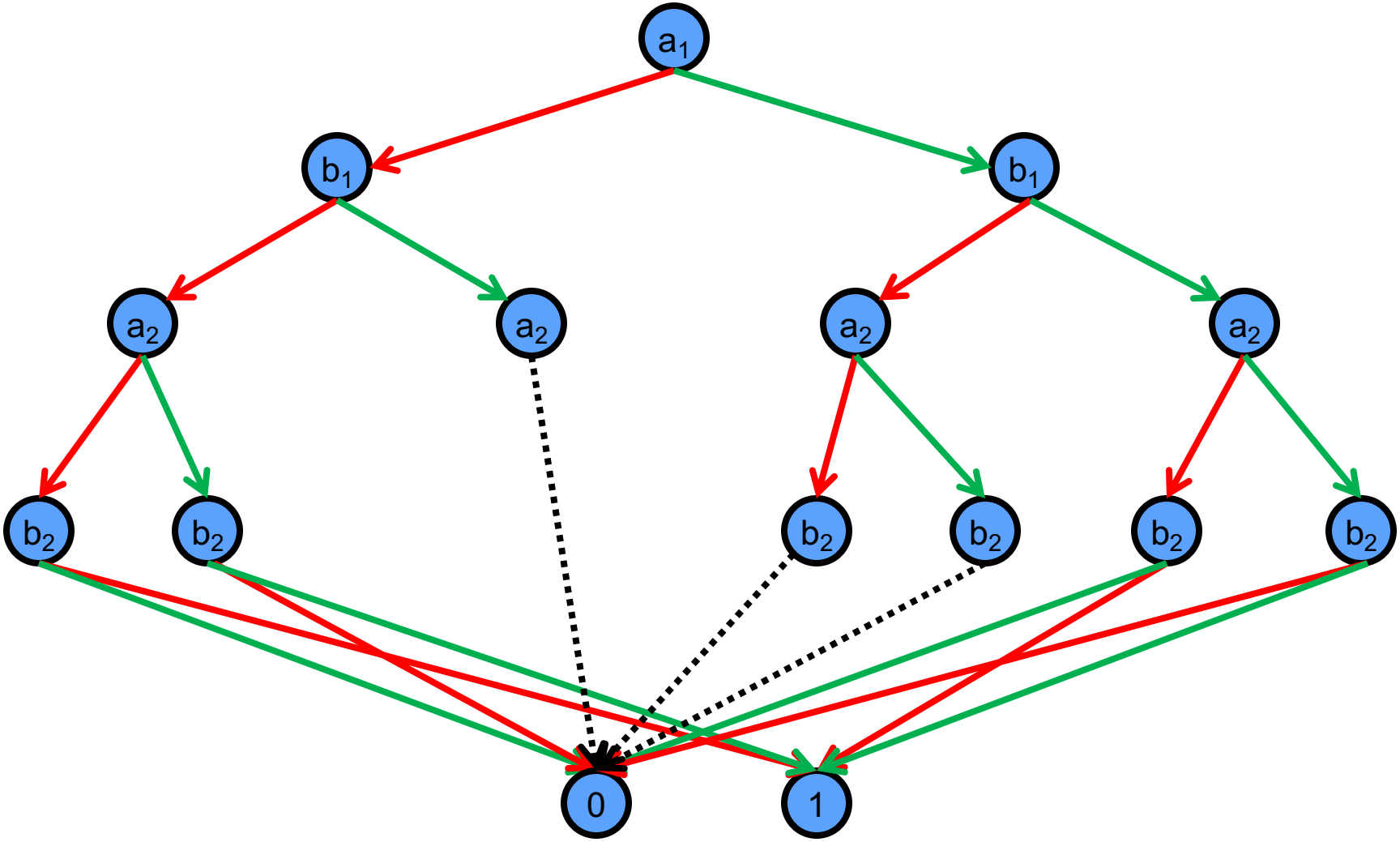
OBDT to ROBDD



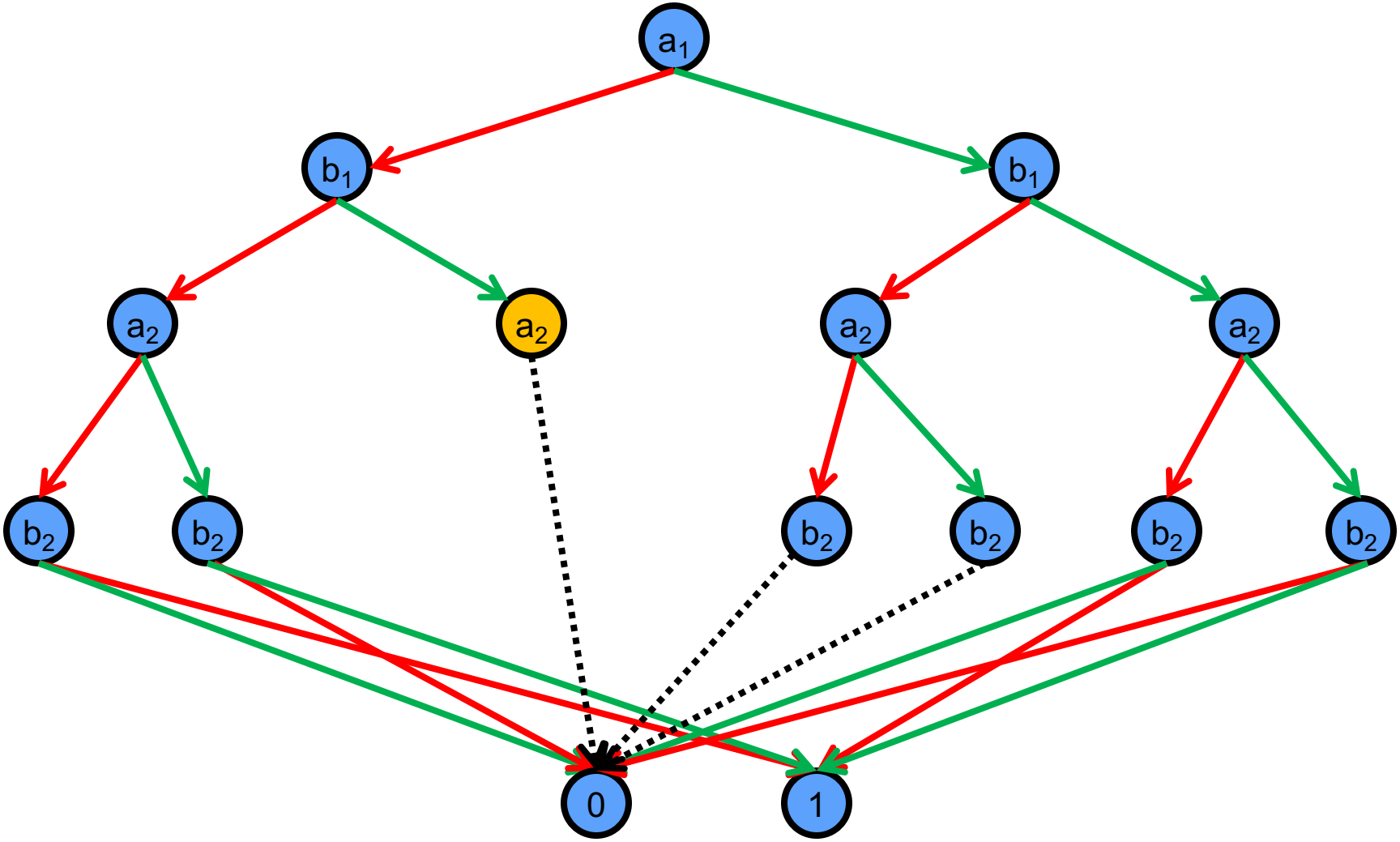
OBDT to ROBDD



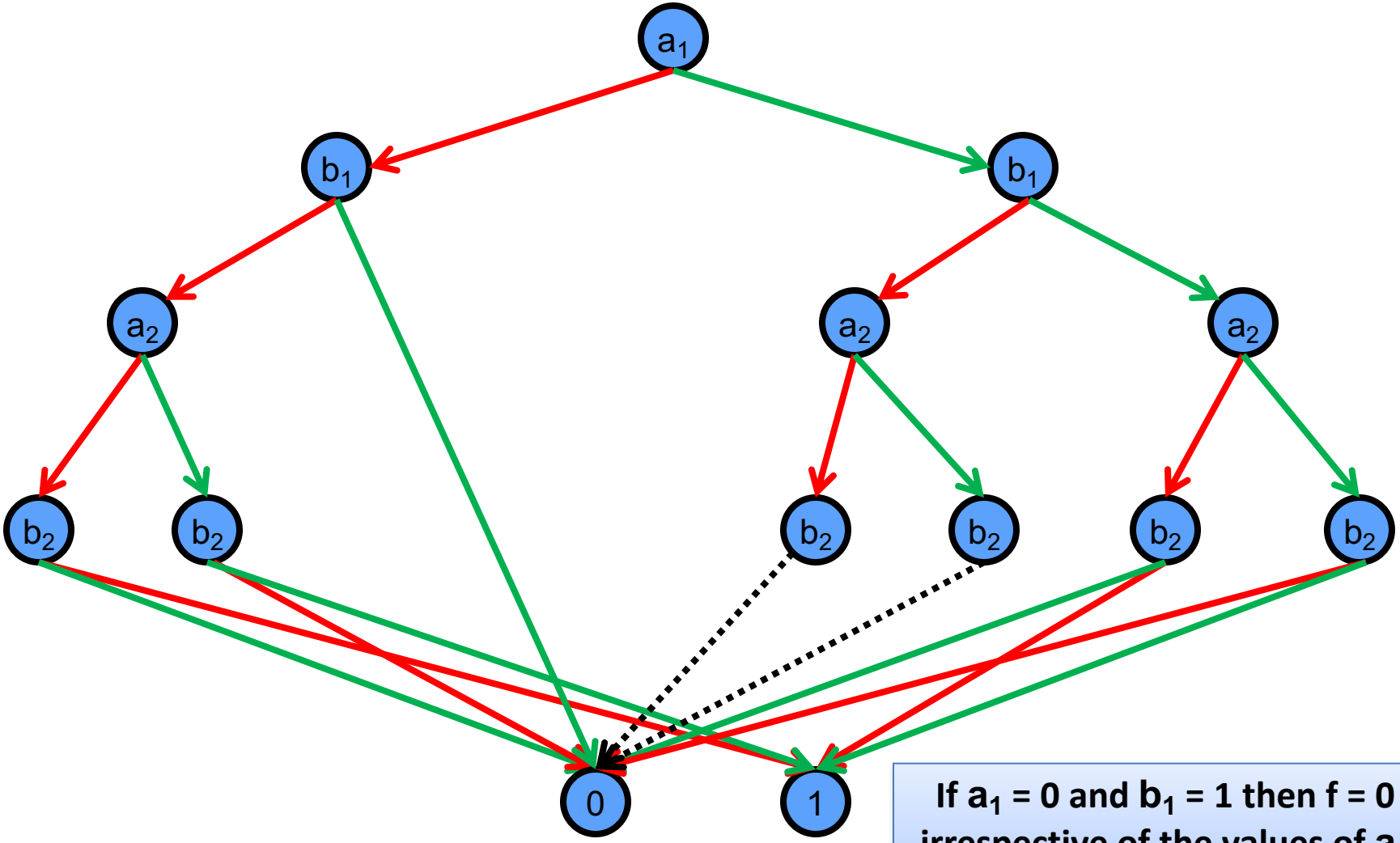
OBDT to ROBDD



OBDT to ROBDD

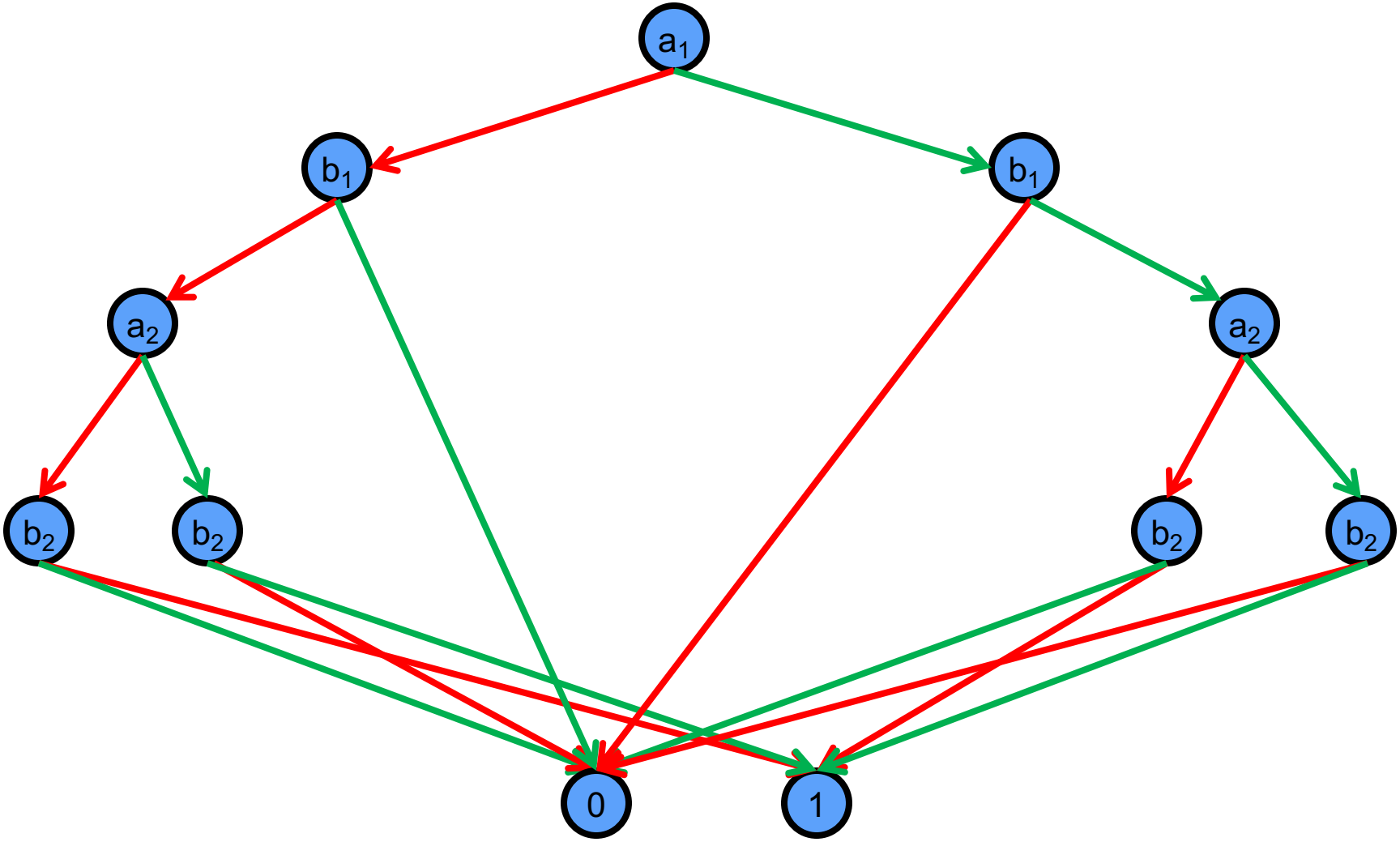


OBDT to ROBDD

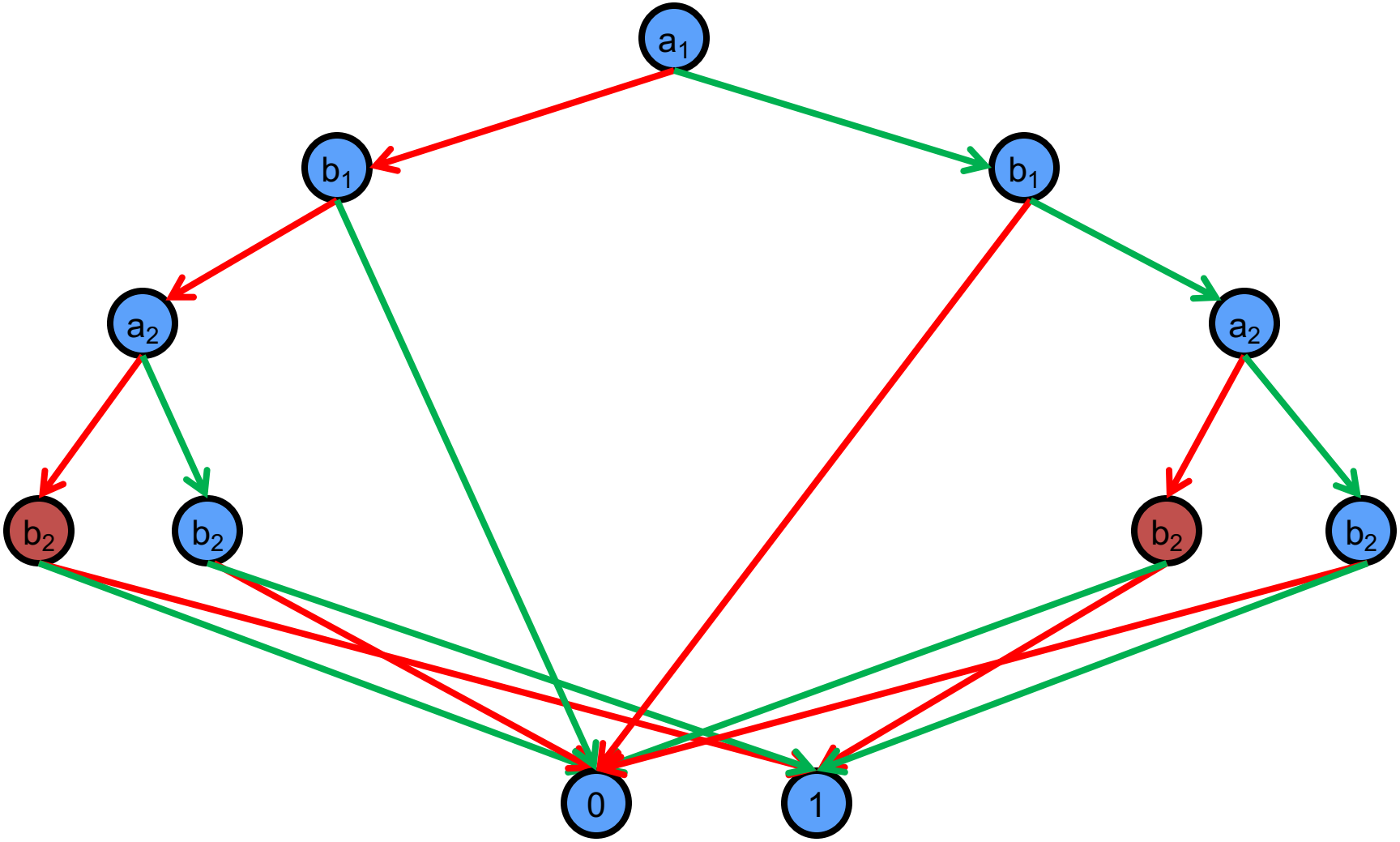


If $a_1 = 0$ and $b_1 = 1$ then f = 0 irrespective of the values of a_2 and b_2

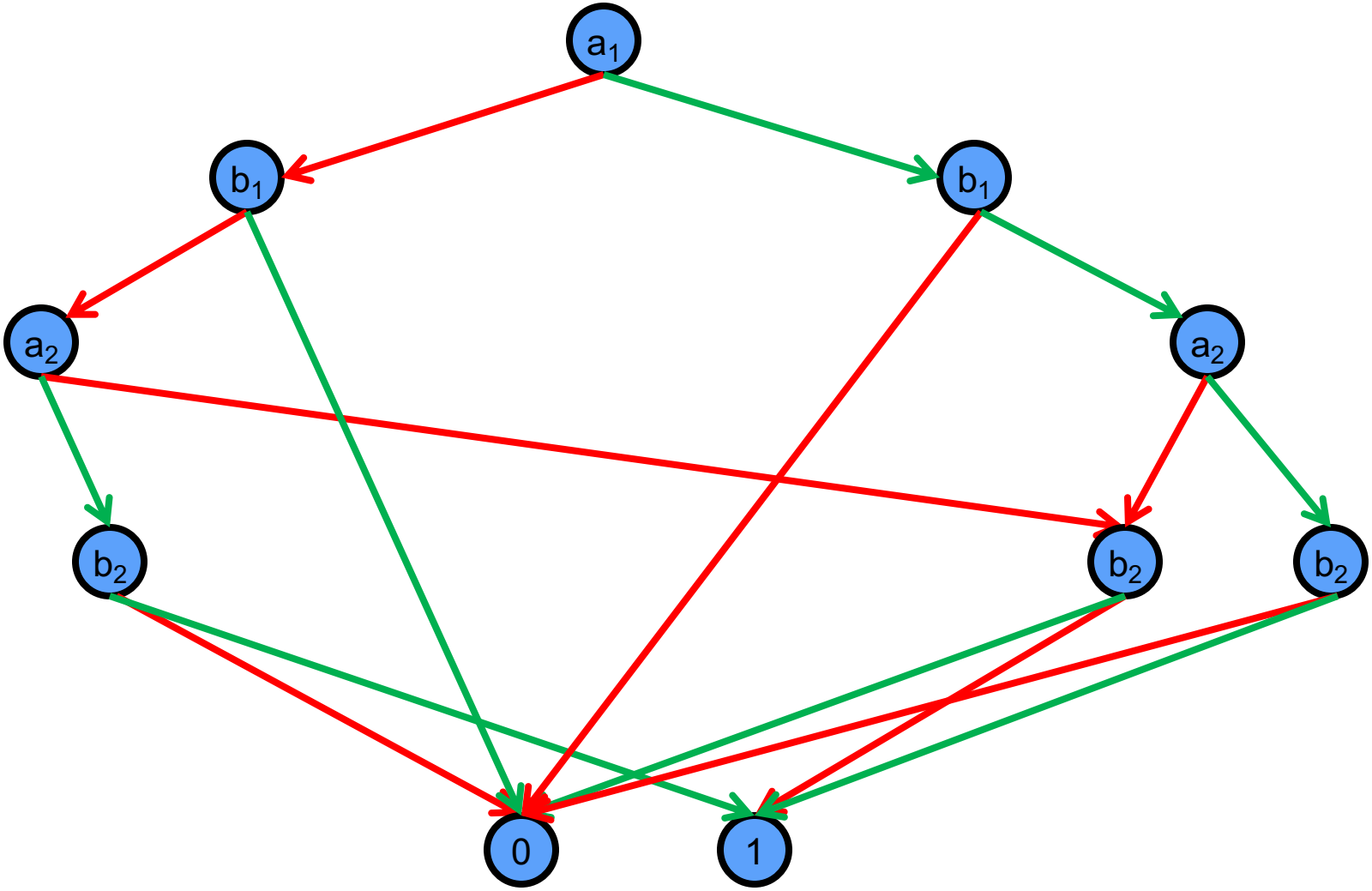
