Algorithmic Verification COMP 3153

Lecture 16
Binary Decision Diagrams

(last update: April 6, 2017)

Symbolic CTL Model Checking Algorithm

***A* = (*Q*, q_0, δ, *L*)** fixed

Procedure SMC(φ)

case $\varphi \in \mathcal{P}$ /* Atomic Proposition */
  return $L^{-1}(\varphi)$;

case $\varphi = \neg\psi$ /* Negation */
  return $Q \setminus \text{SMC}(\psi)$;

case $\varphi = \textbf{EX}\psi$ /* Exists Next */
  return $\delta^{-1}(\text{SMC}(\psi))$;

case $\varphi = \textbf{AX}\psi$ /* Always Next */
  return $Q \setminus \delta^{-1}(Q \setminus \text{SMC}(\psi))$;

case $\varphi = \textbf{E}\ \psi_1\ \textbf{UNTIL}\ \psi_2$ /* Exists Until */
  return $\mu X.\text{SMC}(\varphi_2) \cup (\text{SMC}(\varphi_1) \cap \delta^{-1}(X))$

case $\varphi = \textbf{A}\ \psi_1\ \textbf{UNTIL}\ \psi_2$ /* Always Until */
  return $\mu X.\text{SMC}(\varphi_2) \cup (\text{SMC}(\varphi_1) \cap (Q \setminus \delta^{-1}(Q \setminus X)))$

case $\varphi = \textbf{EG}\varphi$ /* Exists Globally */
  return $\nu X.\text{SMC}(\psi) \cap \delta^{-1}(X)$

case $\varphi = \textbf{EF}\varphi$ /* Exists Eventually */
  return $\mu X.\text{SMC}(\psi) \cup \delta^{-1}(X)$

Symbolic CTL Model Checking Algorithm

$\boldsymbol{A = (Q, q_0, \delta, L)}$ fixed

Procedure SMC(φ)

case $\varphi \in \mathcal{P}$ /* Atomic Proposition */
  return $L^{-1}(\varphi)$;

case $\varphi = \neg\psi$ /* Negation */
  return $Q \setminus$ SMC(ψ);

case $\varphi =$ **EX**ψ /* Exists Next */
  return δ^{-1}(SMC(ψ));

case $\varphi =$ **AX**ψ /* Always Next */
  return $Q \setminus \delta^{-1}(Q \setminus$ SMC(ψ));

case $\varphi =$ **E** ψ_1 **UNTIL** ψ_2 /* Exists Until */
  return $\mu X.$SMC(φ_2) $\cup$ (SMC(φ_1) $\cap \delta^{-1}(X)$)

case $\varphi =$ **A** ψ_1 **UNTIL** ψ_2 /* Always Until */
  return $\mu X.$SMC(φ_2) $\cup$ (SMC(φ_1) $\cap (Q \setminus \delta^{-1}(Q \setminus X)))$

case $\varphi =$ **EG**φ /* Exists Globally */
  return $\nu X.$SMC(ψ) $\cap \delta^{-1}(X)$

case $\varphi =$ **EF**φ /* Exists Eventually */
  return $\mu X.$SMC(ψ) $\cup \delta^{-1}(X)$

Symbolic CTL Model Checking Algorithm

***A* = (*Q*, q_0, δ, *L*)** fixed

Procedure SMC(φ)

case $\varphi \in \mathcal{P}$ /* Atomic Proposition */
  return $L^{-1}(\varphi)$;

case $\varphi = \neg\psi$ /* Negation */
  return $Q \setminus$ SMC(ψ);

case $\varphi =$ **EX**ψ /* Exists Next */
  return δ^{-1}(SMC(ψ));

case $\varphi =$ **AX**ψ /* Always Next */
  return $Q \setminus \delta^{-1}(Q \setminus$ SMC(ψ));

case $\varphi =$ **E** ψ_1 **UNTIL** ψ_2 /* Exists Until */
  return $\mu X.$SMC(φ_2) $\cup$ (SMC(φ_1) $\cap\, \delta^{-1}(X)$)

case $\varphi =$ **A** ψ_1 **UNTIL** ψ_2 /* Always Until */
  return $\mu X.$SMC(φ_2) $\cup$ (SMC(φ_1) $\cap\, (Q \setminus \delta^{-1}(Q \setminus X))$)

case $\varphi =$ **EG**φ /* Exists Globally */
  return $\nu X.$SMC(ψ) $\cap\, \delta^{-1}(X)$

case $\varphi =$ **EF**φ /* Exists Eventually */
  return $\mu X.$SMC(ψ) $\cup\, \delta^{-1}(X)$

Symbolic CTL Model Checking Algorithm

$\boldsymbol{A = (Q, q_0, \delta, L)}$ fixed

Procedure SMC(φ)

case $\varphi \in \mathcal{P}$ /* Atomic Proposition */
  return $L^{-1}(\varphi)$;

case $\varphi = \neg\psi$ /* Negation */
  return $Q \setminus \mathrm{SMC}(\psi)$;

case $\varphi = \mathbf{EX}\psi$ /* Exists Next */
  return $\delta^{-1}(\mathrm{SMC}(\psi))$;

case $\varphi = \mathbf{AX}\psi$ /* Always Next */
  return $Q \setminus \delta^{-1}(Q \setminus \mathrm{SMC}(\psi))$;

case $\varphi = \mathbf{E}\ \psi_1\ \mathbf{UNTIL}\ \psi_2$ /* Exists Until */
  return $\mu X.\mathrm{SMC}(\varphi_2) \cup (\mathrm{SMC}(\varphi_1) \cap \delta^{-1}(X))$

case $\varphi = \mathbf{A}\ \psi_1\ \mathbf{UNTIL}\ \psi_2$ /* Always Until */
  return $\mu X.\mathrm{SMC}(\varphi_2) \cup (\mathrm{SMC}(\varphi_1) \cap (Q \setminus \delta^{-1}(Q \setminus X)))$

case $\varphi = \mathbf{EG}\varphi$ /* Exists Globally */
  return $\nu X.\mathrm{SMC}(\psi) \cap \delta^{-1}(X)$

case $\varphi = \mathbf{EF}\varphi$ /* Exists Eventually */
  return $\mu X.\mathrm{SMC}(\psi) \cup \delta^{-1}(X)$

Symbolic CTL Model Checking Algorithm

$A = (Q, q_0, \delta, L)$ fixed

Procedure SMC(φ)

case $\varphi \in \mathcal{P}$ /* Atomic Proposition */
  return $L^{-1}(\varphi)$;

case $\varphi = \neg\psi$ /* Negation */
  return $Q \setminus \mathrm{SMC}(\psi)$;

case $\varphi = \mathbf{EX}\psi$ /* Exists Next */
  return $\delta^{-1}(\mathrm{SMC}(\psi))$;

case $\varphi = \mathbf{AX}\psi$ /* Always Next */
  return $Q \setminus \delta^{-1}(Q \setminus \mathrm{SMC}(\psi))$;

case $\varphi = \mathbf{E}\ \psi_1\ \mathbf{UNTIL}\ \psi_2$ /* Exists Until */
  return $\mu X.\mathrm{SMC}(\varphi_2) \cup (\mathrm{SMC}(\varphi_1) \cap \delta^{-1}(X))$

case $\varphi = \mathbf{A}\ \psi_1\ \mathbf{UNTIL}\ \psi_2$ /* Always Until */
  return $\mu X.\mathrm{SMC}(\varphi_2) \cup (\mathrm{SMC}(\varphi_1) \cap (Q \setminus \delta^{-1}(Q \setminus X)))$

case $\varphi = \mathbf{EG}\varphi$ /* Exists Globally */
  return $\nu X.\mathrm{SMC}(\psi) \cap \delta^{-1}(X)$

case $\varphi = \mathbf{EF}\varphi$ /* Exists Eventually */
  return $\mu X.\mathrm{SMC}(\psi) \cup \delta^{-1}(X)$

Symbolic CTL Model Checking Algorithm

Procedure SMC(φ)

case $\varphi \in \mathcal{P}$ /* Atomic Proposition */
  return $L^{-1}(\varphi)$;
case $\varphi = \neg\psi$ /* Negation */
  return $Q \setminus$ SMC(ψ);
case $\varphi =$ **EX**ψ /* Exists Next */
  return δ^{-1}(SMC(ψ));
case $\varphi =$ **AX**ψ /* Always Next */
  return $Q \setminus \delta^{-1}(Q \setminus$ SMC(ψ));
case $\varphi =$ **E** ψ_1 **UNTIL** ψ_2 /* Exists Until */
  return $\mu X.$SMC(φ_2) $\cup$ (SMC(φ_1) $\cap \delta^{-1}(X)$)
case $\varphi =$ **A** ψ_1 **UNTIL** ψ_2 /* Always Until */
  return $\mu X.$SMC(φ_2) $\cup$ (SMC(φ_1) $\cap (Q \setminus \delta^{-1}(Q \setminus X)))$
case $\varphi =$ **EG**φ /* Exists Globally */
  return $\nu X.$SMC(ψ) $\cap \delta^{-1}(X)$
case $\varphi =$ **EF**φ /* Exists Eventually */
  return $\mu X.$SMC(ψ) $\cup \delta^{-1}(X)$

***A* = (*Q*, q_0, δ, *L*)** fixed

Procedure FixPoint(f, ι)