OBDT to ROBDD



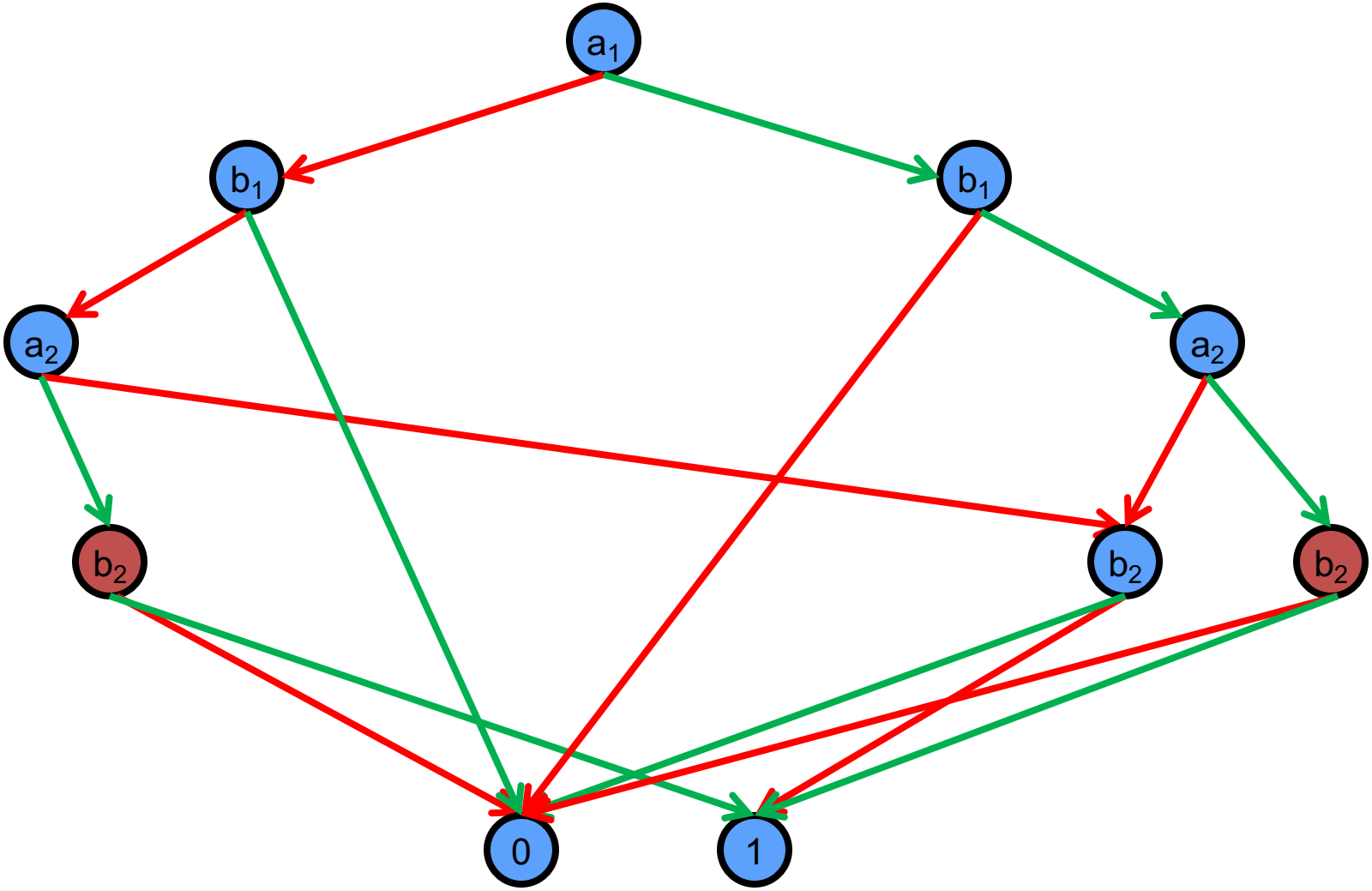
OBDT to ROBDD



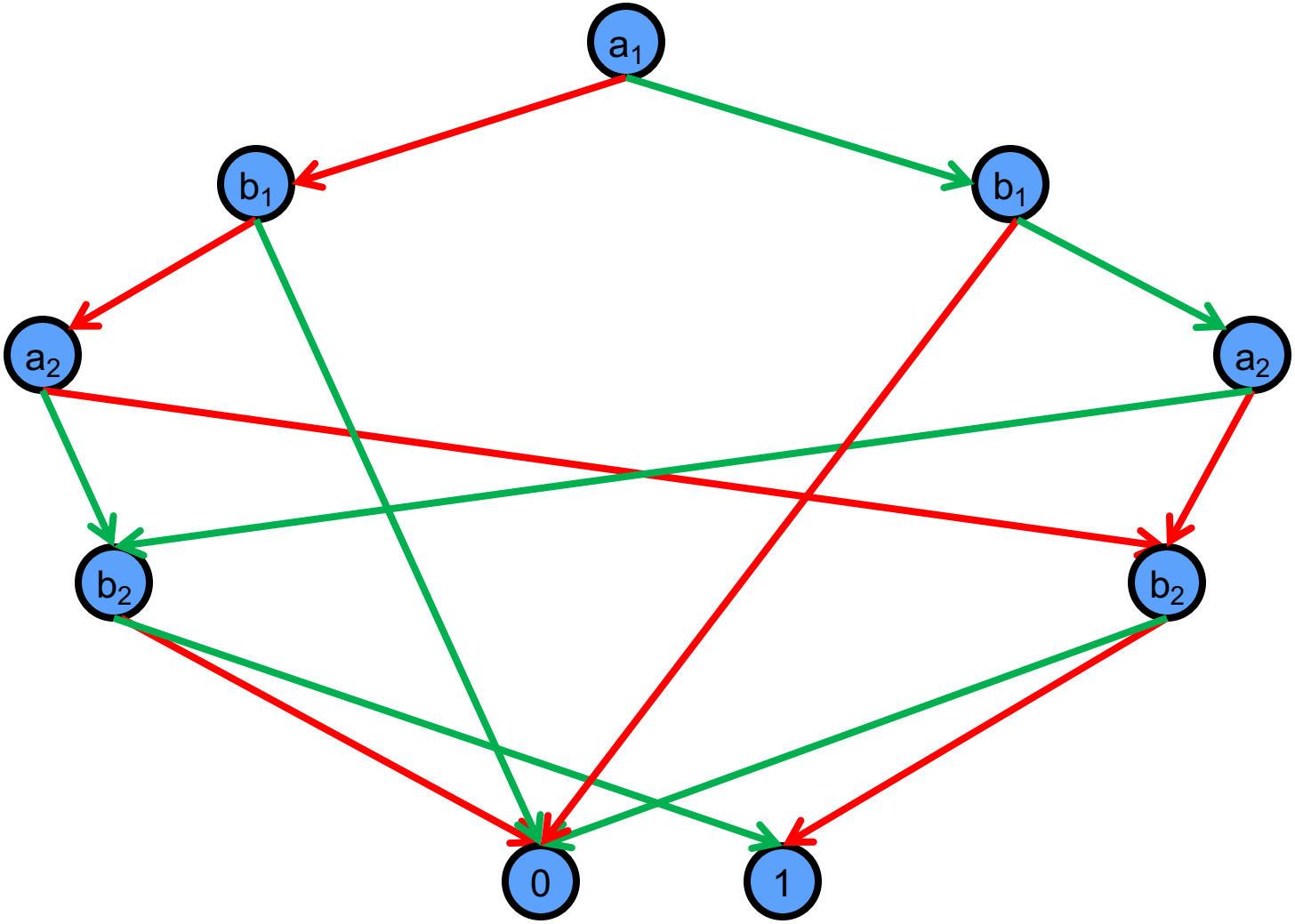
OBDT to ROBDD



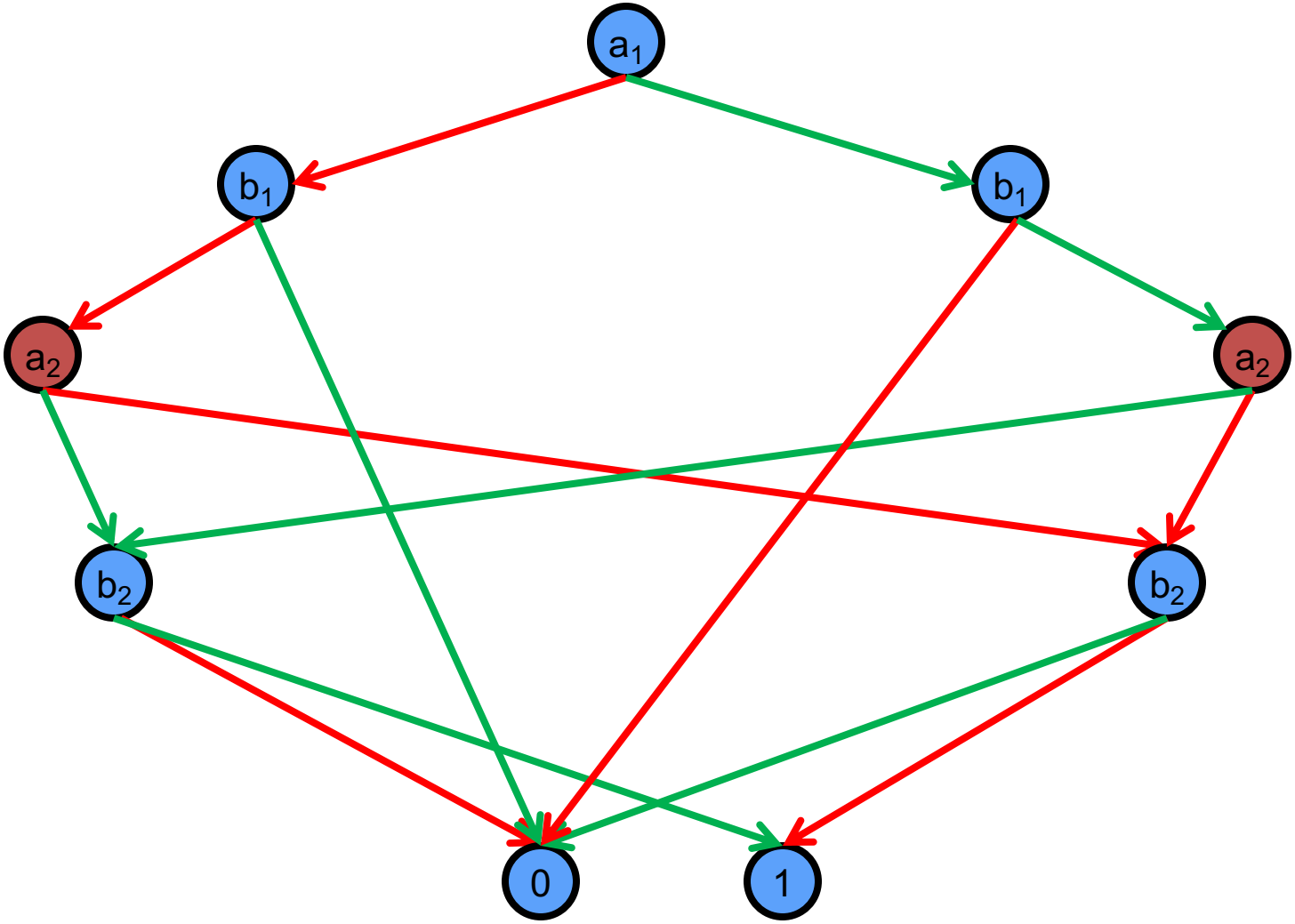
OBDT to ROBDD



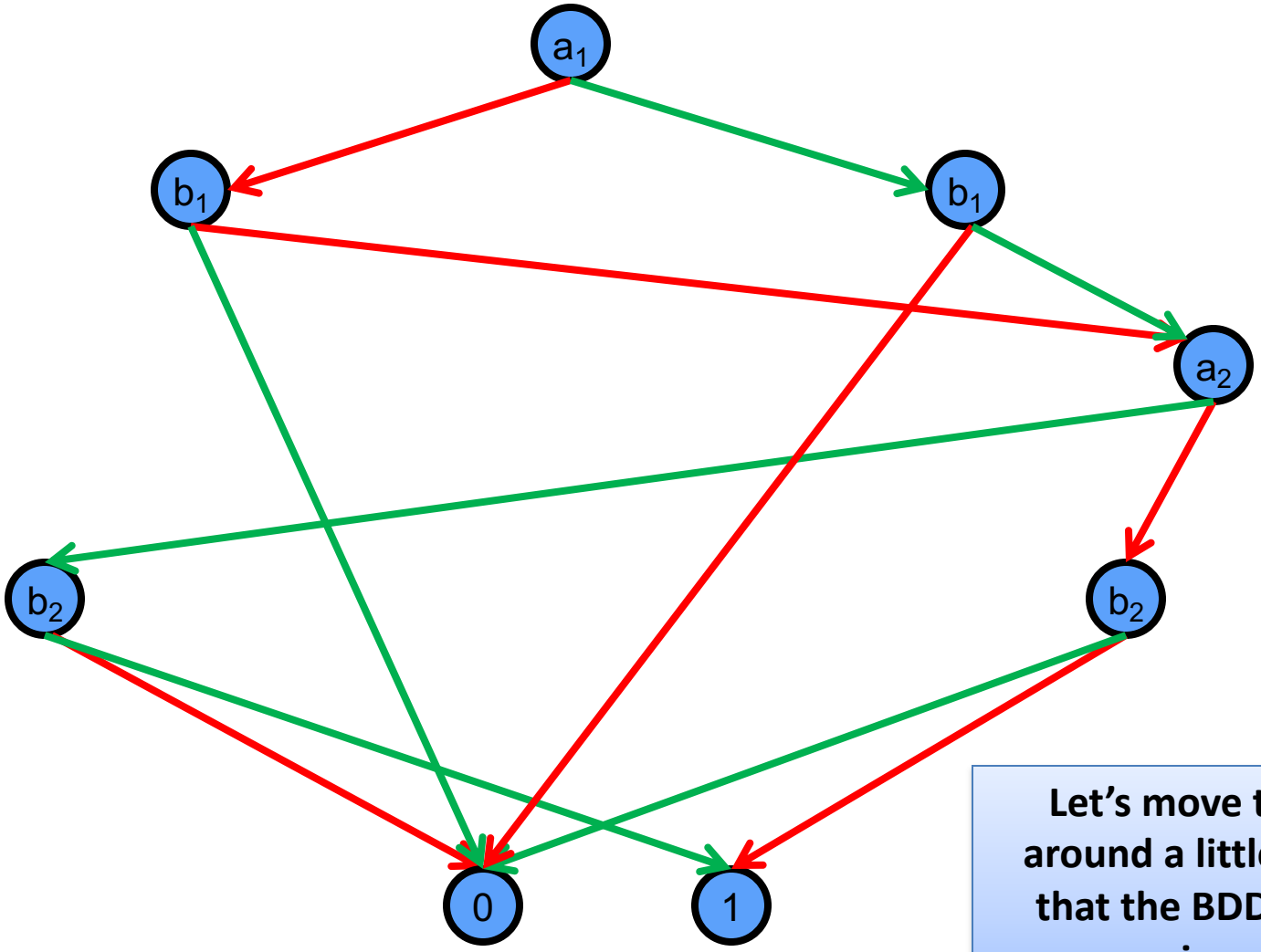
OBDT to ROBDD



OBDT to ROBDD

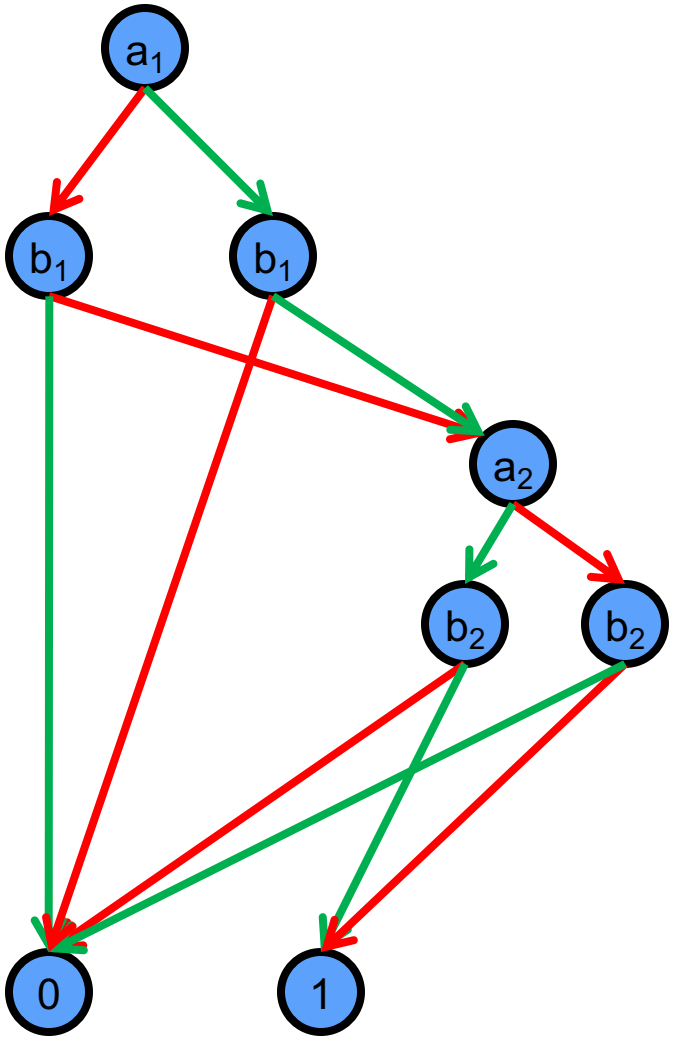


OBDT to ROBDD



Let's move things around a little bit so that the BDD looks nicer.

OBDT to ROBDD