$P := \iota$;
$N := f(P)$;
while $P \neq N$ **do**
  $P := N$;
  $N := f(N)$;
return N;

Symbolic CTL Model Checking Algorithm

Procedure SMC(φ)

case $\varphi \in \mathcal{P}$ /* Atomic Proposition */
  return $L^{-1}(\varphi)$;
case $\varphi = \neg\psi$ /* Negation */
  return $Q \setminus$ SMC(ψ);
case $\varphi =$ **EX**ψ /* Exists Next */
  return δ^{-1}(SMC(ψ));
case $\varphi =$ **AX**ψ /* Always Next */
  return $Q \setminus \delta^{-1}(Q \setminus$ SMC(ψ));
case $\varphi =$ **E** ψ_1 **UNTIL** ψ_2 /* Exists Until */
  return $\mu X.$SMC(φ_2) $\cup$ (SMC(φ_1) $\cap\, \delta^{-1}(X)$)
case $\varphi =$ **A** ψ_1 **UNTIL** ψ_2 /* Always Until */
  return $\mu X.$SMC(φ_2) $\cup$ (SMC(φ_1) $\cap\, (Q \setminus \delta^{-1}(Q \setminus X))$)
case $\varphi =$ **EG**φ /* Exists Globally */
  return $\nu X.$SMC(ψ) $\cap\, \delta^{-1}(X)$
case $\varphi =$ **EF**φ /* Exists Eventually */
  return $\mu X.$SMC(ψ) $\cup\, \delta^{-1}(X)$

***A* = (*Q*, *q*$_0$, δ, *L*) fixed**

Procedure FixPoint(f, ι)

$P := \iota$;
$N := f(P)$;
while $P \neq N$ **do**
  $P := N$;
  $N := f(N)$;
return N;

Symbolic CTL Model Checking Algorithm

Procedure SMC(φ)

case $\varphi \in \mathcal{P}$ /* Atomic Proposition */
  return $L^{-1}(\varphi)$;

case $\varphi = \neg\psi$ /* Negation */
  return $Q \setminus \mathrm{SMC}(\psi)$;

case $\varphi = \mathbf{EX}\psi$ /* Exists Next */
  return $\delta^{-1}(\mathrm{SMC}(\psi))$;

case $\varphi = \mathbf{AX}\psi$ /* Always Next */
  return $Q \setminus \delta^{-1}(Q \setminus \mathrm{SMC}(\psi))$;

case $\varphi = \mathbf{E}\ \psi_1\ \mathbf{UNTIL}\ \psi_2$ /* Exists Until */
  return $\mu X.\mathrm{SMC}(\varphi_2) \cup (\mathrm{SMC}(\varphi_1) \cap \delta^{-1}(X))$

case $\varphi = \mathbf{A}\ \psi_1\ \mathbf{UNTIL}\ \psi_2$ /* Always Until */
  return $\mu X.\mathrm{SMC}(\varphi_2) \cup (\mathrm{SMC}(\varphi_1) \cap (Q \setminus \delta^{-1}(Q \setminus X)))$

case $\varphi = \mathbf{EG}\varphi$ /* Exists Globally */
  return $\nu X.\mathrm{SMC}(\psi) \cap \delta^{-1}(X)$

case $\varphi = \mathbf{EF}\varphi$ /* Exists Eventually */
  return $\mu X.\mathrm{SMC}(\psi) \cup \delta^{-1}(X)$

$A = (Q, q_0, \delta, L)$ fixed

Procedure FixPoint(f, ι)

$P := \iota$;
$N := f(P)$;
while $P \neq N$ **do**
  $P := N$;
  $N := f(N)$;
return N;

$$A \models \varphi \iff q_0 \models \varphi$$
$$\iff q_0 \in \mathrm{SMC}(\varphi)$$

Symbolic CTL Model Checking Algorithm

Procedure SMC(φ)

```
case φ ∈ 𝒫                                   /* Atomic Proposition */
   return L⁻¹(φ);
case φ = ¬ψ                                  /* Negation */
   return Q \ SMC(ψ);
case φ = EXψ                                 /* Exists Next */
   return δ⁻¹(SMC(ψ));
case φ = AXψ                                 /* Always Next */
   return Q \ δ⁻¹(Q \ SMC(ψ));
case φ = E ψ1 UNTIL ψ2                       /* Exists Until */
   return μX.SMC(φ2) ∪ (SMC(φ1) ∩ δ⁻¹(X))
case φ = A ψ1 UNTIL ψ2                       /* Always Until */
   return μX.SMC(φ2) ∪ (SMC(φ1) ∩ (Q \ δ⁻¹(Q \ X)))
case φ = EGφ                                 /* Exists Globally */
   return νX.SMC(ψ) ∩ δ⁻¹(X)
case φ = EFφ                                 /* Exists Eventually */
   return μX.SMC(ψ) ∪ δ⁻¹(X)
```

$A = (Q, q_0, \delta, L)$ fixed

Procedure FixPoint(f, ι)

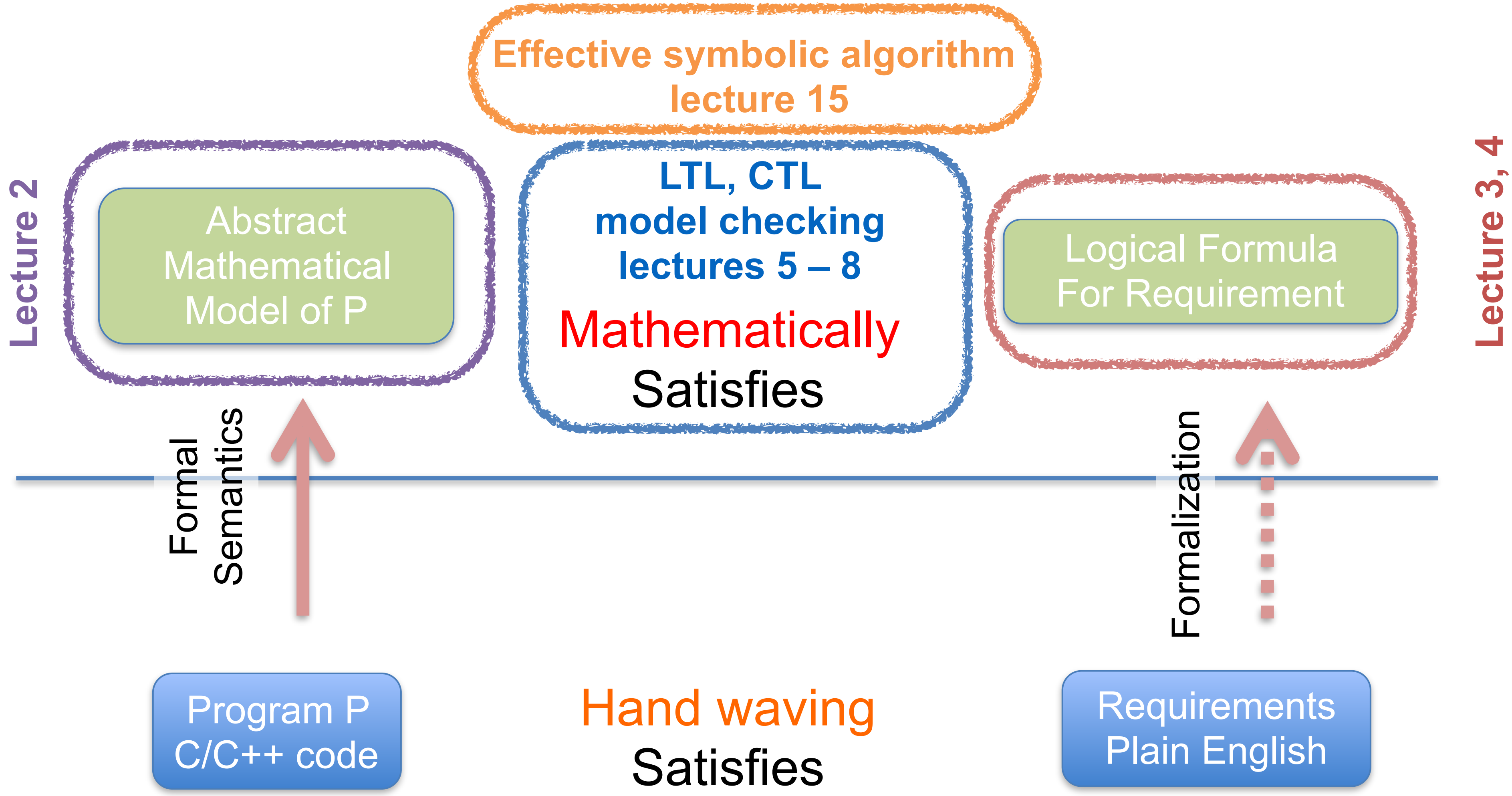
```
P := ι;
N := f(P);
while P ≠ N do
   P := N;
   N := f(N);
return N;
```

$$A \models \varphi \iff q_0 \models \varphi \iff q_0 \in \mathrm{SMC}(\varphi)$$

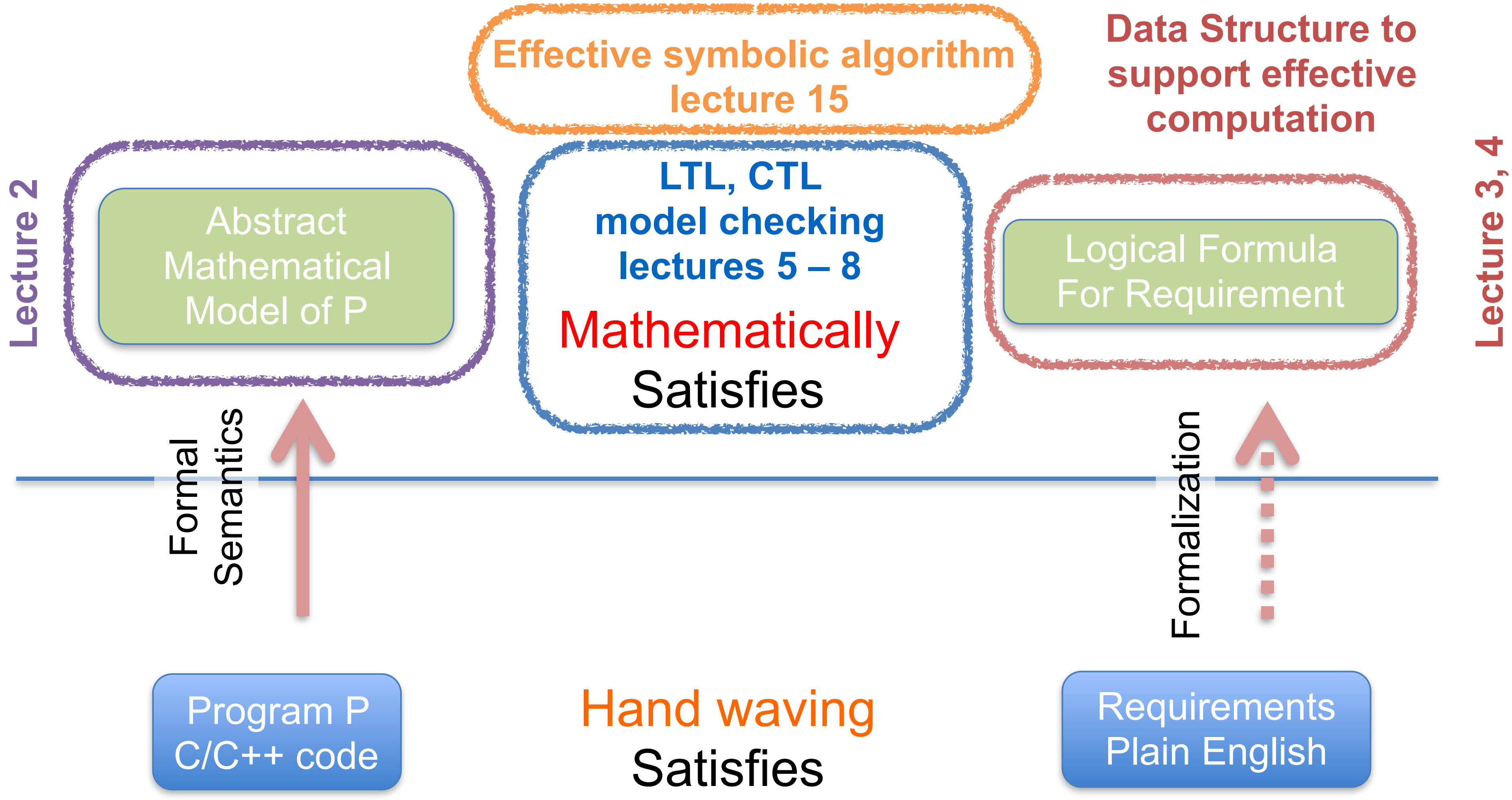
Set operations:

- $\cap, \cup, \setminus$
- Equality test
- δ, δ^{-1}

The Big Picture



The Big Picture



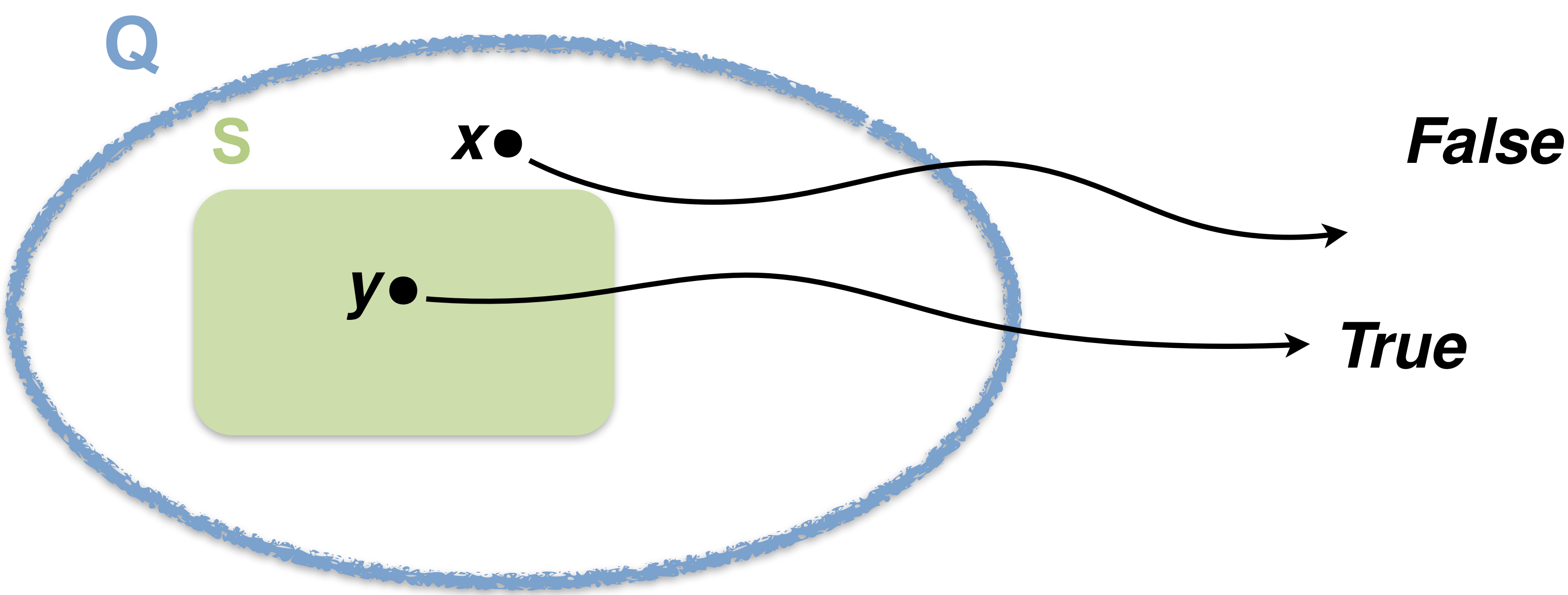
Content for this lecture

1. Sets as Boolean functions
2. Binary Decision Diagrams (BDDs)
3. Operations on BDDs
4. Properties and transition relation
5. Model Checking with BDDs

Sets as Boolean Functions

Characteristic function of a set

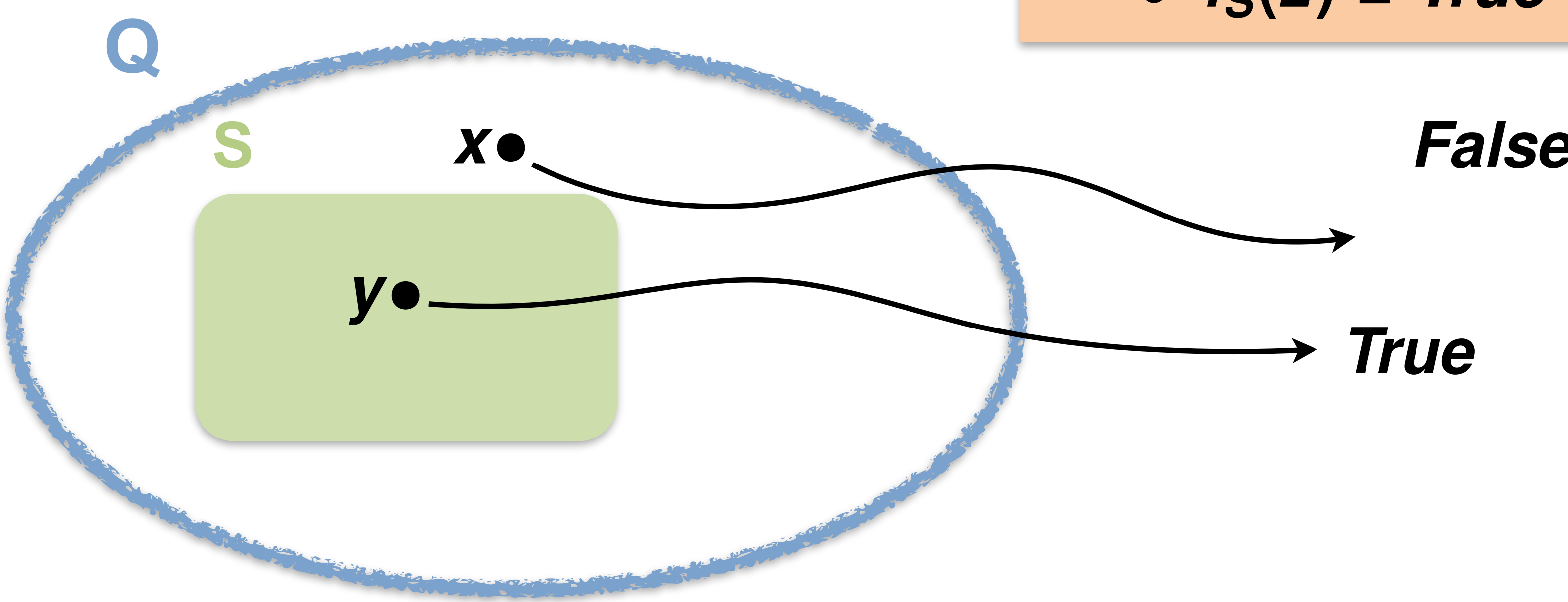
Characteristic Functions



Characteristic Functions

Characteristic function f_S of S

- $f_S(z)$ = ***False*** for $z \notin S$
- $f_S(z)$ = ***True*** for $z \in S$



Boolean Encoding

Finite set $\boldsymbol{E}$

Let $\boldsymbol{n = \log_2 |E|}$

Boolean Encoding

Finite set ***E***

Let $\boldsymbol{n = \log_2 |E|}$

Given $\boldsymbol{S} \subseteq \boldsymbol{E}$:

Boolean Encoding

Finite set $\boldsymbol{E}$

Let $\boldsymbol{n} = \log_2 |\boldsymbol{E}|$

Given $\boldsymbol{S} \subseteq \boldsymbol{E}$:

$$f_S : \mathbb{B}^n \longrightarrow \mathbb{B}$$

Boolean Encoding

Finite set E

Let $n = \log_2 |E|$

$E = \{0, 1, 2, 3, 4, 5, 6, 7\}$

3 bits: b_3, b_2, b_1

Given $S \subseteq E$:

$$f_S : \mathbb{B}^n \longrightarrow \mathbb{B}$$

Boolean Encoding

Finite set E

Let $n = \log_2 |E|$

$E = \{0, 1, 2, 3, 4, 5, 6, 7\}$

3 bits: b_3, b_2, b_1

Given $S \subseteq E$:

$$f_S : \mathbb{B}^n \longrightarrow \mathbb{B}$$

	b_3	b_2	b_1
6	1	1	0
3	0	1	1

Boolean Encoding

Finite set $\boldsymbol{E}$

Let $n = \log_2 |E|$

$\mathbf{E} = \{0, 1, 2, 3, 4, 5, 6, 7\}$

3 bits: b_3, b_2, b_1

$S = \{k \mid k \mod 2 = 0\}$

$f_s : \mathbb{B}^3 \longrightarrow \mathbb{B}$

$f_s(b_1, b_2, b_3) = (b_1 = 0)$

Given $S \subseteq E$:

$$f_S : \mathbb{B}^n \longrightarrow \mathbb{B}$$

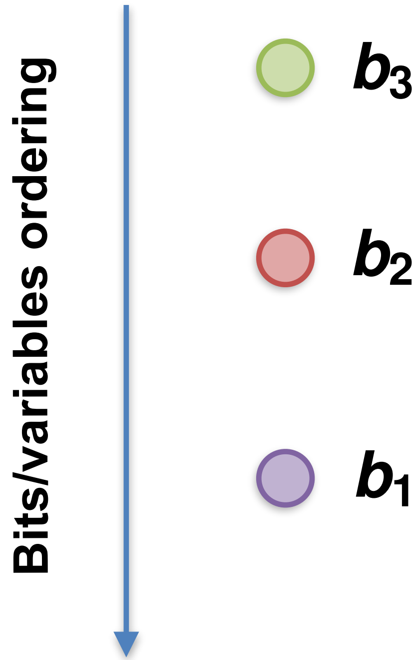
	b_3	b_2	b_1
6	**1**	**1**	**0**
3	**0**	**1**	**1**