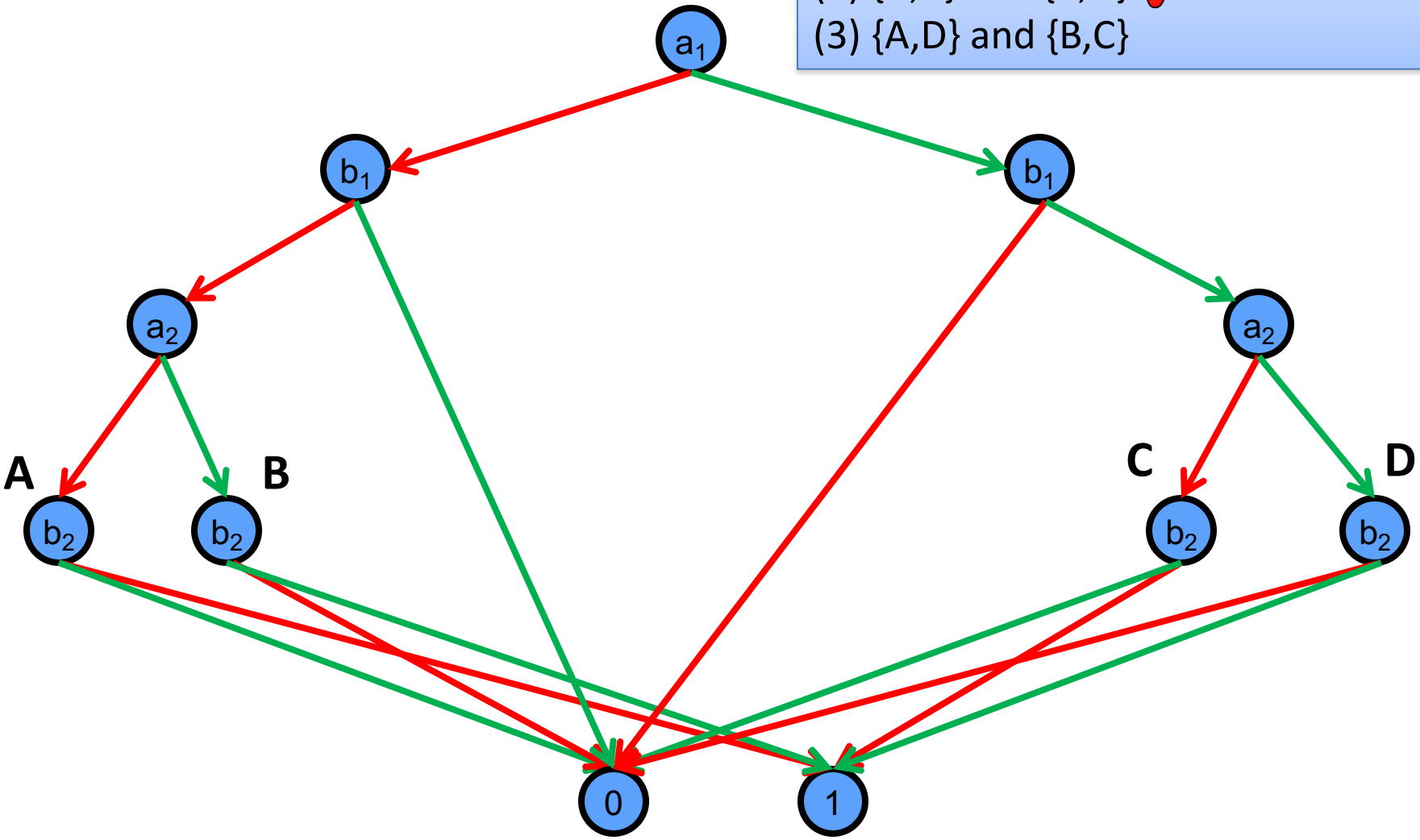


Bryant gave a linear-time algorithm (called Reduce) to convert OBDT to ROBDD.

In practice, BDD packages don't use Reduce directly. They apply the reductions on-the-fly as new BDDs are constructed from existing ones.

Exercise 1:

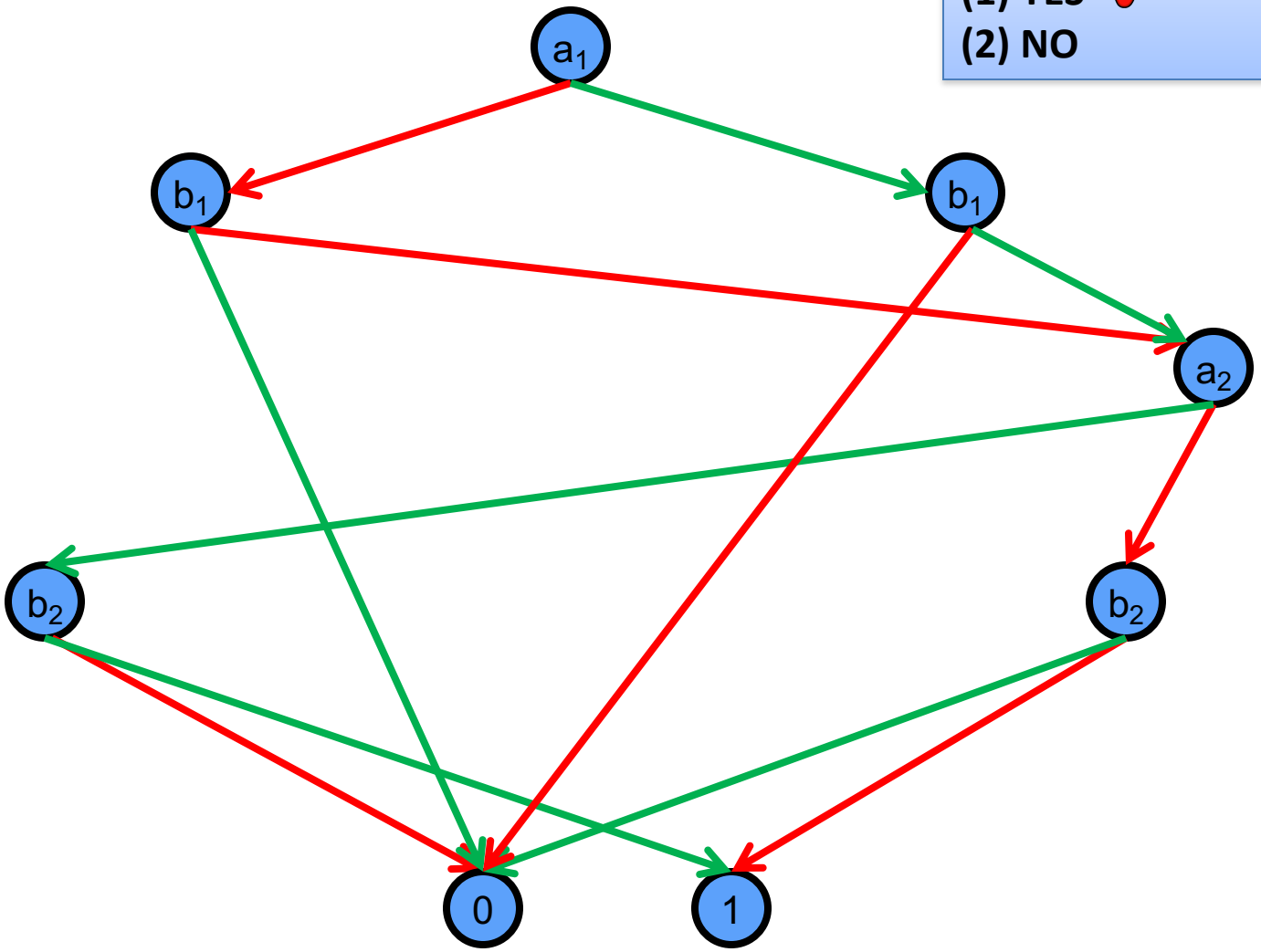
Which pairs are isomorphic?
(1) {A,B} and {C,D}
(2) {A,C} and {B,D} ✓
(3) {A,D} and {B,C}