Ordered Binary Decision Trees

$S = \{k \mid k \mod 2 = 0\}$

$f_S : \mathbb{B}^3 \longrightarrow \mathbb{B}$

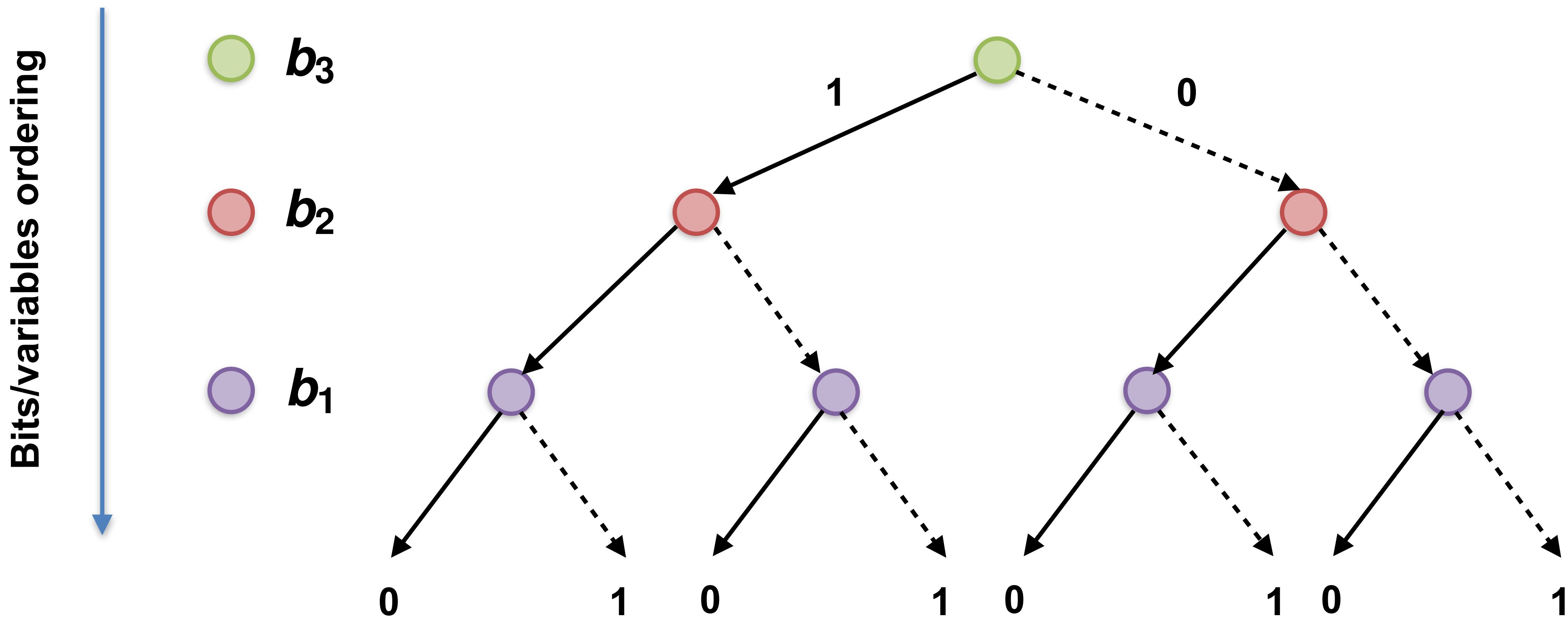
Ordered Binary Decision Trees



$S = \{k \mid k \mod 2 = 0\}$

$f_s : \mathbb{B}^3 \longrightarrow \mathbb{B}$

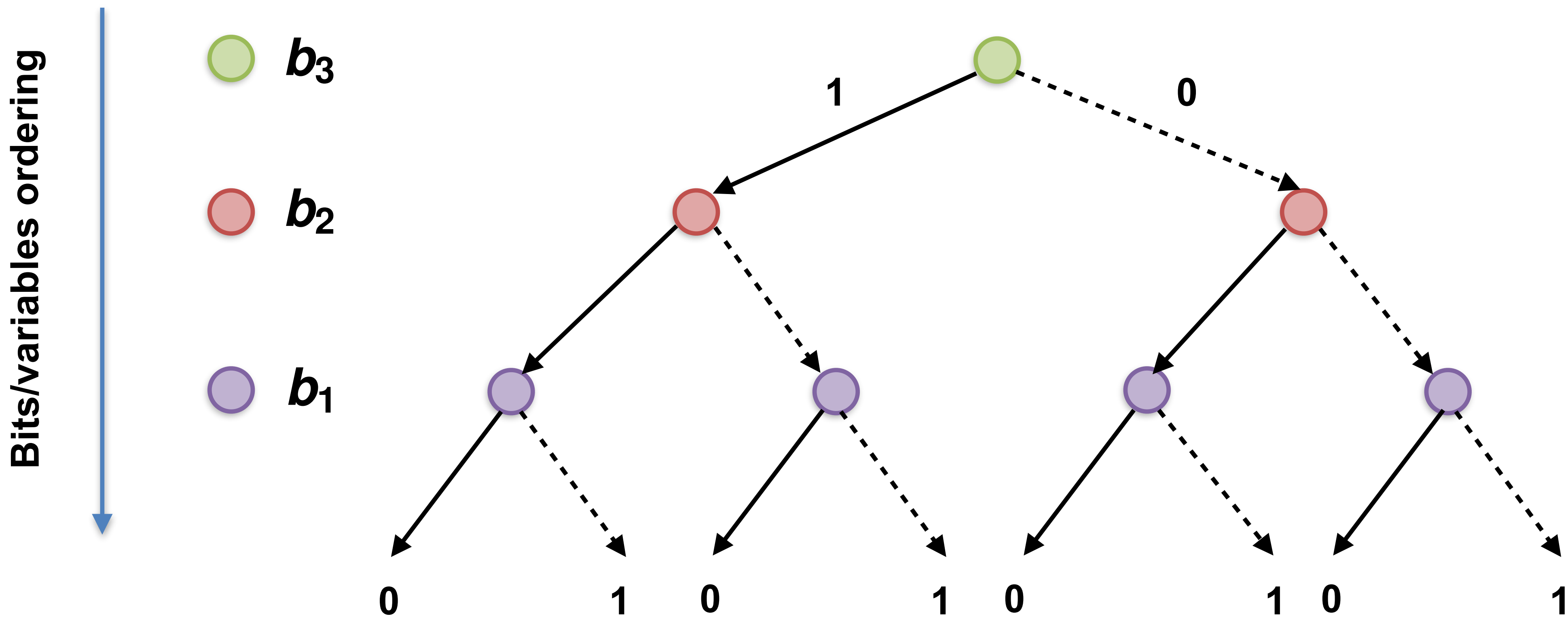
Ordered Binary Decision Trees



$S = \{k \mid k \mod 2 = 0\}$

$f_s : \mathbb{B}^3 \longrightarrow \mathbb{B}$

Ordered Binary Decision Trees

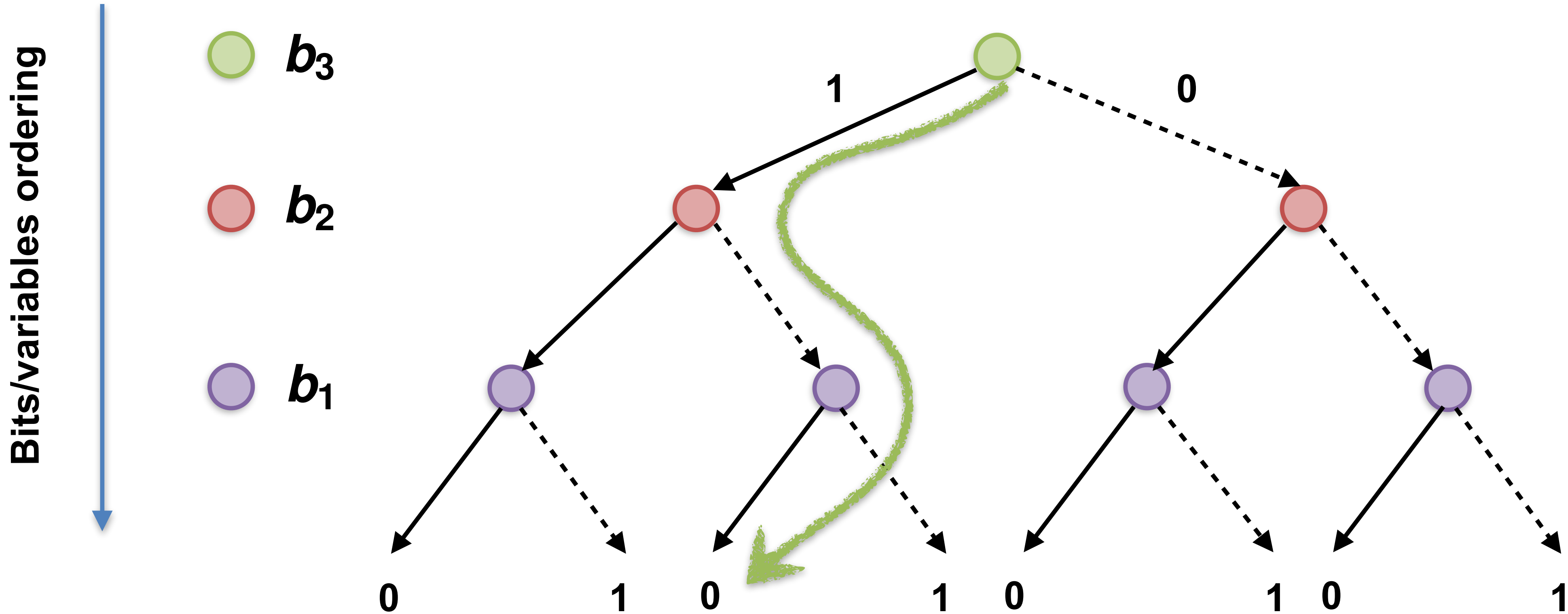


$S = \{k \mid k \mod 2 = 0\}$

$f_s : \mathbb{B}^3 \longrightarrow \mathbb{B}$

$f_s(b_1, b_2, b_3) = (b_1 = 0)$

Ordered Binary Decision Trees

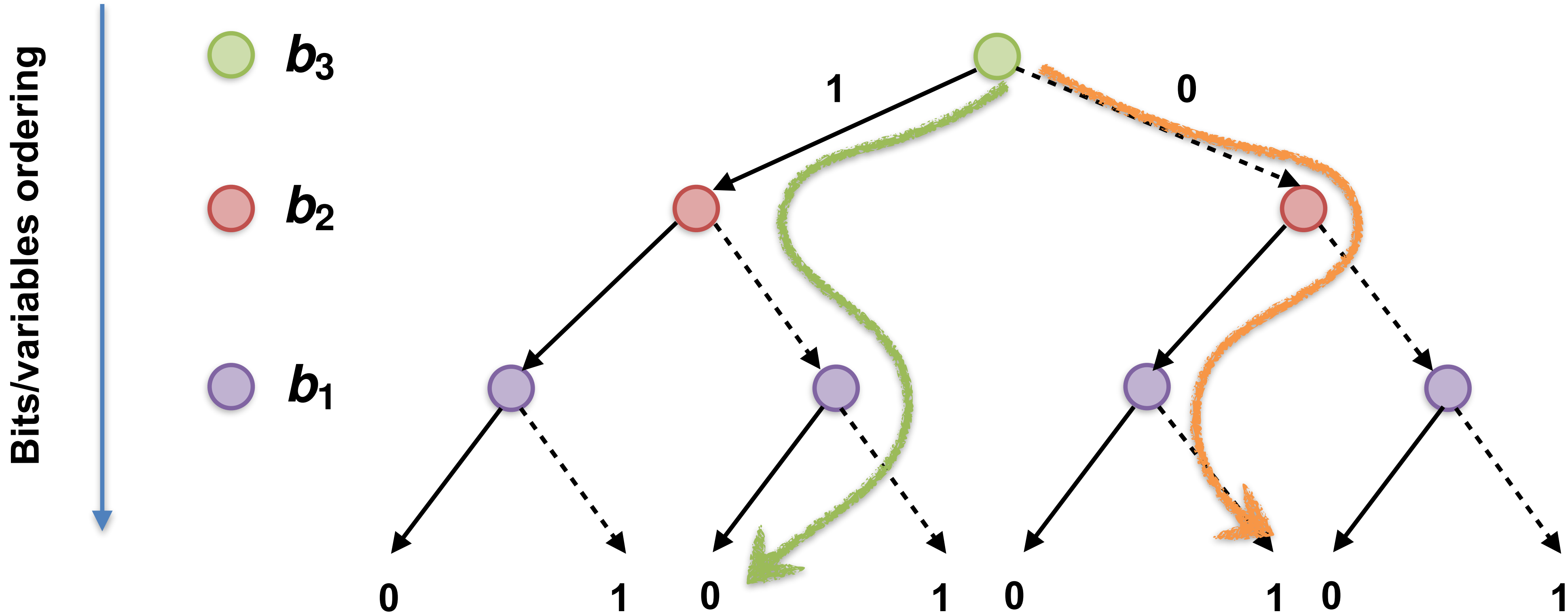


$S = \{k \mid k \mod 2 = 0\}$

$f_s : \mathbb{B}^3 \longrightarrow \mathbb{B}$

$f_s(b_1, b_2, b_3) = (b_1 = 0)$

Ordered Binary Decision Trees

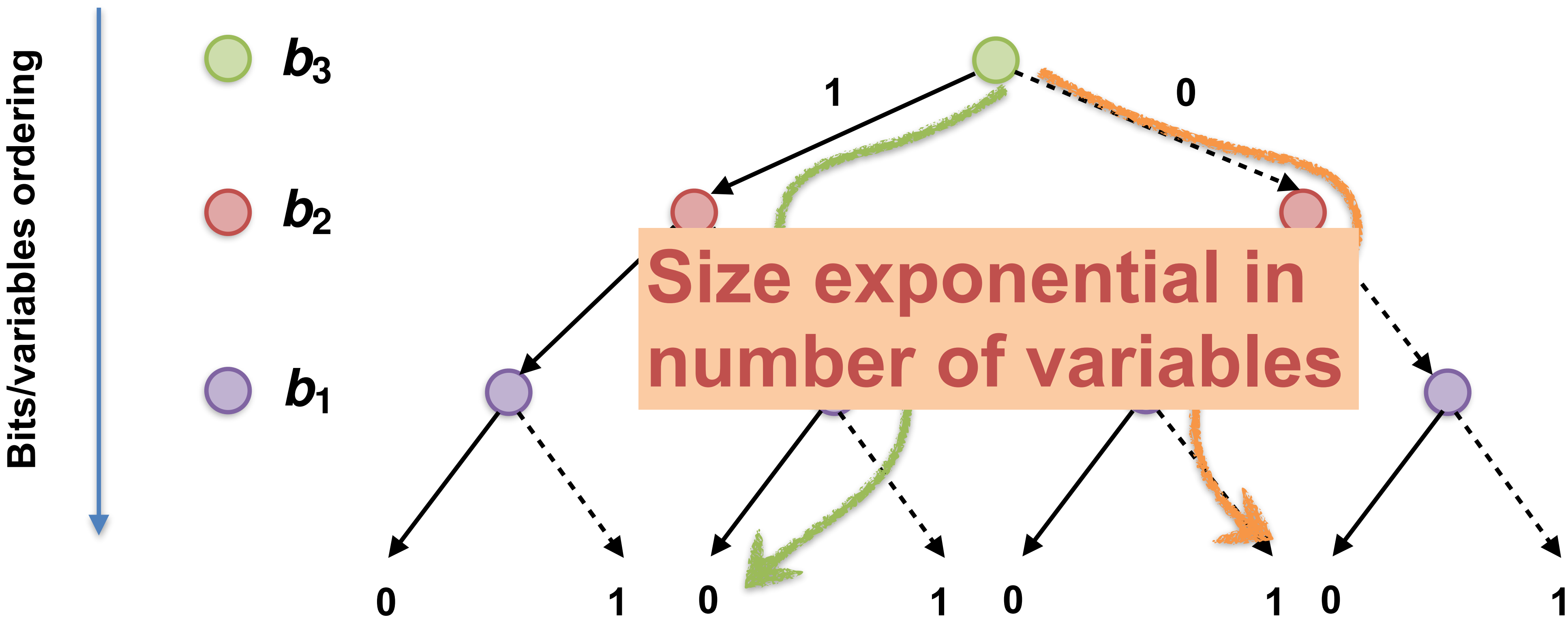


$S = \{k \mid k \mod 2 = 0\}$

$f_s : \mathbb{B}^3 \longrightarrow \mathbb{B}$

$f_s(b_1, b_2, b_3) = (b_1 = 0)$