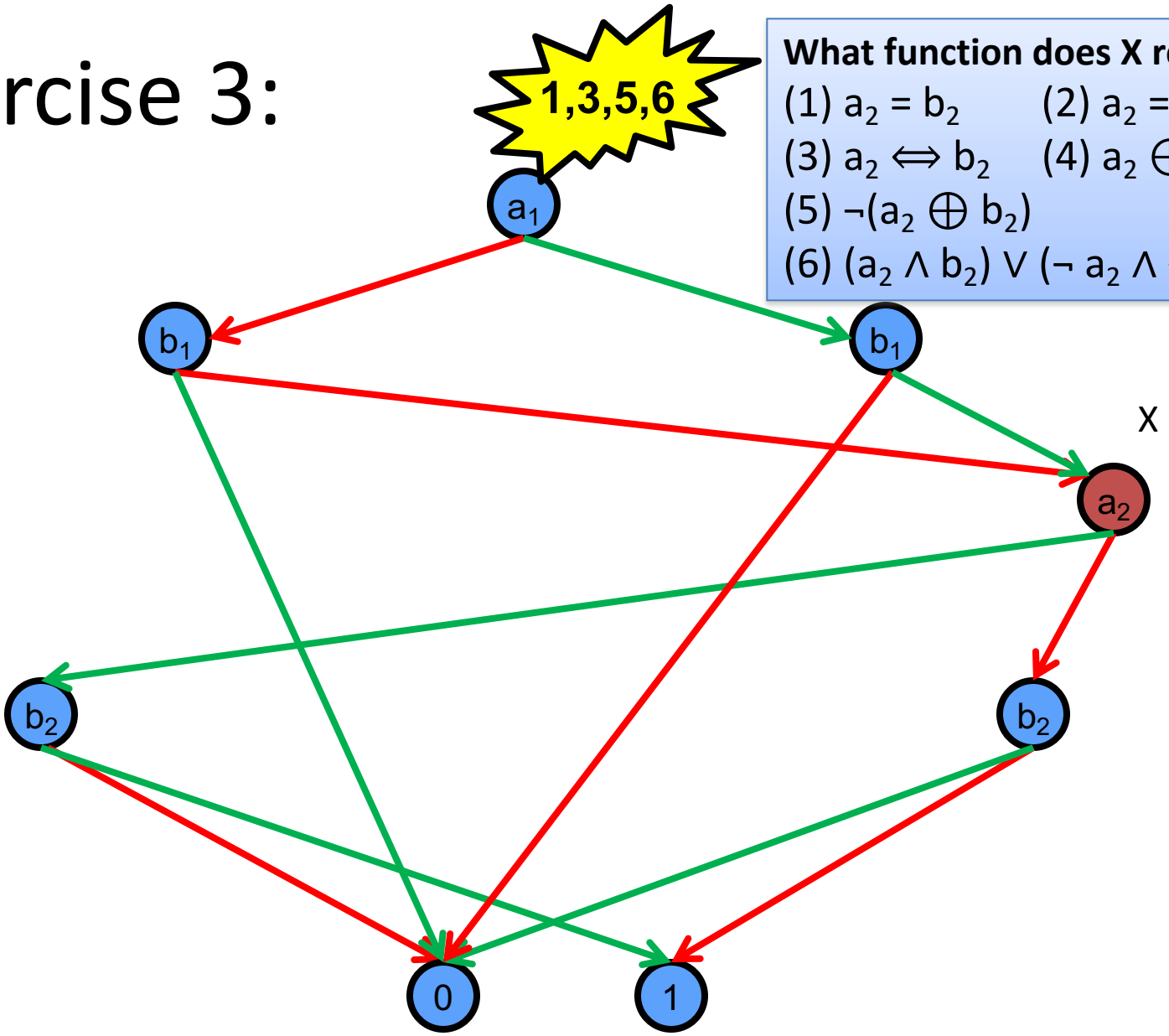
Exercise 2:

Is this a ROBDD?

(1) YES ✓
(2) NO



Exercise 3:



What function does X represent?

(1) $a_2 = b_2$ (2) $a_2 = (\neg\ b_2)$

(3) $a_2 \Leftrightarrow b_2$ (4) $a_2 \oplus b_2$

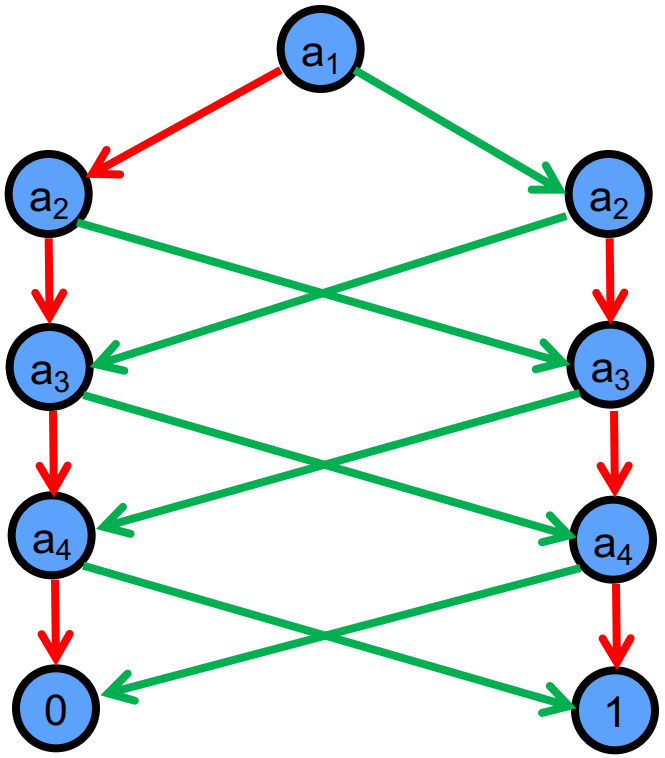
(5) $\neg(a_2 \oplus b_2)$

(6) $(a_2 \wedge b_2) \vee (\neg\ a_2 \wedge \neg\ b_2)$

Exercise 4

- Construct an ROBDD for the odd parity function with 4 boolean variables:

$$\mathbf{f = 1} \iff a_1 + a_2 + a_3 + a_4 \text{ is odd}$$



ROBDD (a.k.a. BDD) Summary

BDDs are canonical representations of Boolean formulas

- $f_1 = f_2 \Leftrightarrow$ BDD(f_1) and BDD(f_2) are isomorphic
- f is unsatisfiable $\Leftrightarrow$ BDD(f) is the leaf node "0"
- f is valid $\Leftrightarrow$ BDD(f) is the leaf node "1"
- BDD packages do these operations in constant time

Logical operations can be performed efficiently on BDDs

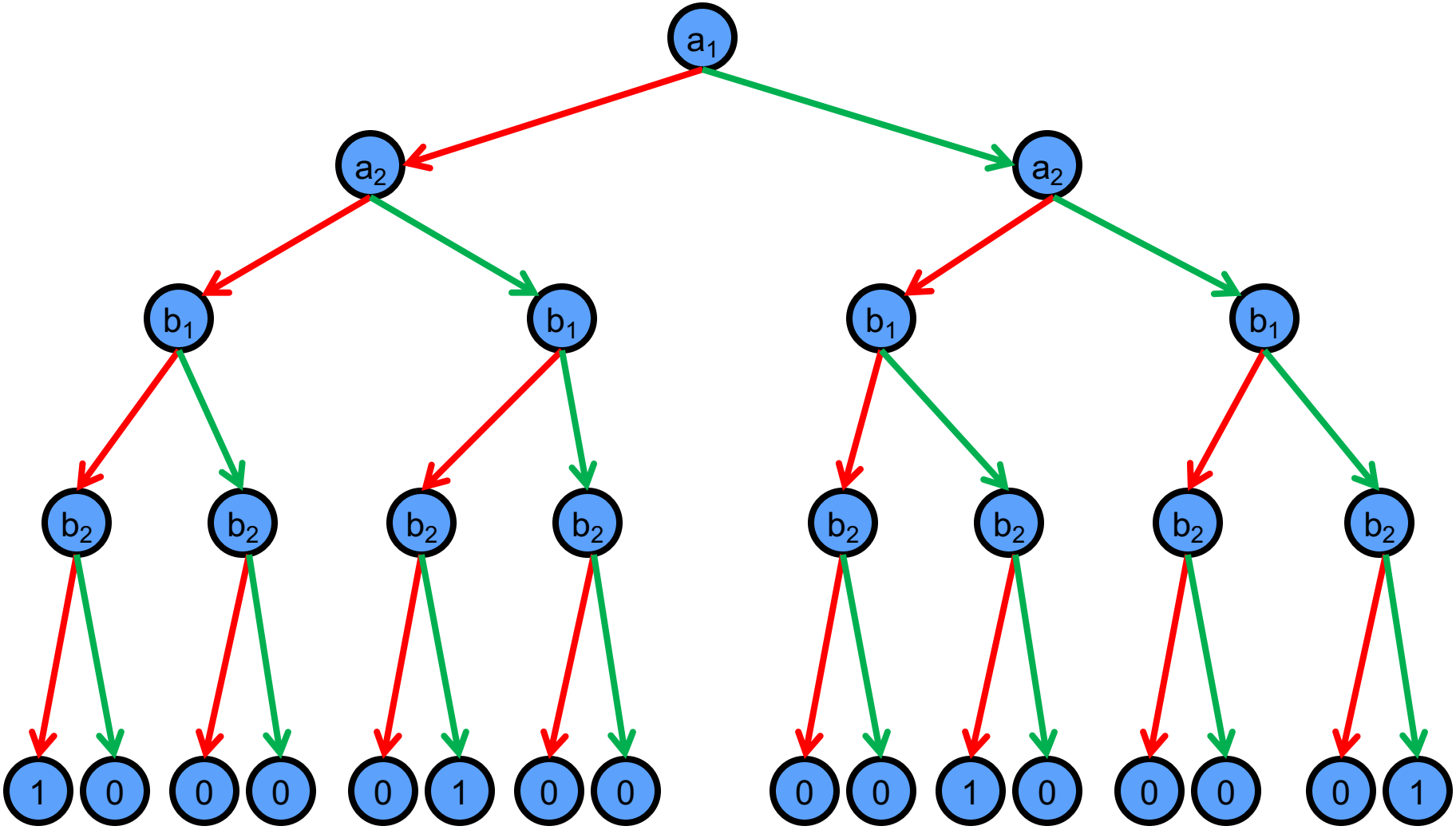
- Polynomial in argument size
- More details in next lecture

BDD size depends critically on the variable ordering

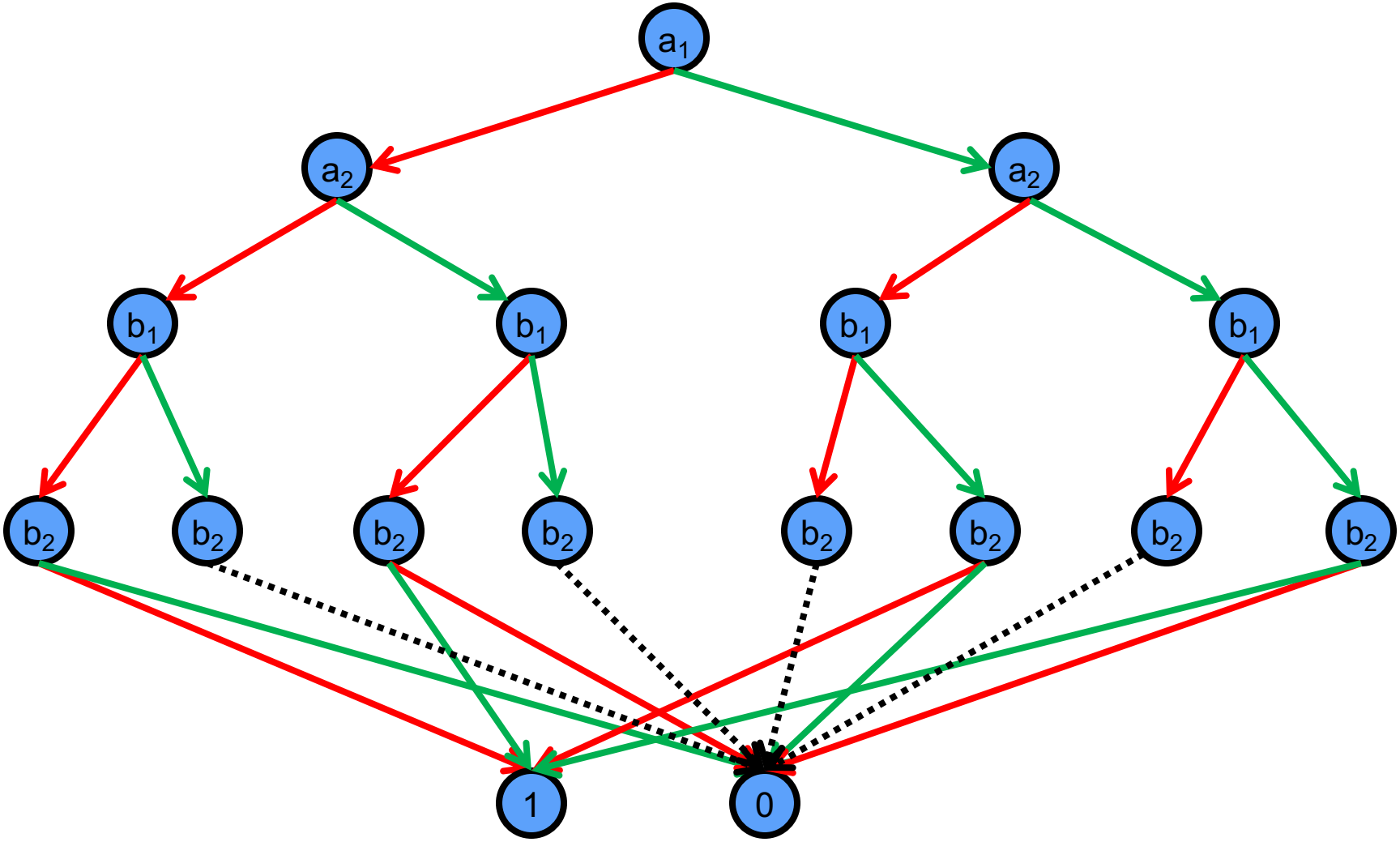
- Some formulas have exponentially large sizes for all orderings
- Others are polynomial for some orderings and exponential for others