Ordered Binary Decision Trees



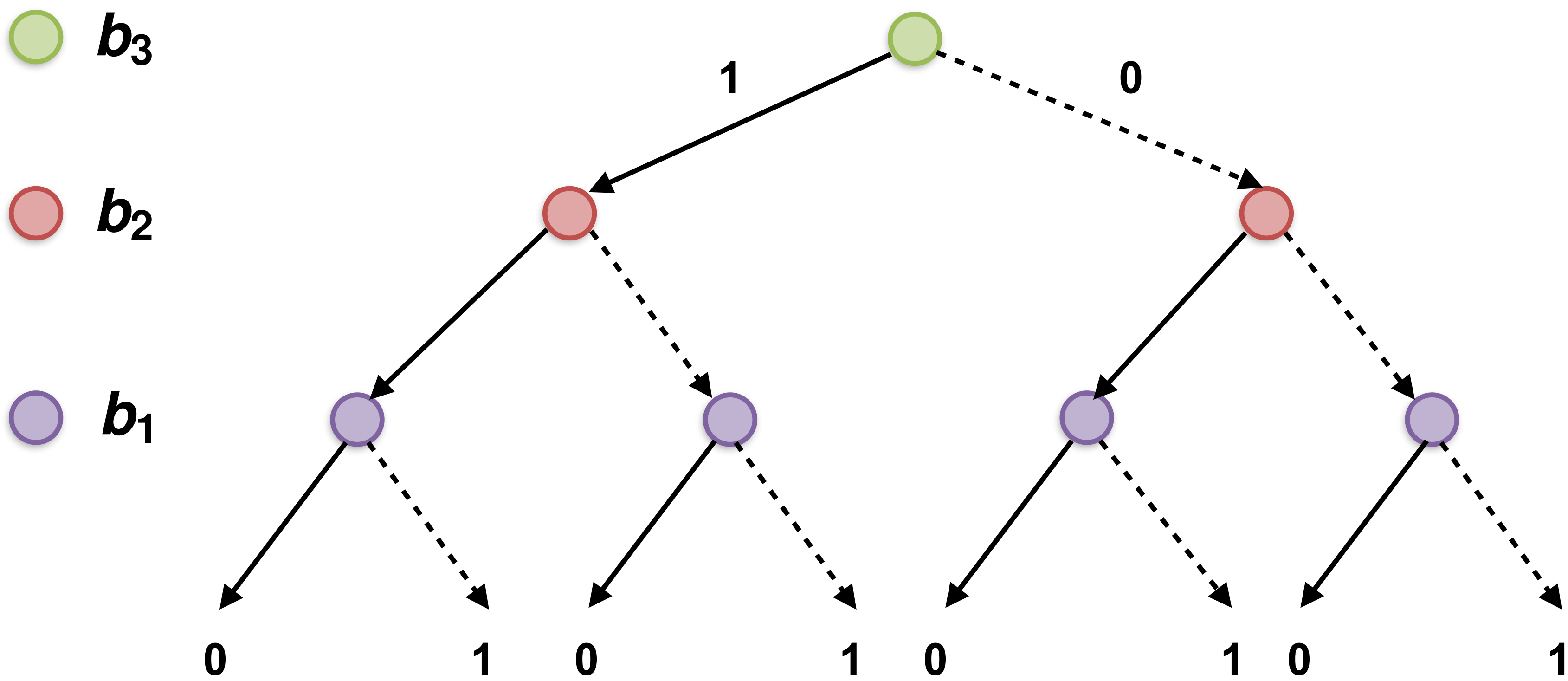
$S = \{k \mid k \mod 2 = 0\}$

$f_s : \mathbb{B}^3 \longrightarrow \mathbb{B}$

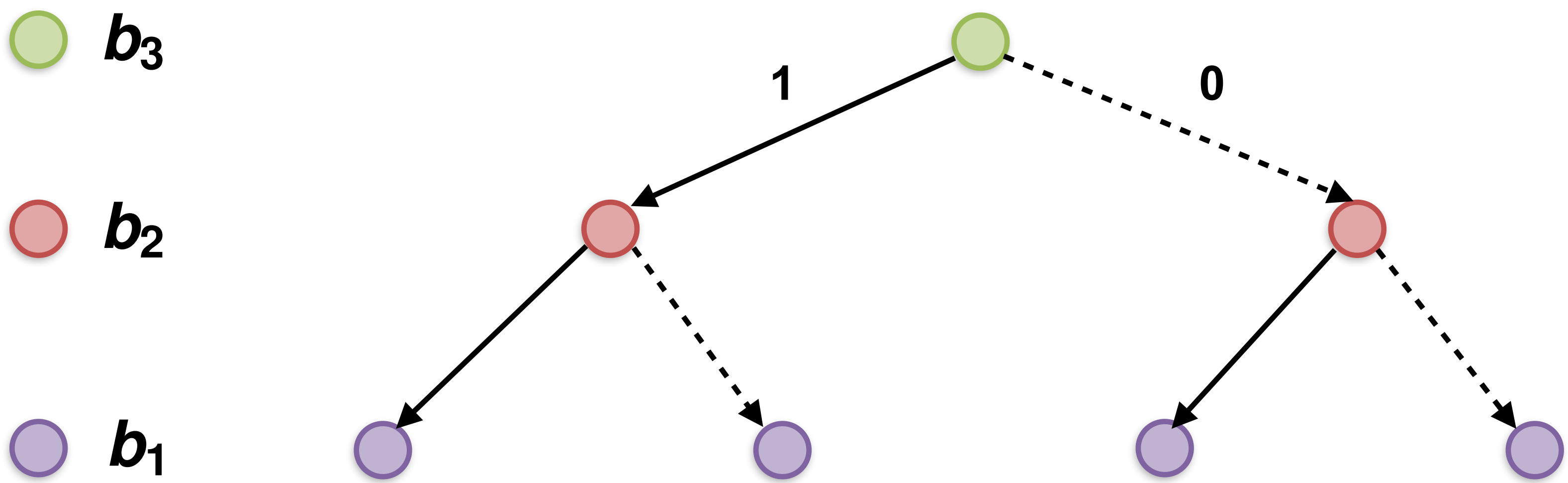
$f_s(b_1, b_2, b_3) = (b_1 = 0)$

Binary Decision Diagrams

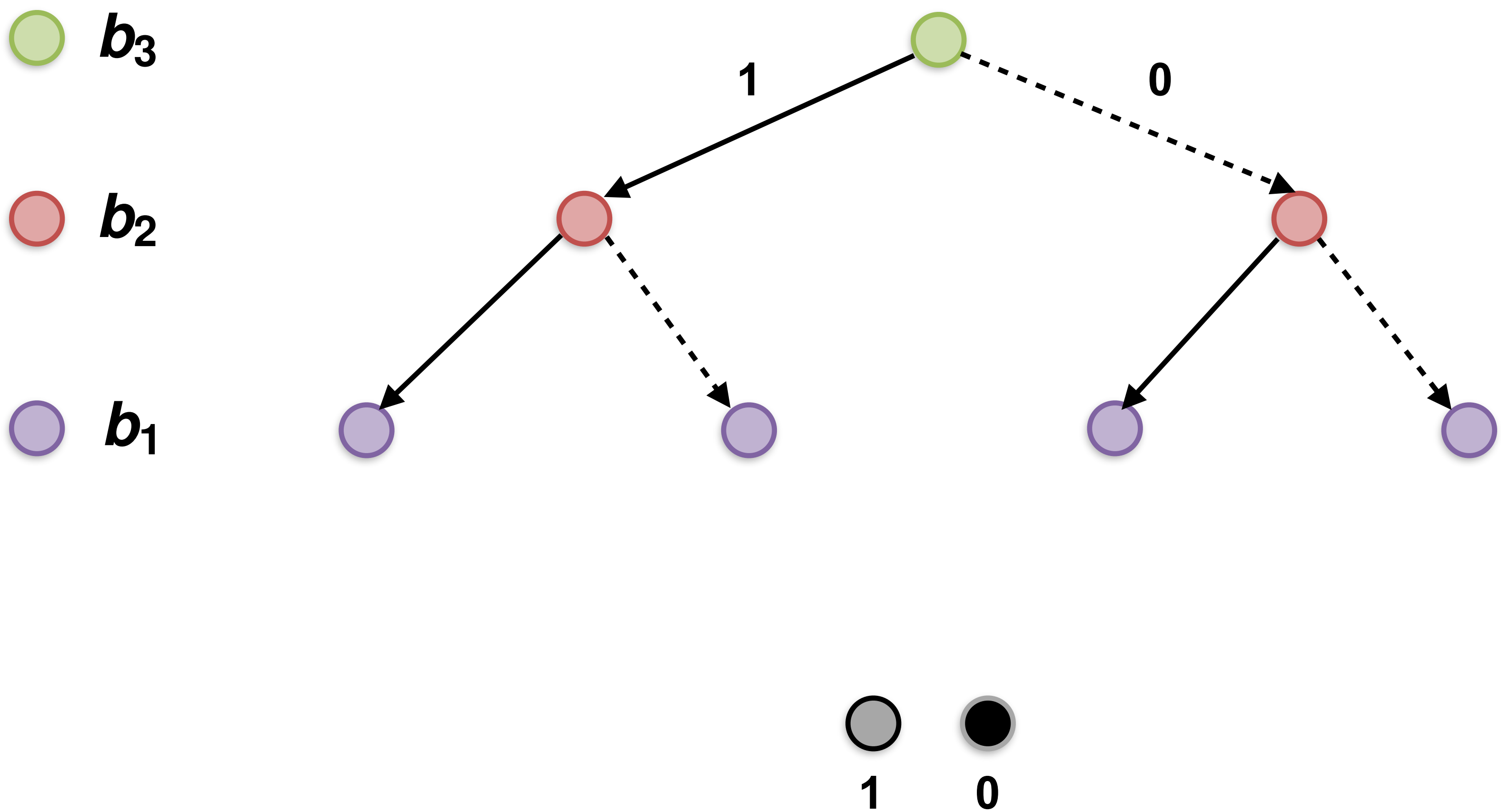
Ordered (Reduced) Binary Decision Diagram



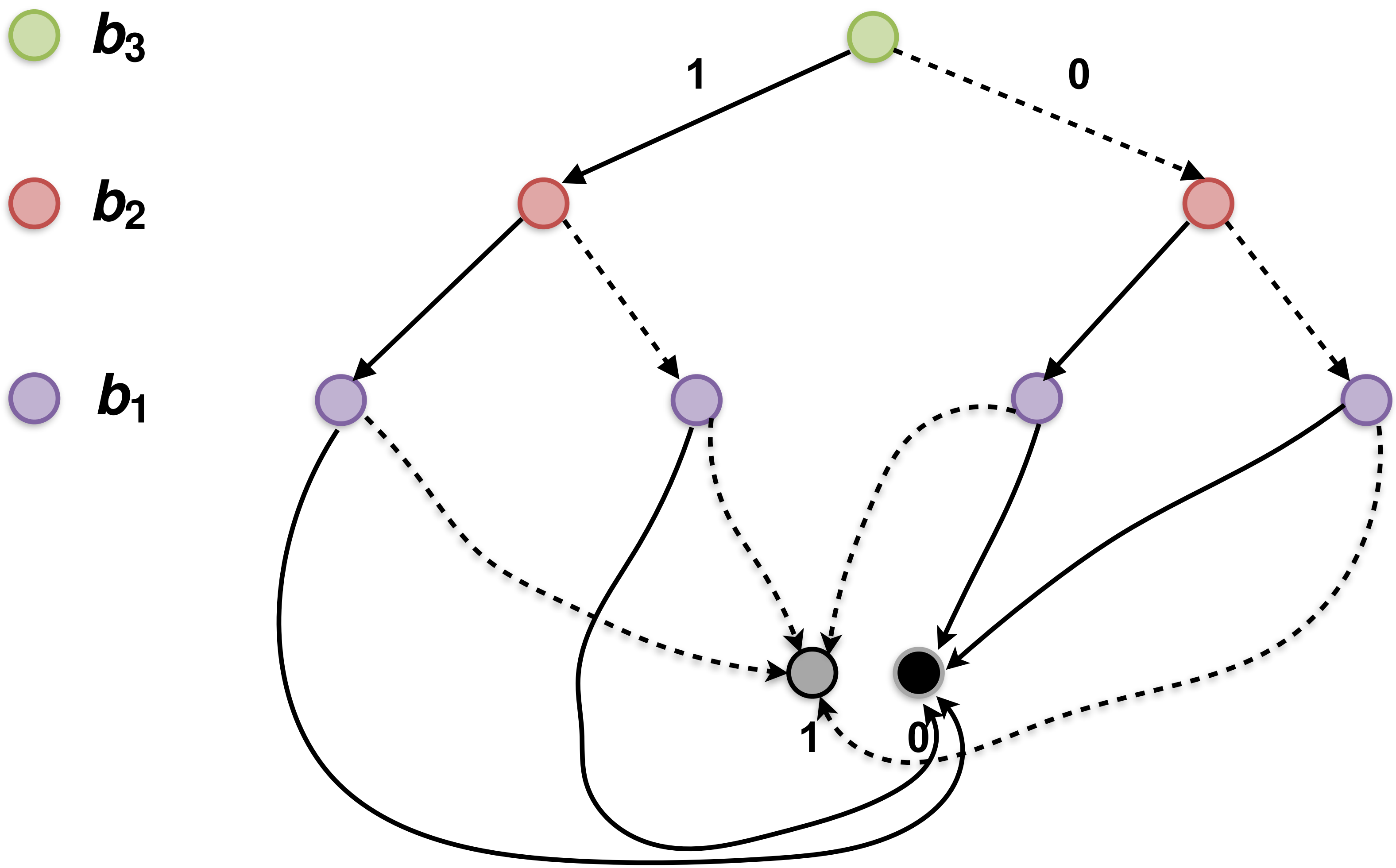
Ordered (Reduced) Binary Decision Diagram



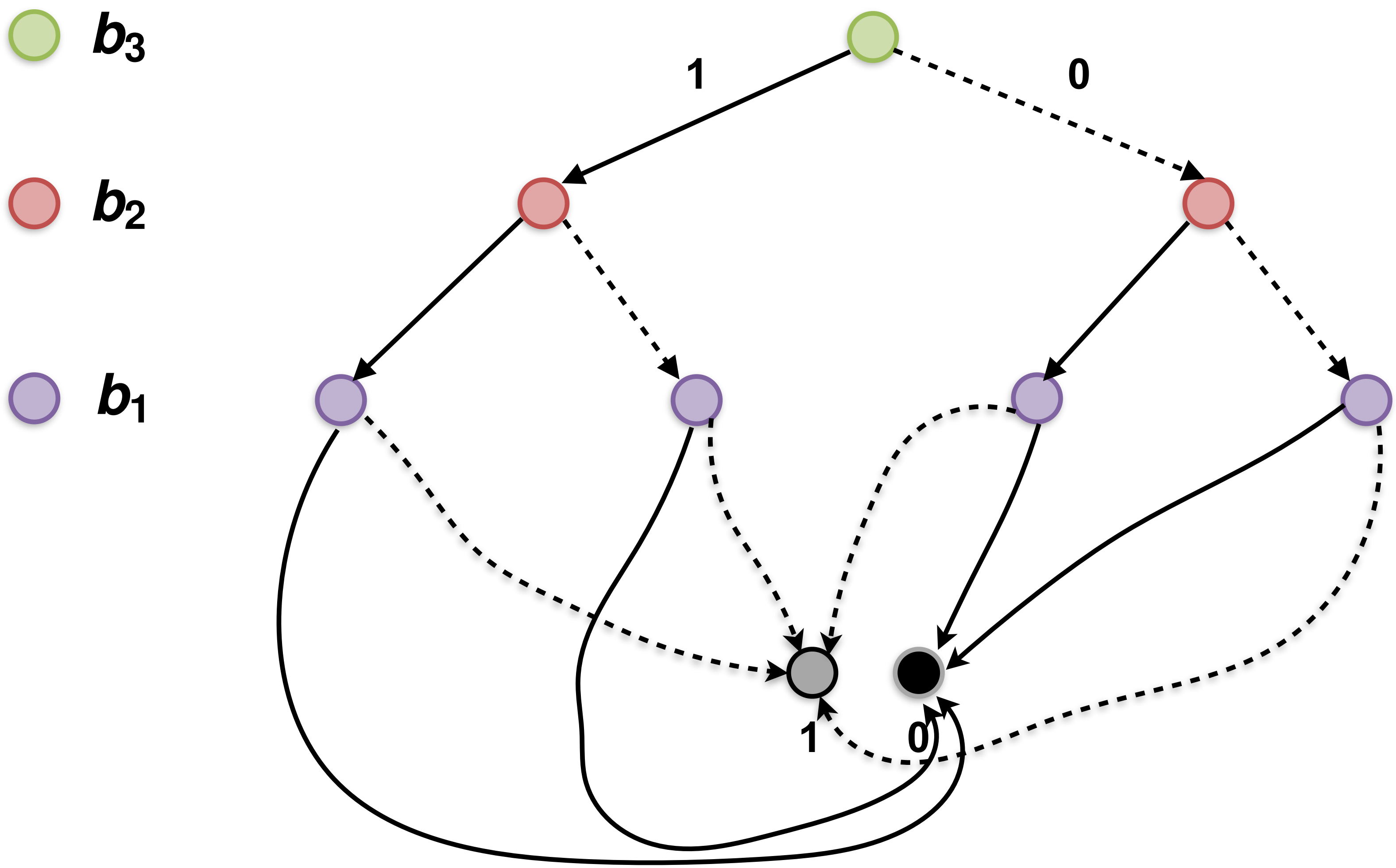
Ordered (Reduced) Binary Decision Diagram