Variable Ordering

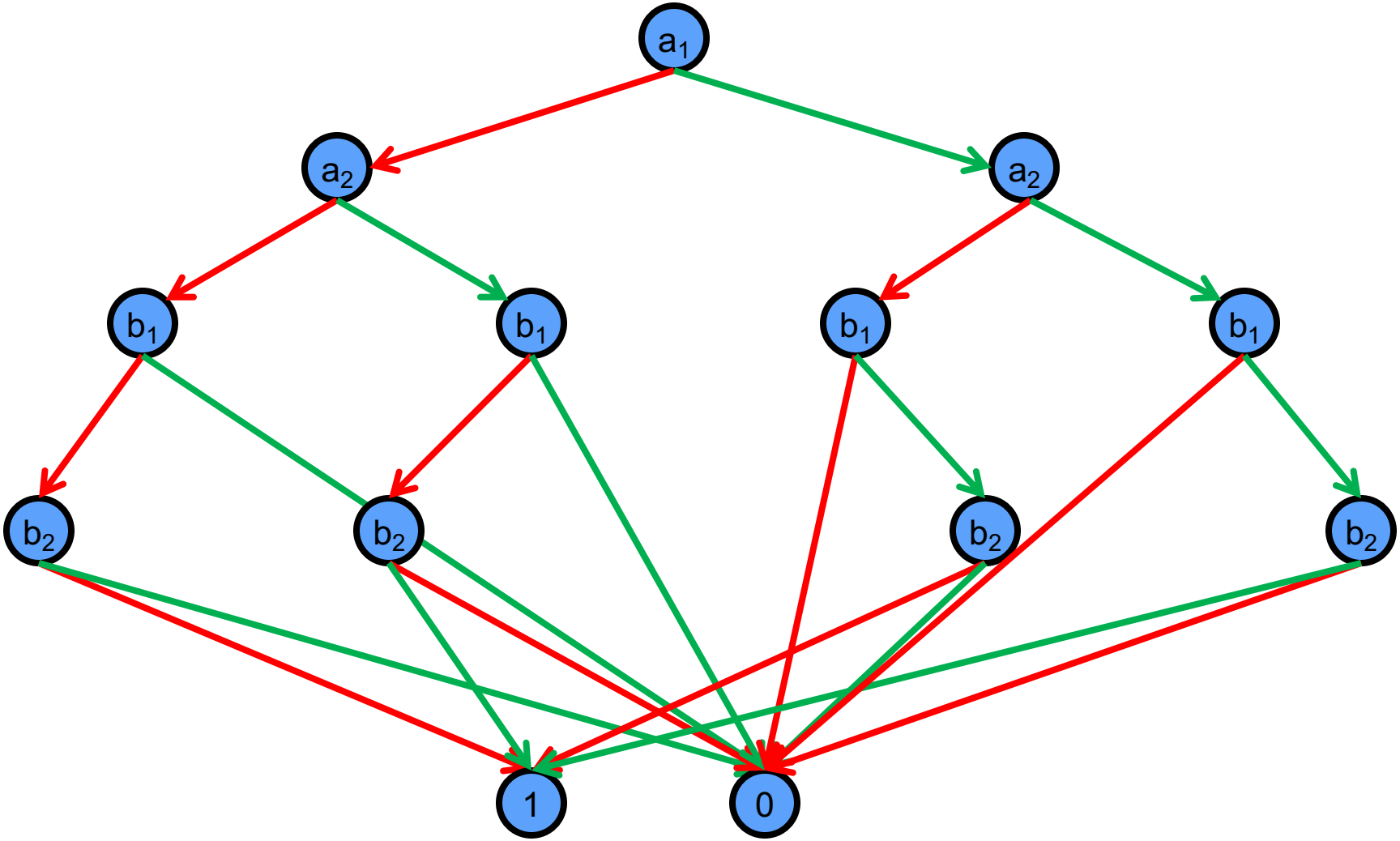
ROBDD and variable ordering



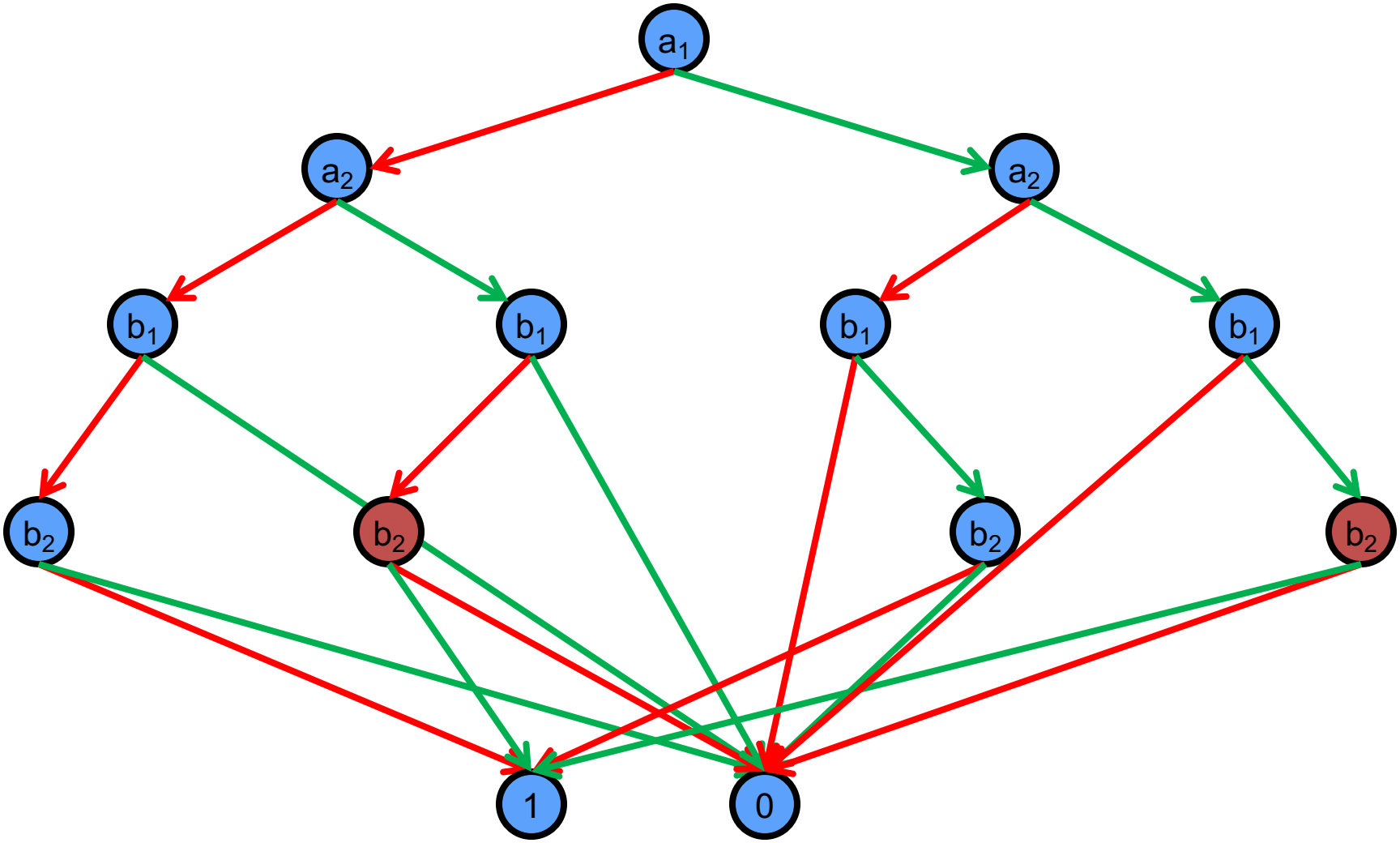
ROBDD and variable ordering



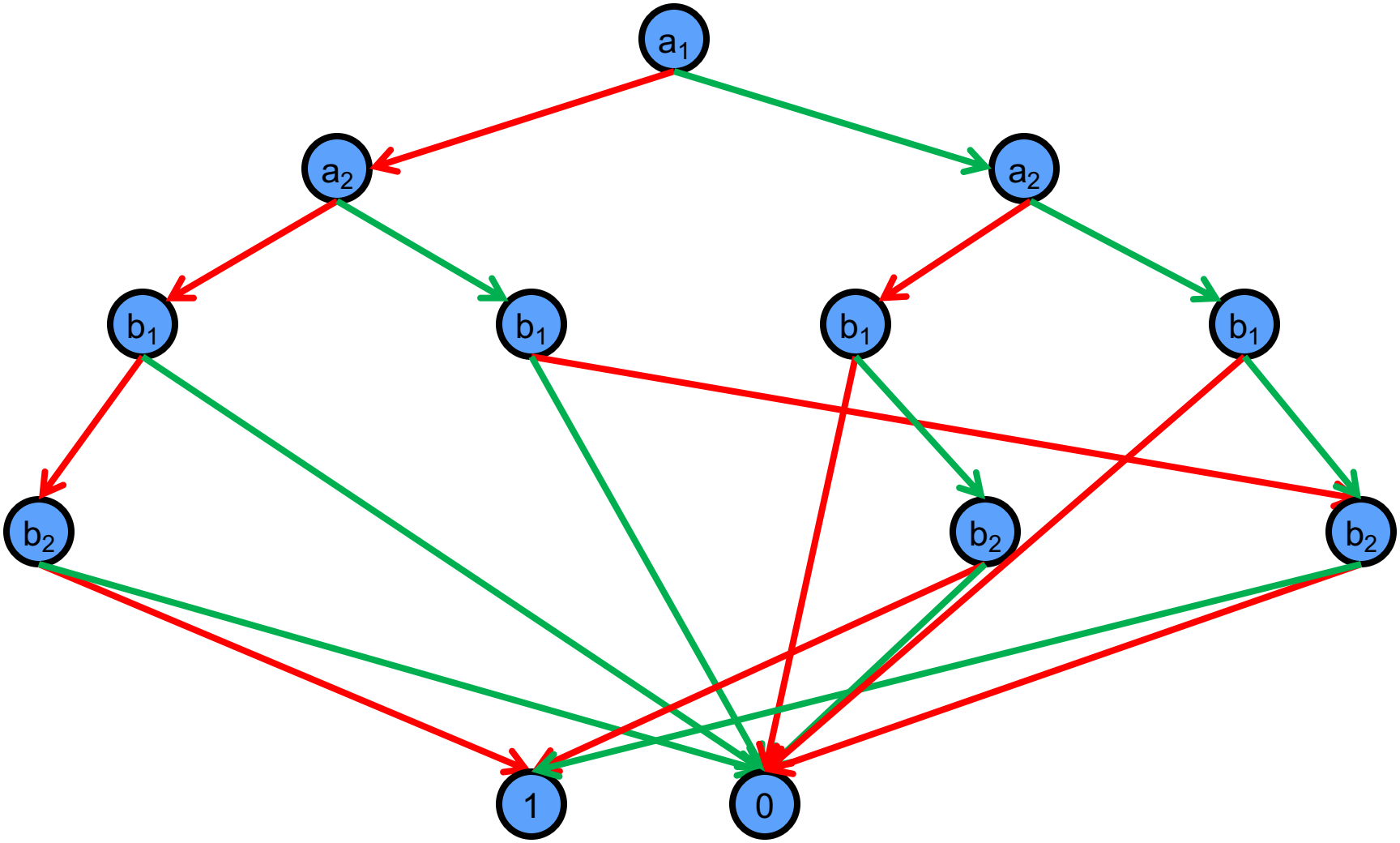
ROBDD and variable ordering



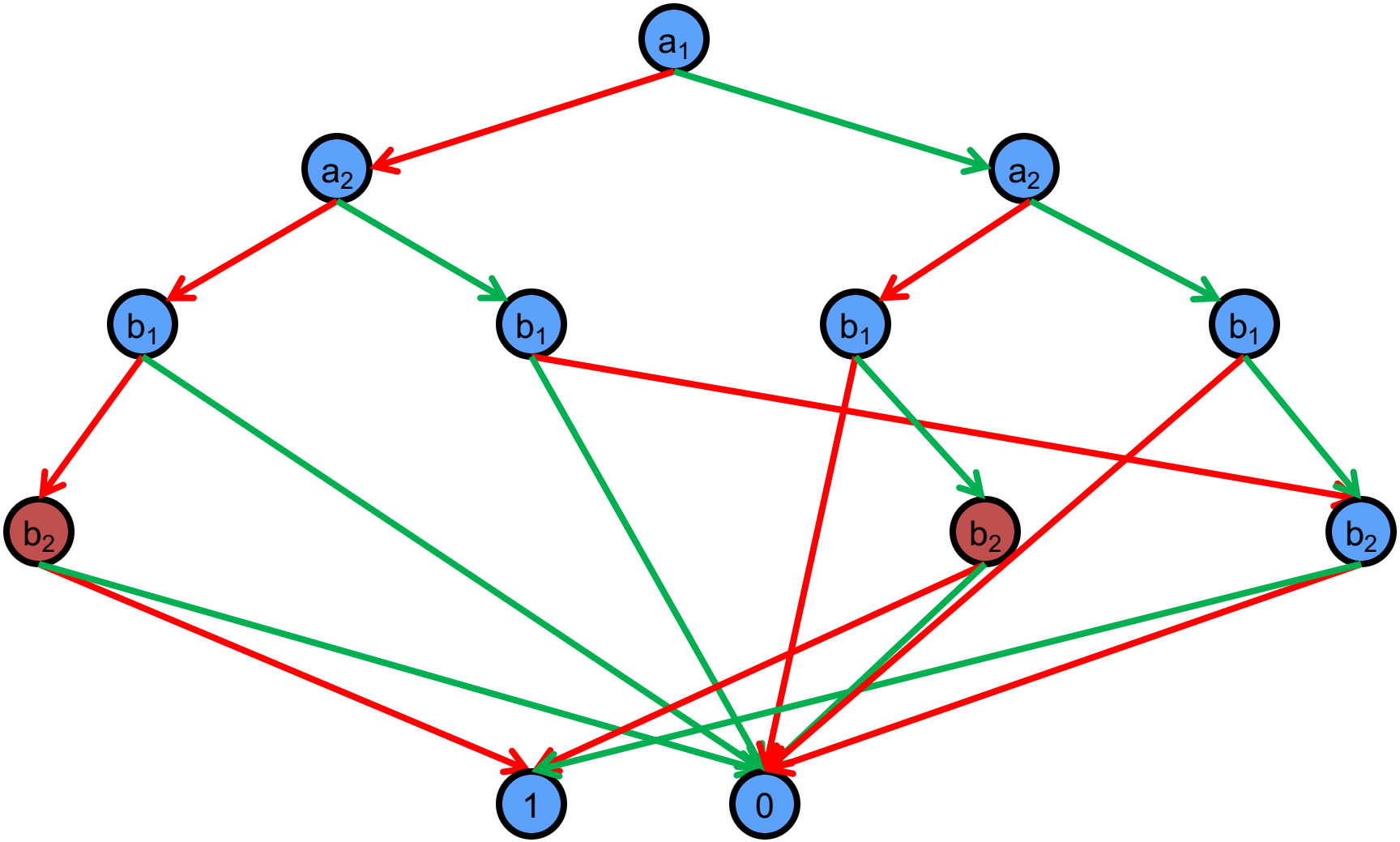
ROBDD and variable ordering



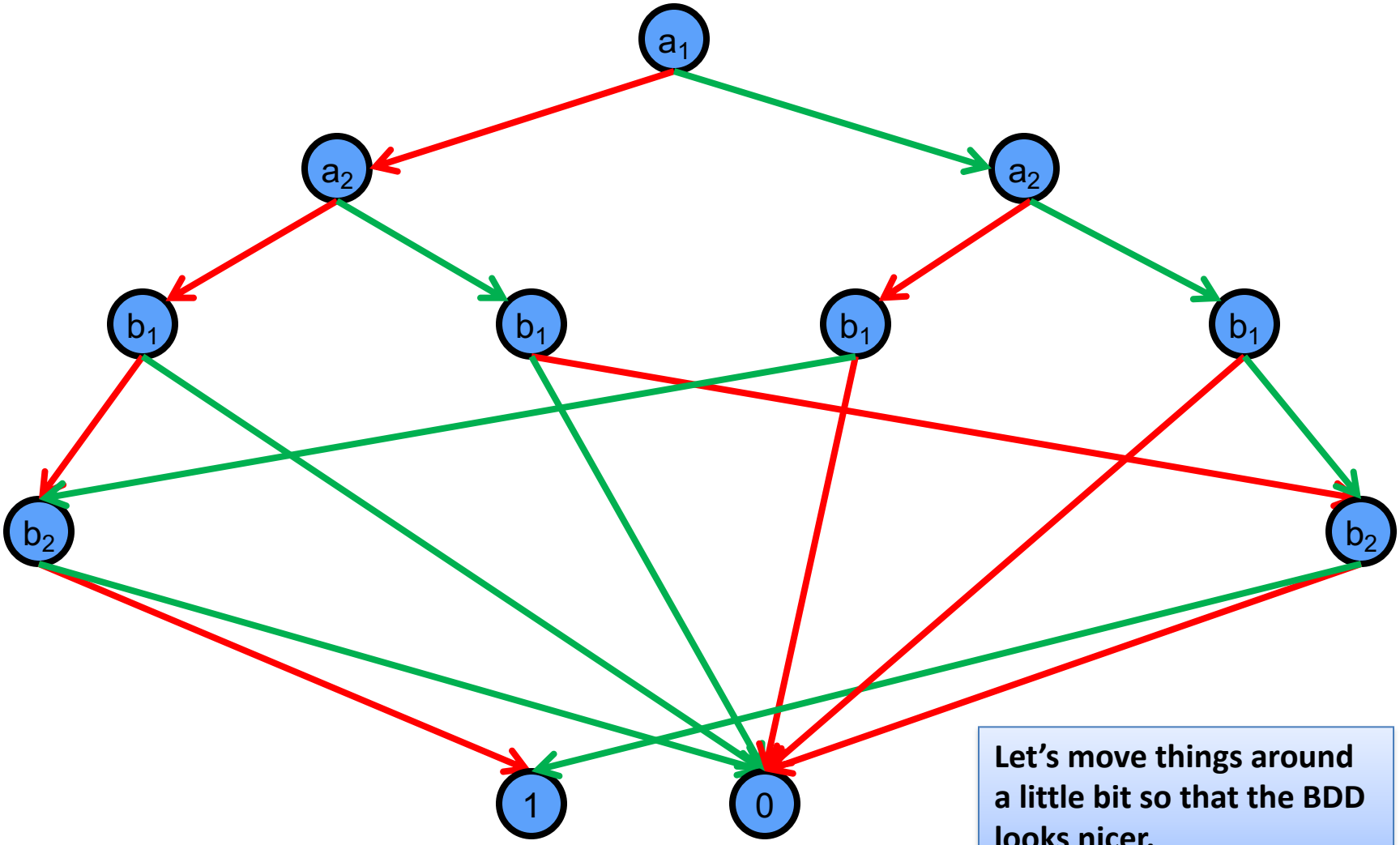
ROBDD and variable ordering



ROBDD and variable ordering

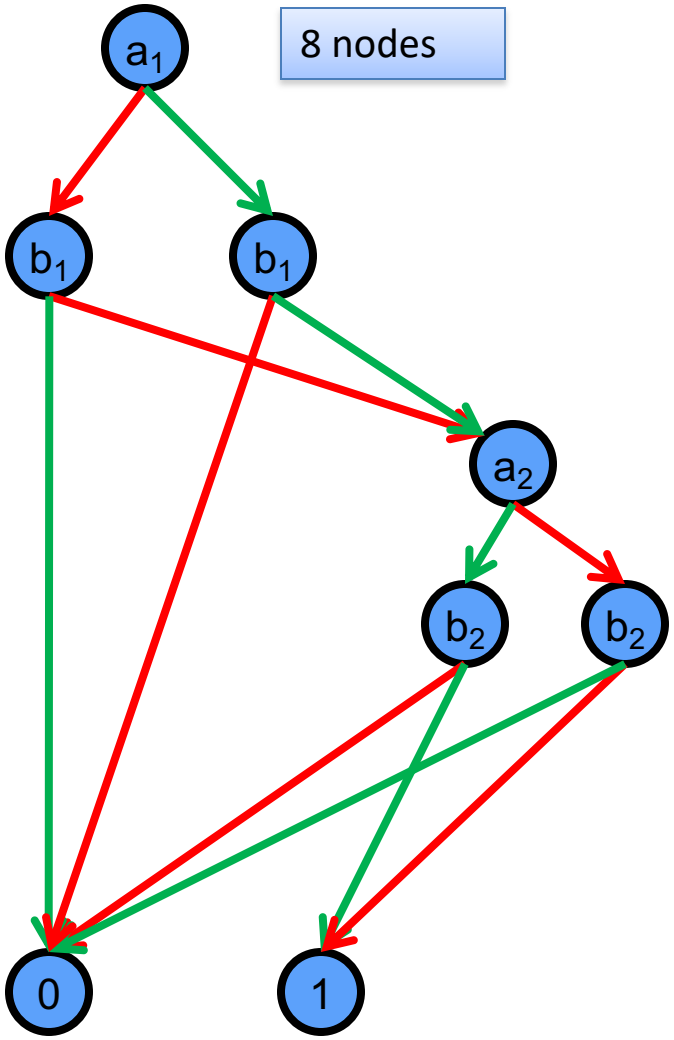


ROBDD and variable ordering



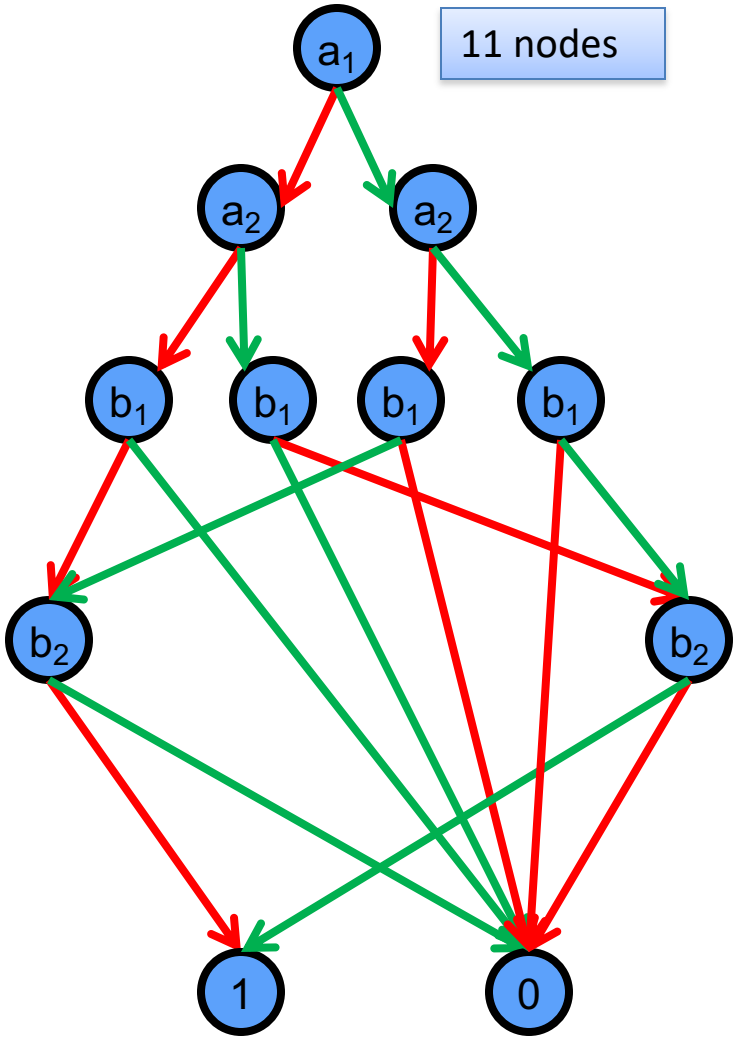
Let's move things around a little bit so that the BDD looks nicer.

ROBDD and variable ordering



8 nodes

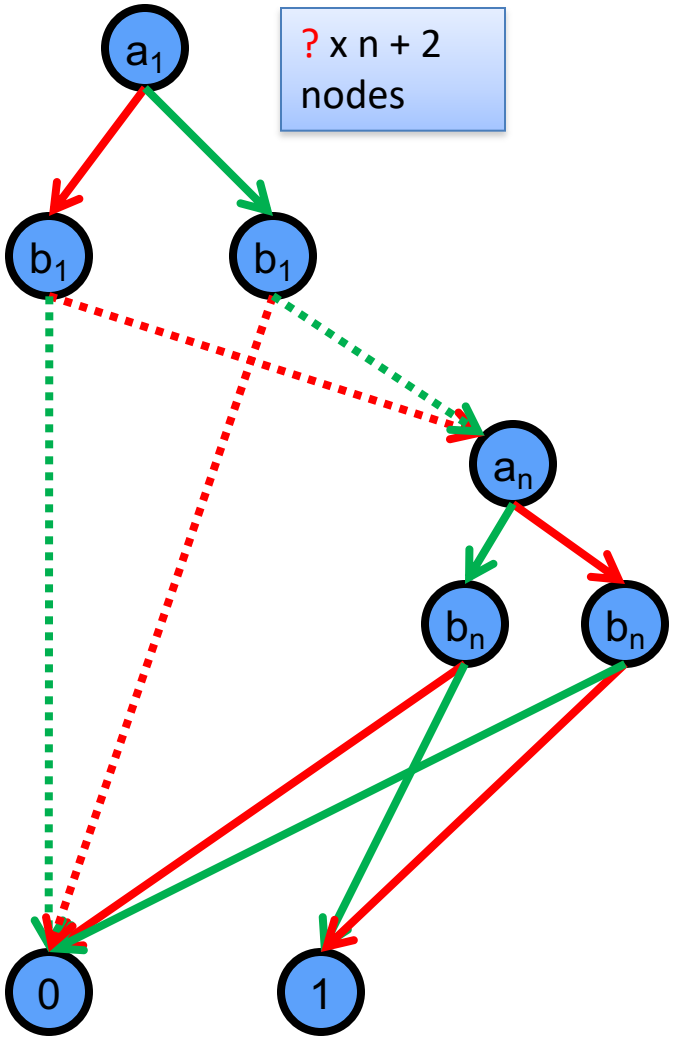
$a_1 < b_1 < a_2 < b_2$



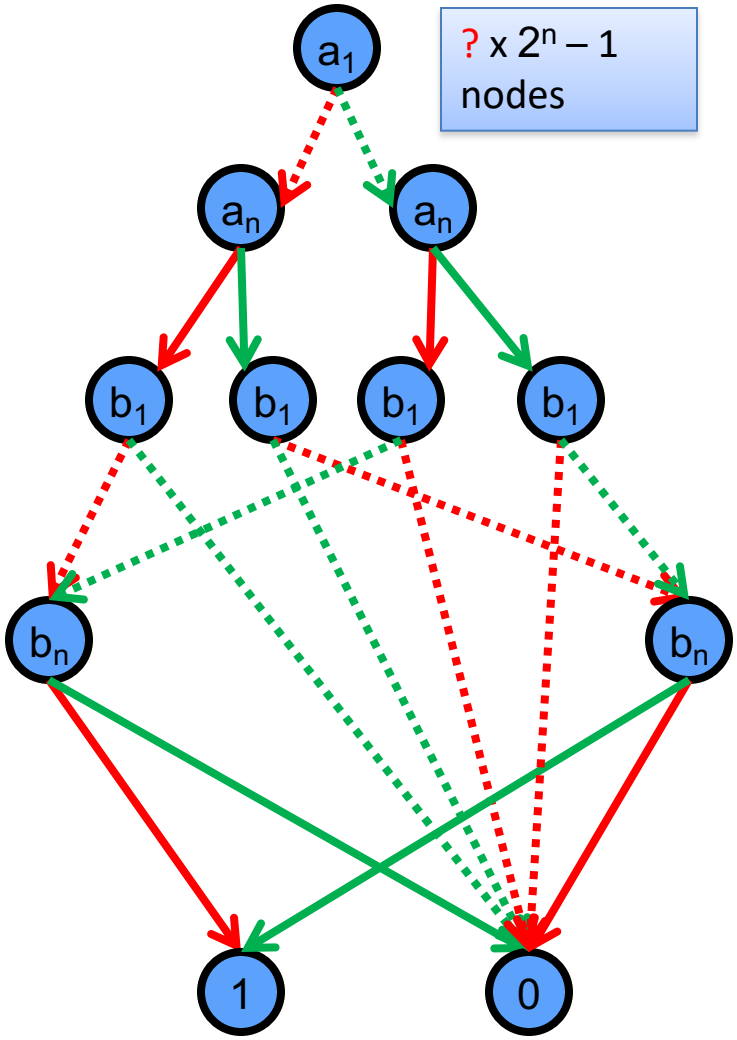
11 nodes

$a_1 < a_2 < b_1 < b_2$

ROBDD and variable ordering

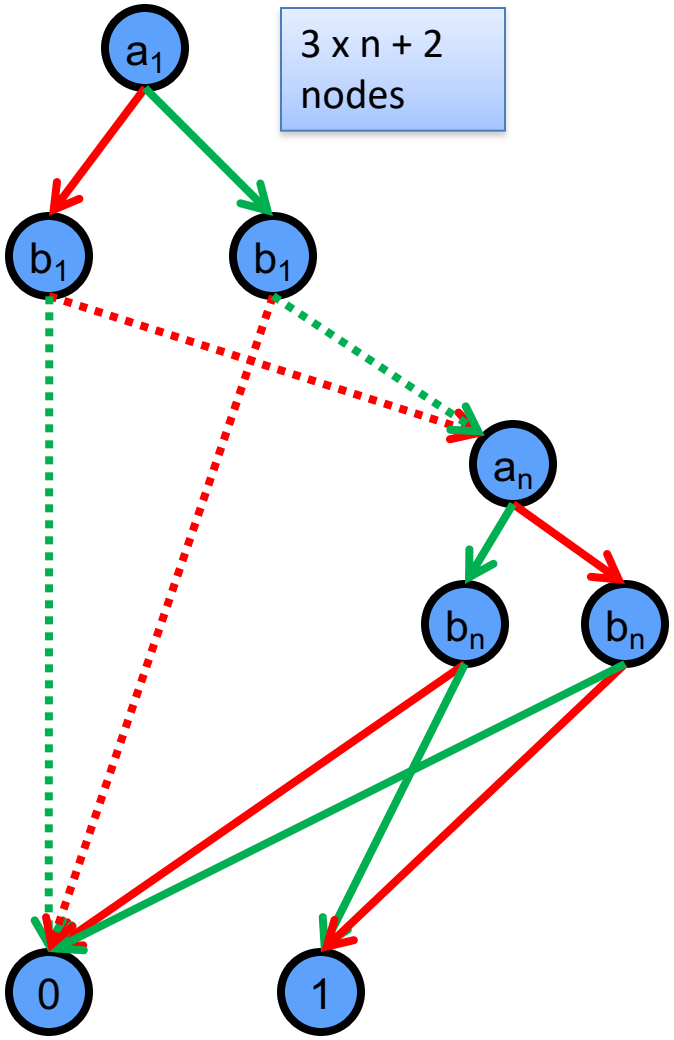


$a_1 < b_1 < \ldots < a_n < b_n$



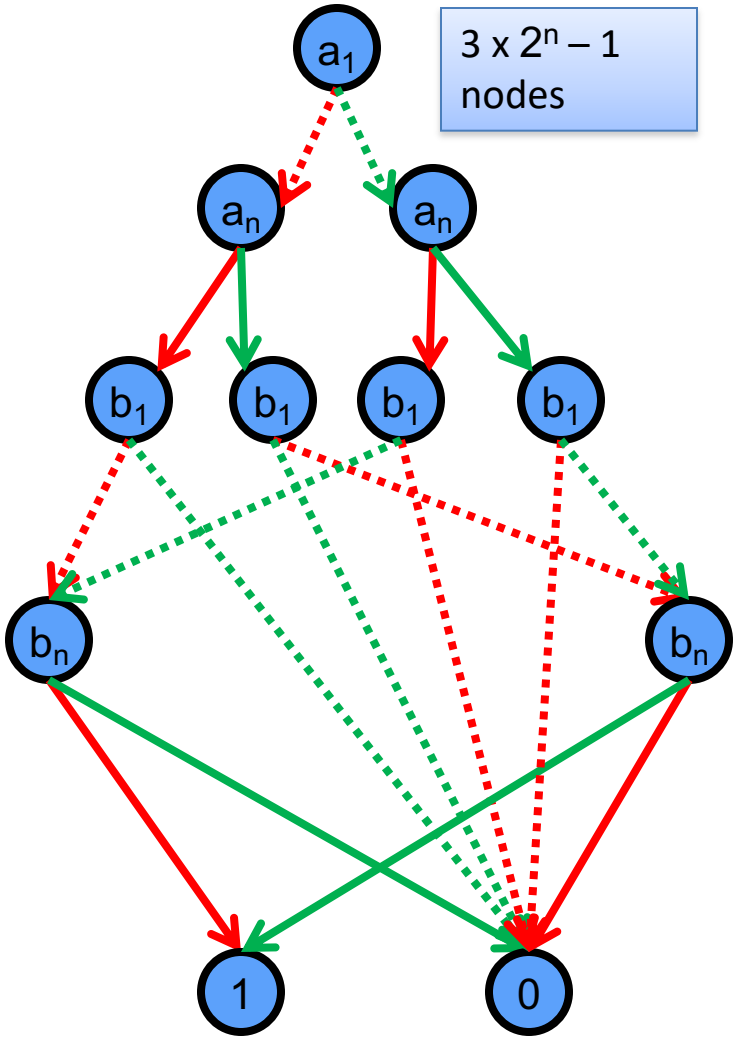
$a_1 < \ldots < a_n < b_1 < \ldots < b_n$

ROBDD and variable ordering



3 x n + 2 nodes

$a_1 < b_1 < \ldots < a_n < b_n$



3 x 2^n – 1 nodes

$a_1 < \ldots < a_n < b_1 < \ldots < b_n$

Variable Ordering Problem

For an n-bit comparator:

- if we use the ordering $a_1 < b_1 < \ldots < a_n < b_n$, the number of vertices is $3n + 2$

- if we use the ordering $a_1 < \ldots < a_n < b_1 \ldots < b_n$, the number of vertices is $3 \cdot 2^n - 1$

There are Boolean functions that have exponential size OBDDs for any variable ordering

Logical Operations on ROBDDs

ROBDD Operations

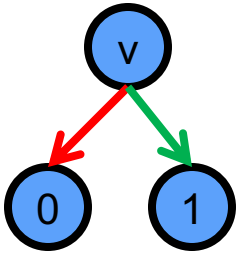
- True : BDD(TRUE)
- False: BDD(FALSE)
- Var : $v \mapsto$ BDD(v)
- Not : BDD(f) $\mapsto$ BDD(¬f)
- And : $\mathrm{BDD}(f_1) \times \mathrm{BDD}(f_2) \mapsto \mathrm{BDD}(f_1 \wedge f_2)$
- Exist : BDD(f) × $v \mapsto$ BDD(∃v. f)

Basic BDD Operations

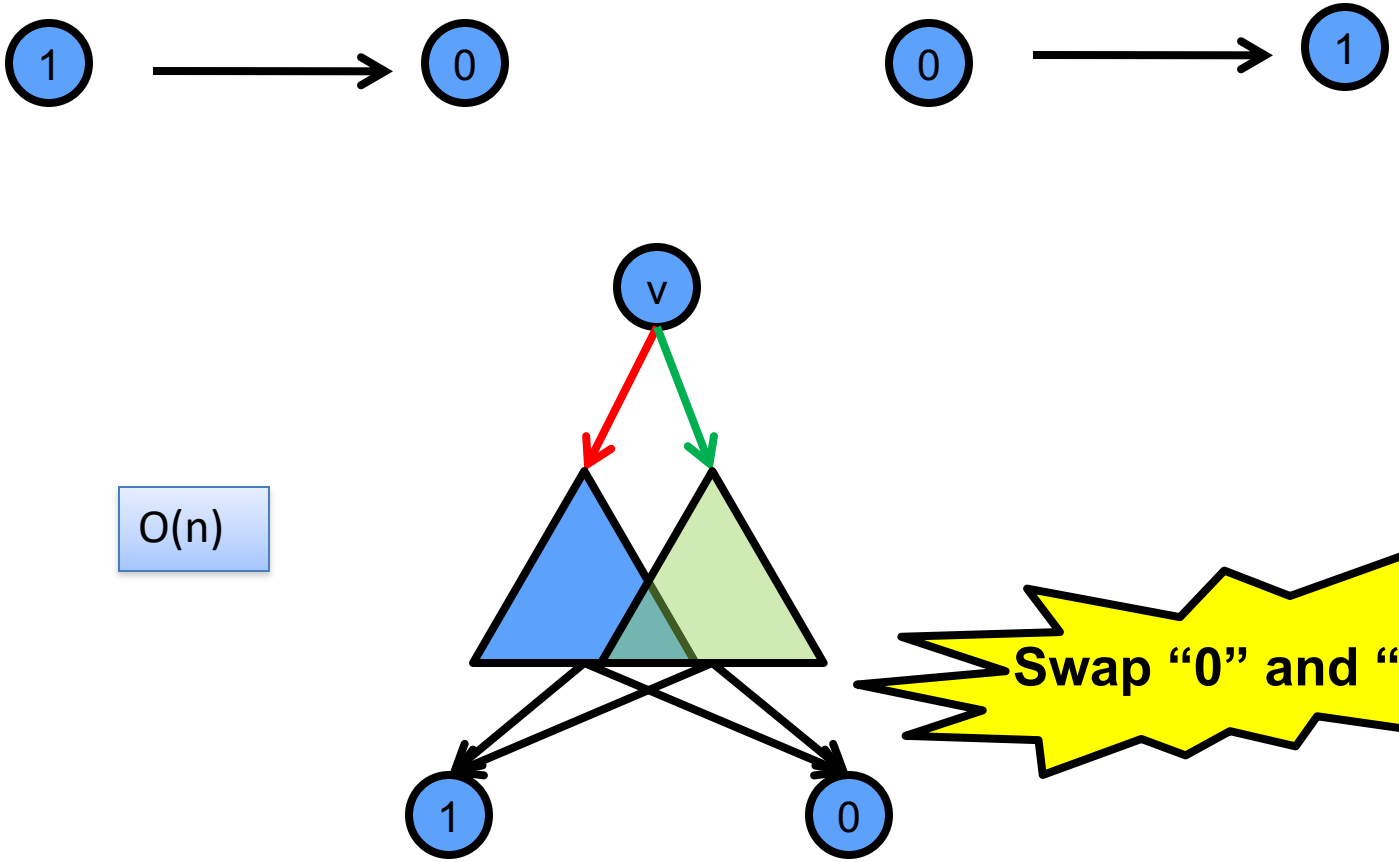
True

False

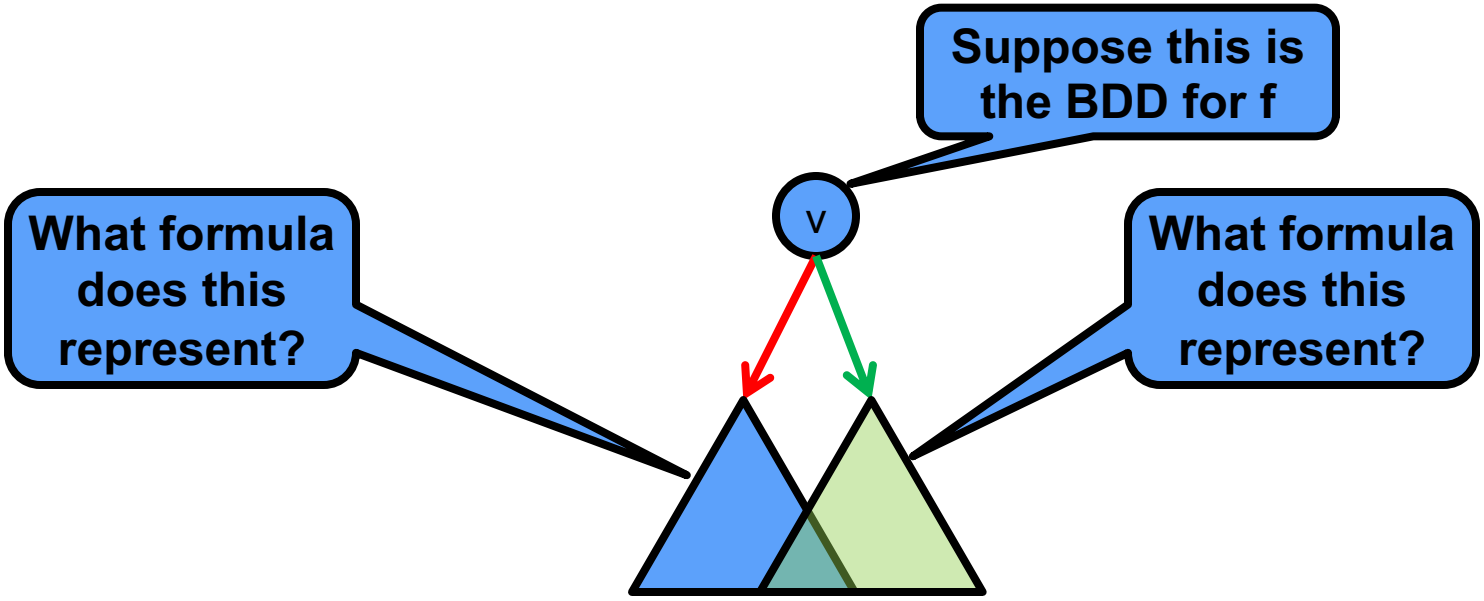
Var(v)



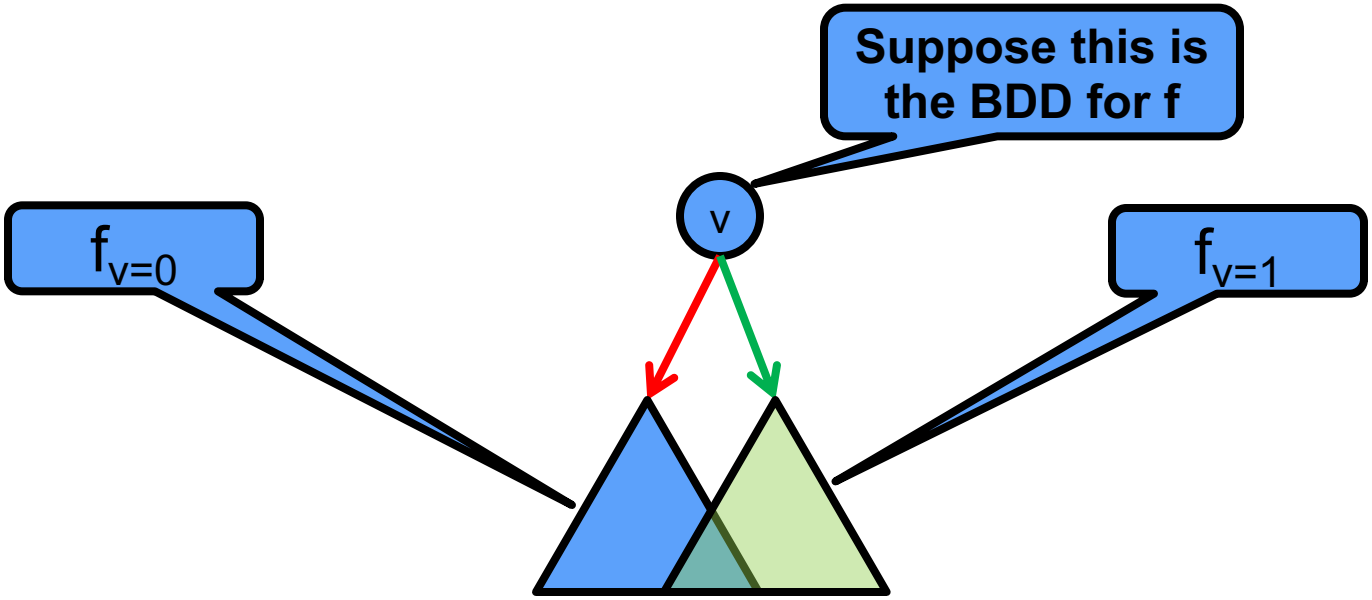
BDD Operations: Not



BDD Operations: And



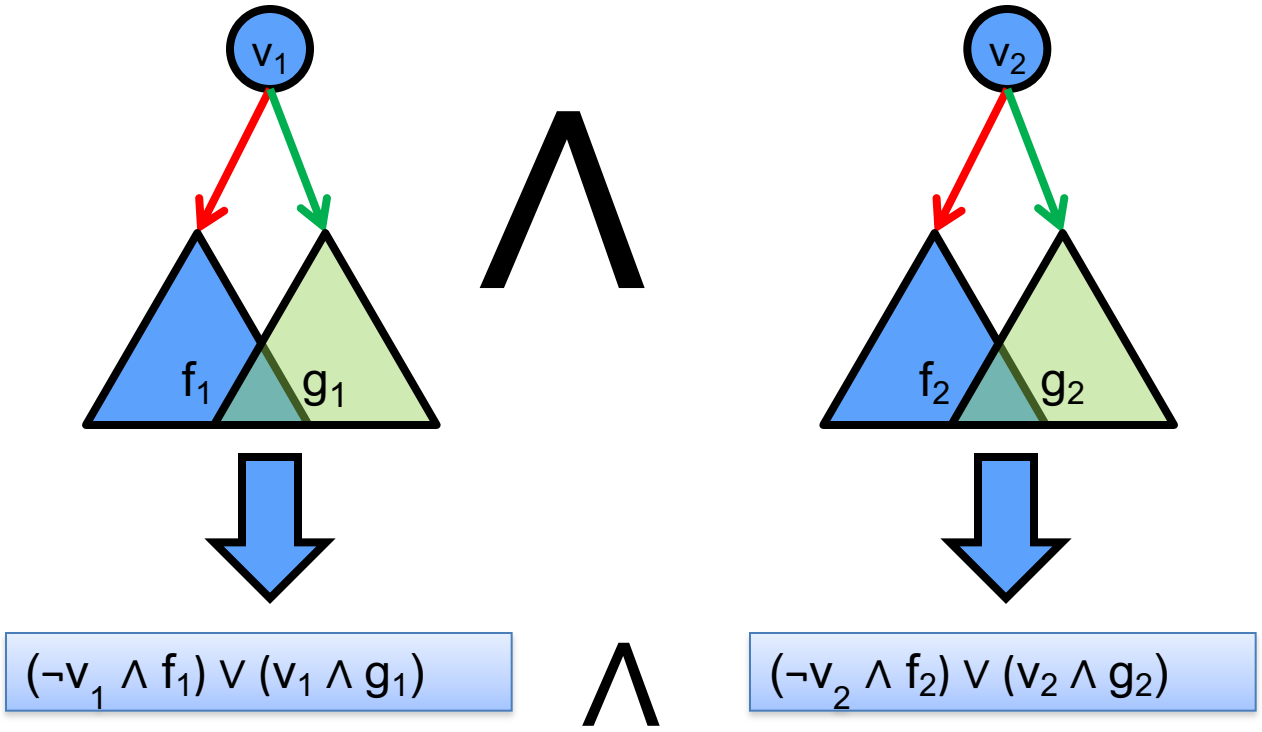
BDD Operations: And



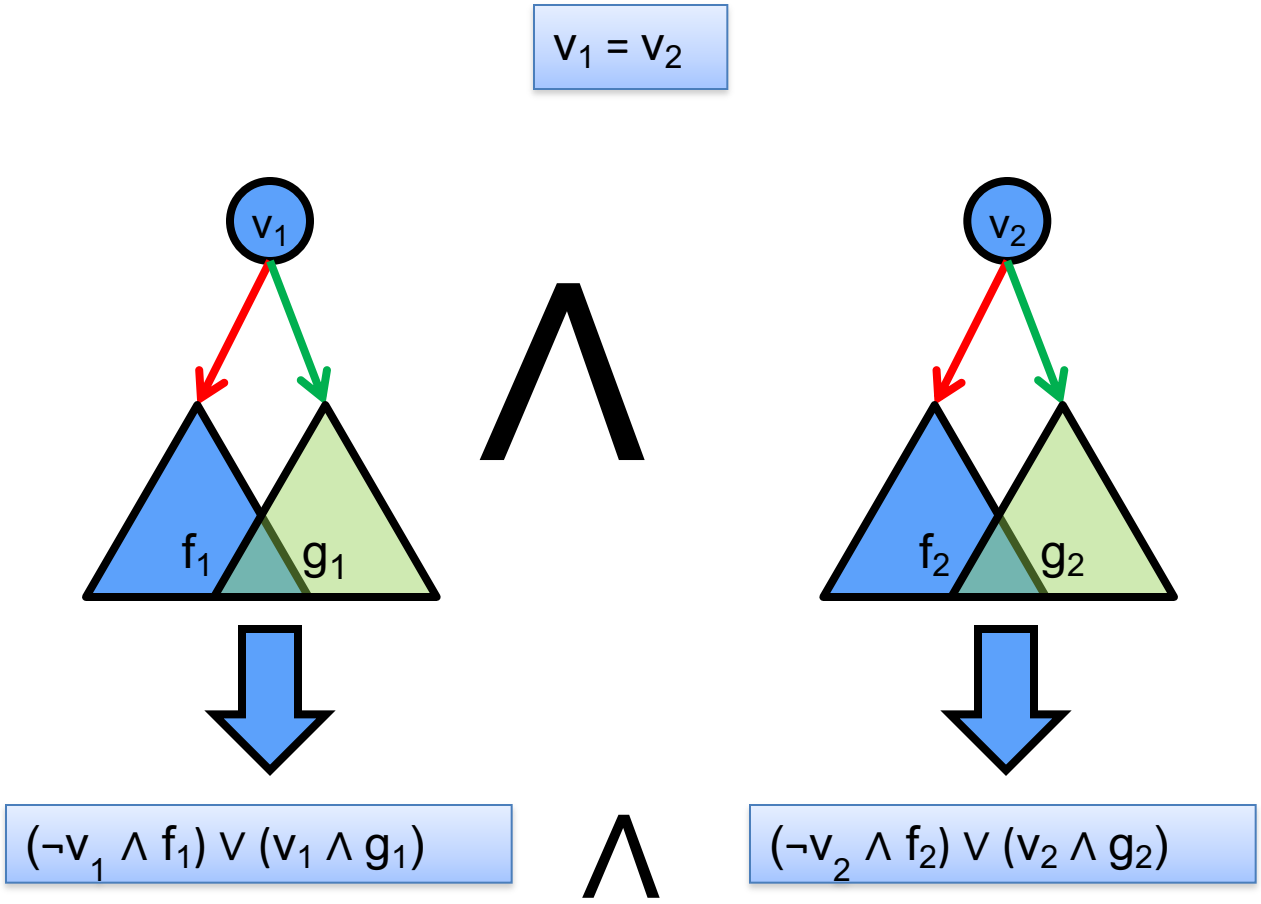
$f_{v=0}$ and $f_{v=1}$ are known as the co-factors of f w.r.t. v

$f = (\neg v \wedge f_{v=0}) \vee (v \wedge f_{v=1})$

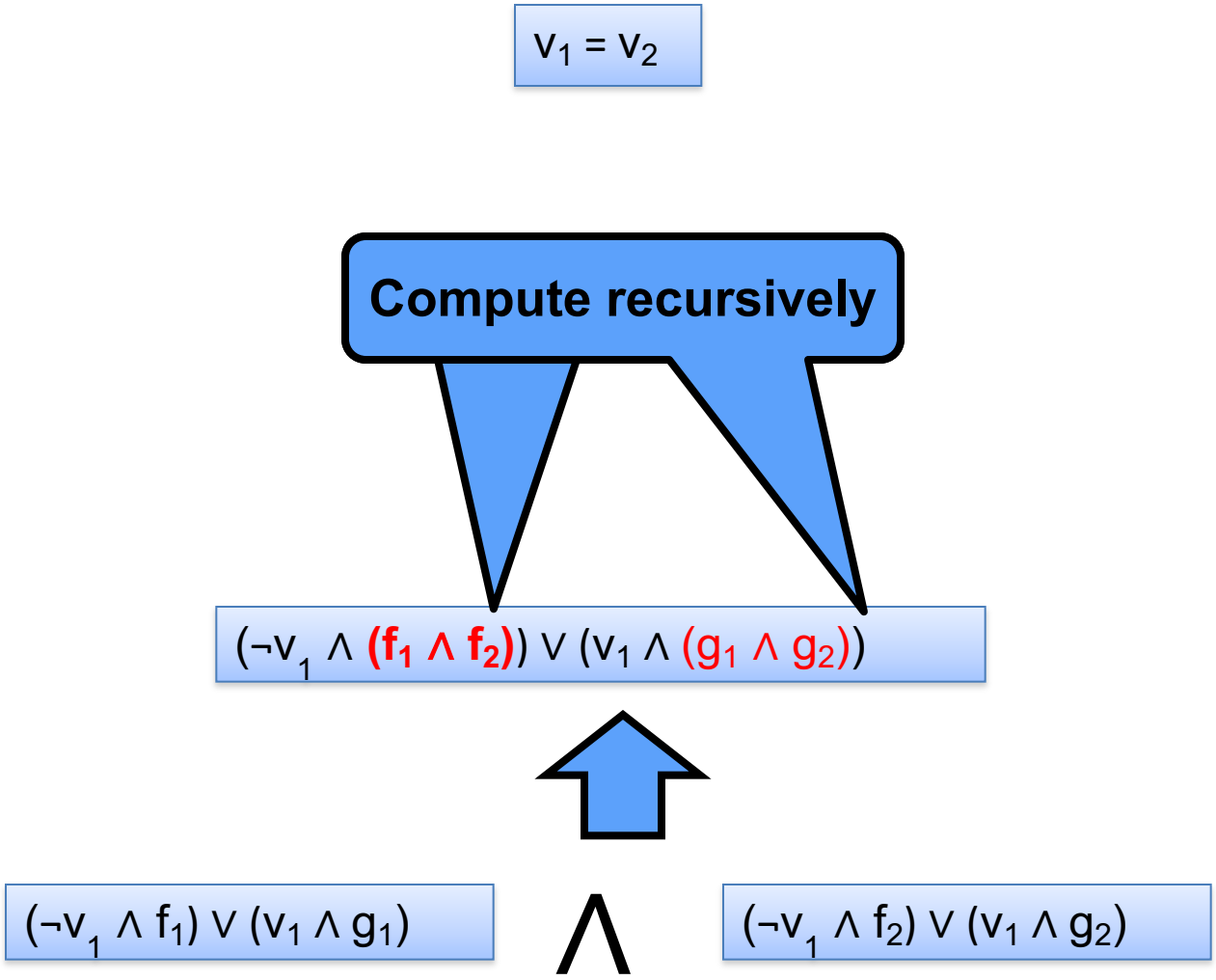
BDD Operations: And



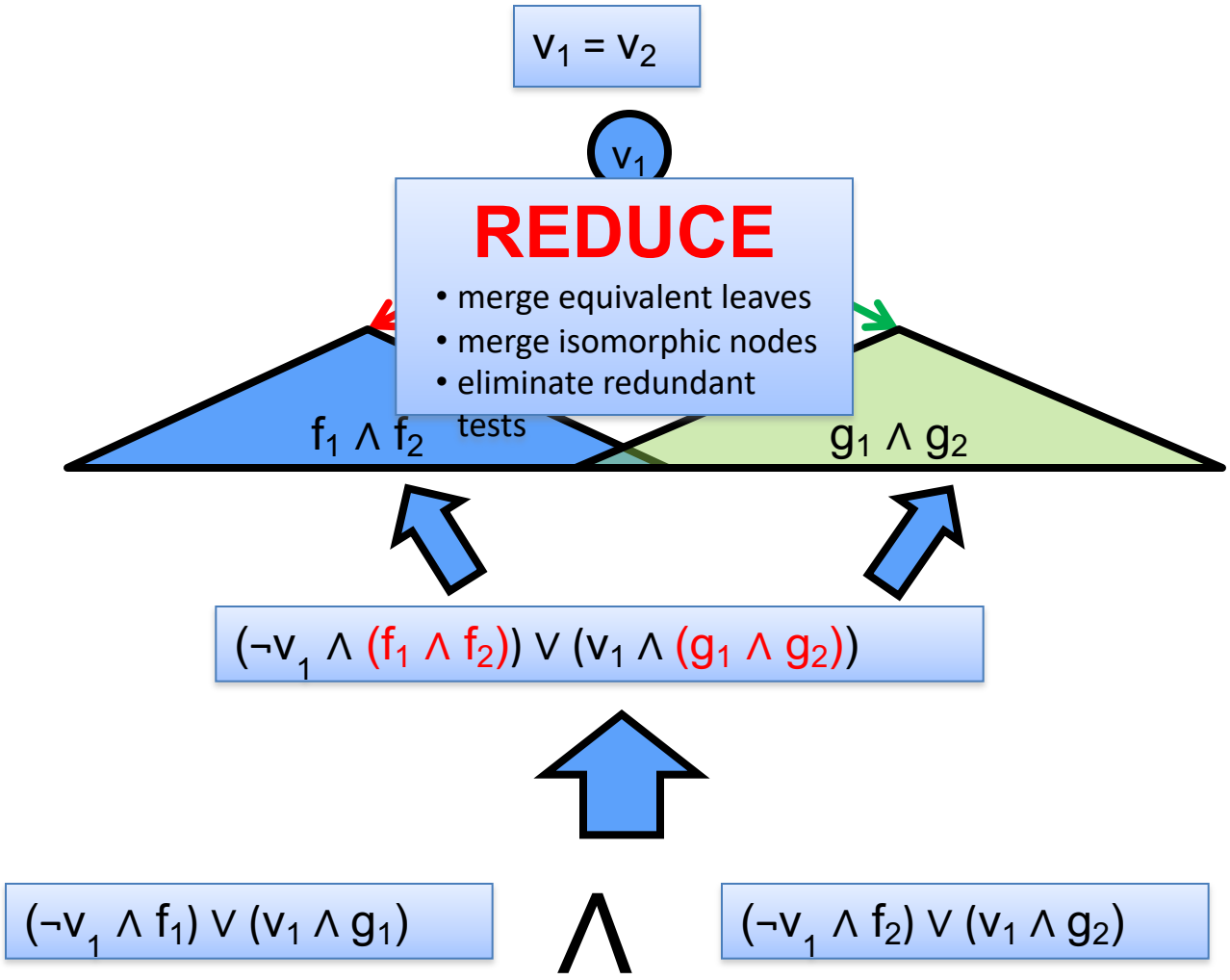
BDD Operations: And (Case 1)



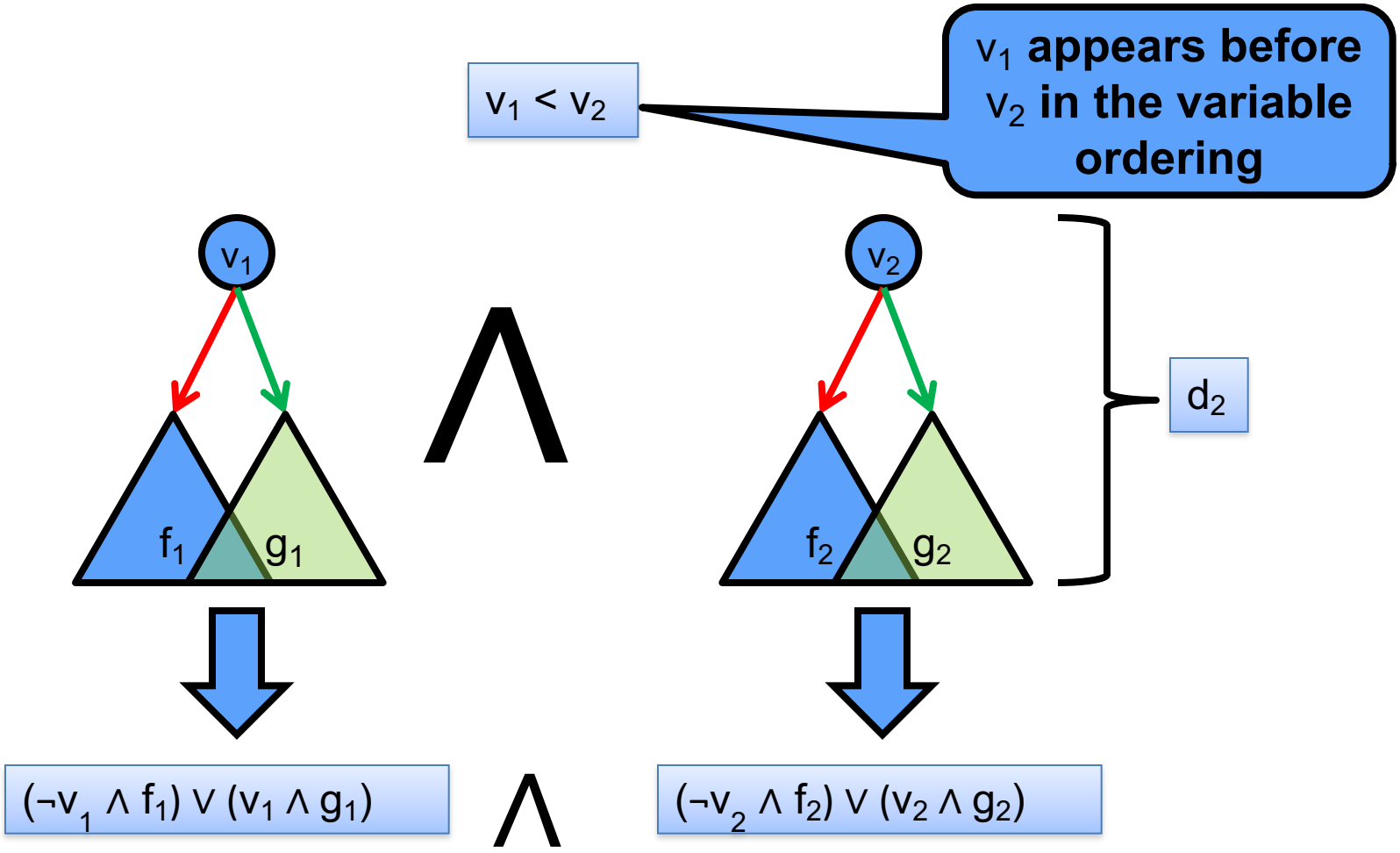
BDD Operations: And (Case 1)



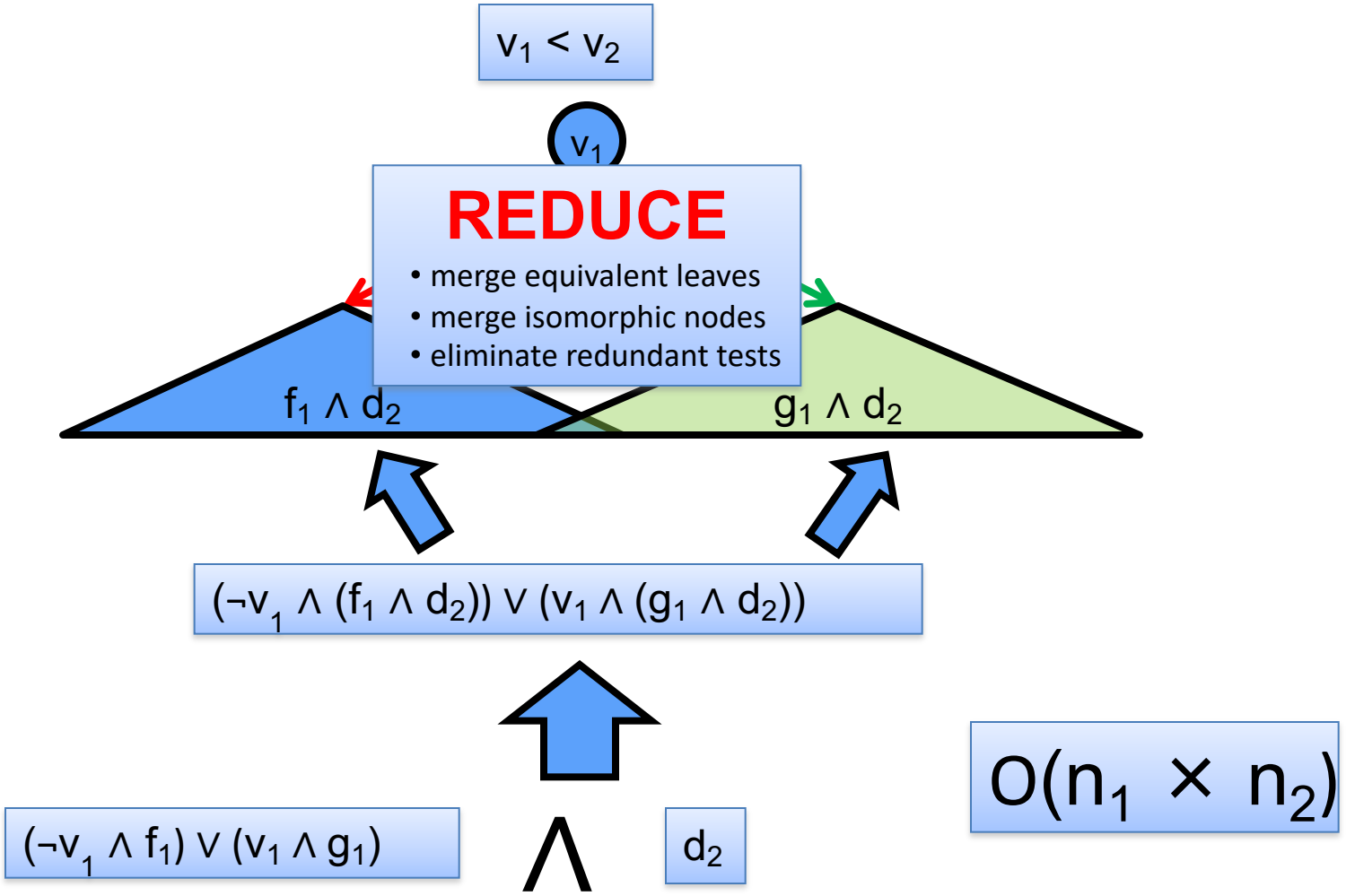
BDD Operations: And (Case 1)



BDD Operations: And (Case 2)



BDD Operations: And (Case 2)



BDD Operations: Or

$$Or(d_1, d_2)$$
$$=$$
$$\textbf{Not}\left(\textbf{And}\left(\textbf{Not}(d_1), \textbf{Not}(d_2)\right)\right)$$

$$O(n_1 \times n_2)$$

BDD Operations: Exists

Boolean quantification: $\exists a.\ f(a, b, c, d)$

– by definition

$$\exists a.\ f \ = \ f_{a=0} \vee f_{a=1}$$

– $f_{a=0}$: replace all a nodes by left sub-tree

– $f_{a=1}$: replace all a nodes by right sub-tree

BDD based model-checking

Image computation

Given

- the BDD for a set of configurations S, and
- the BDD for the set of all transitions Trans,

the BDD for all the nodes that are adjacent to S can be computed using the BDD operations

Variable renaming : replace a' with a

$$\text{Image}(S,\text{Trans}) = (\exists\, x_1,\ldots,x_n\ .\ (S(x_1,\ldots,x_n) \wedge \text{Trans}(x_1,\ldots,x_n, x_1',\ldots,x_n'))\ )\ [\, x_1' | x_1,\ \ldots,\ x_n' | x_n]$$

Reachability Algorithm

```
Init  = BDD for initial set of configurations
Trans = BDD for all transitions
Target = BDD for the target configurations

S = Init;
while (true) {
   I = Image(S,Trans); //compute adjacent nodes to S
   if (And(Not(S),I) == False) //no new nodes found
      break;
   S = Or(S,I); //add newly discovered nodes to result
}
return S;
```

Symbolic Model Checking. Has been done for graphs with 10^{20} nodes.

Exercise

Construct an ROBDD for the boolean function

$$f = 1 \iff (x_1 \leftrightarrow \neg x_{2)} \wedge (x_2 \leftrightarrow \neg x_3)$$

by combining ROBDDs for the boolean functions $x_1 \leftrightarrow \neg x_2$ and $x_2 \leftrightarrow \neg x_3$.