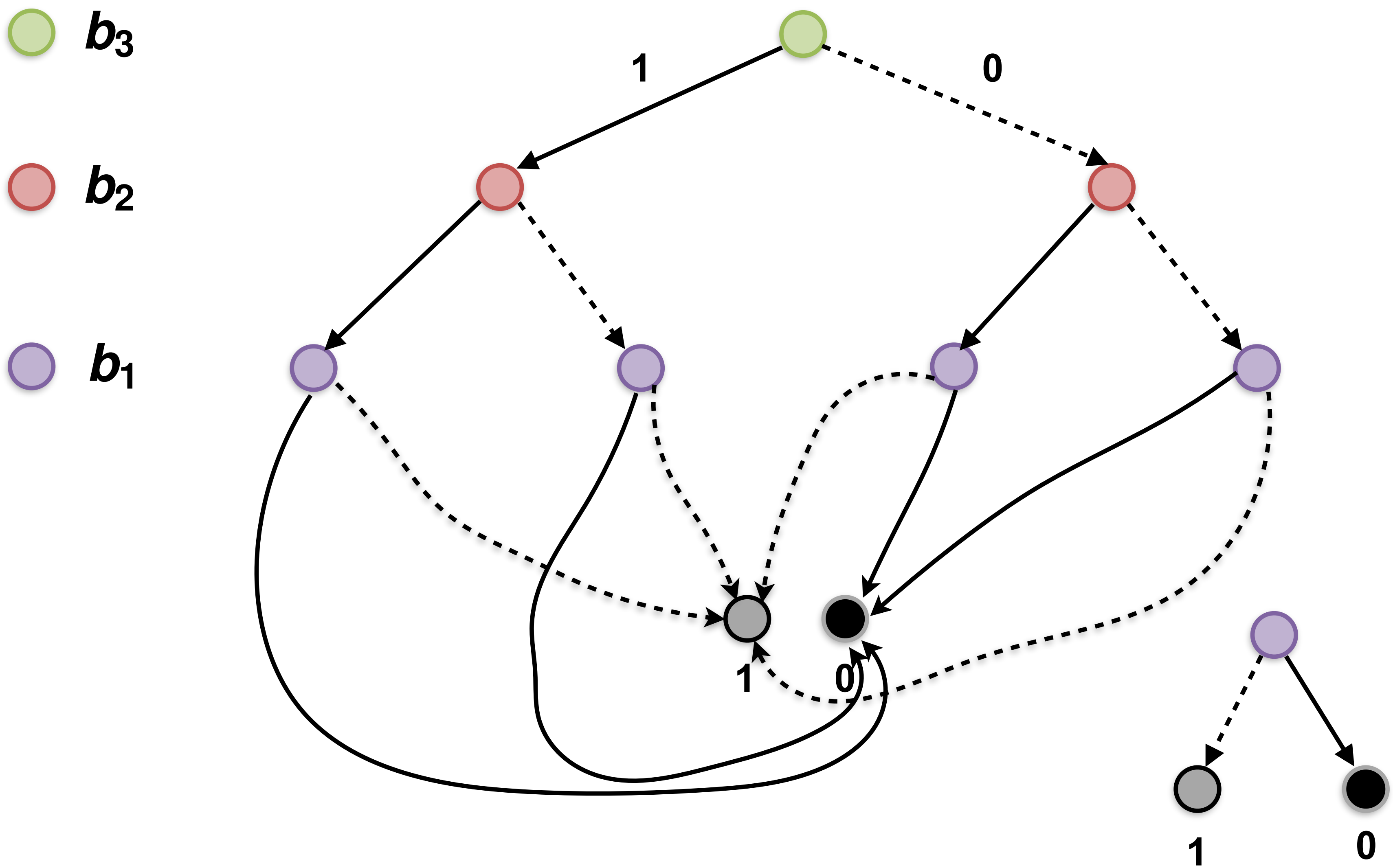
Ordered (Reduced) Binary Decision Diagram



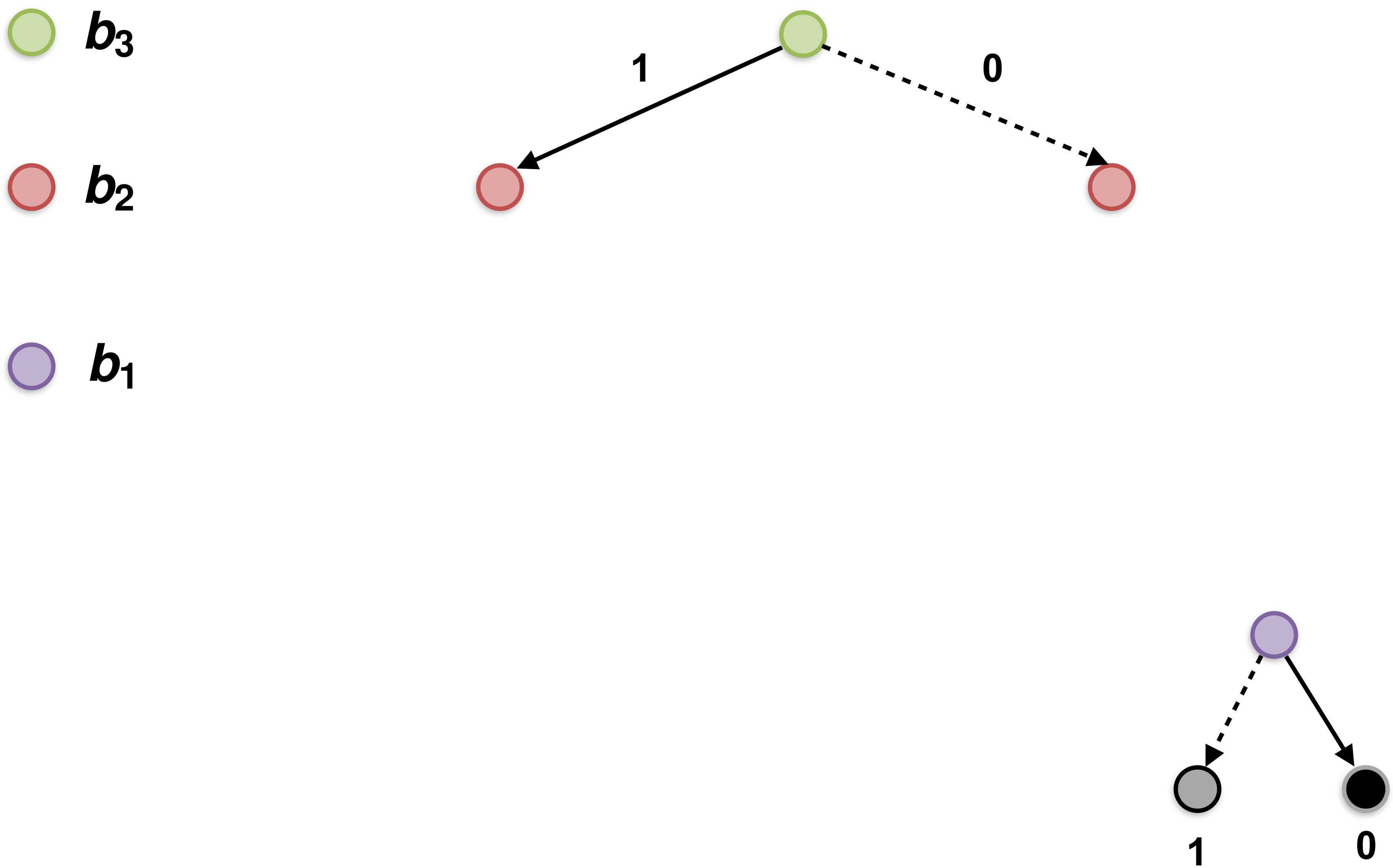
Ordered (Reduced) Binary Decision Diagram



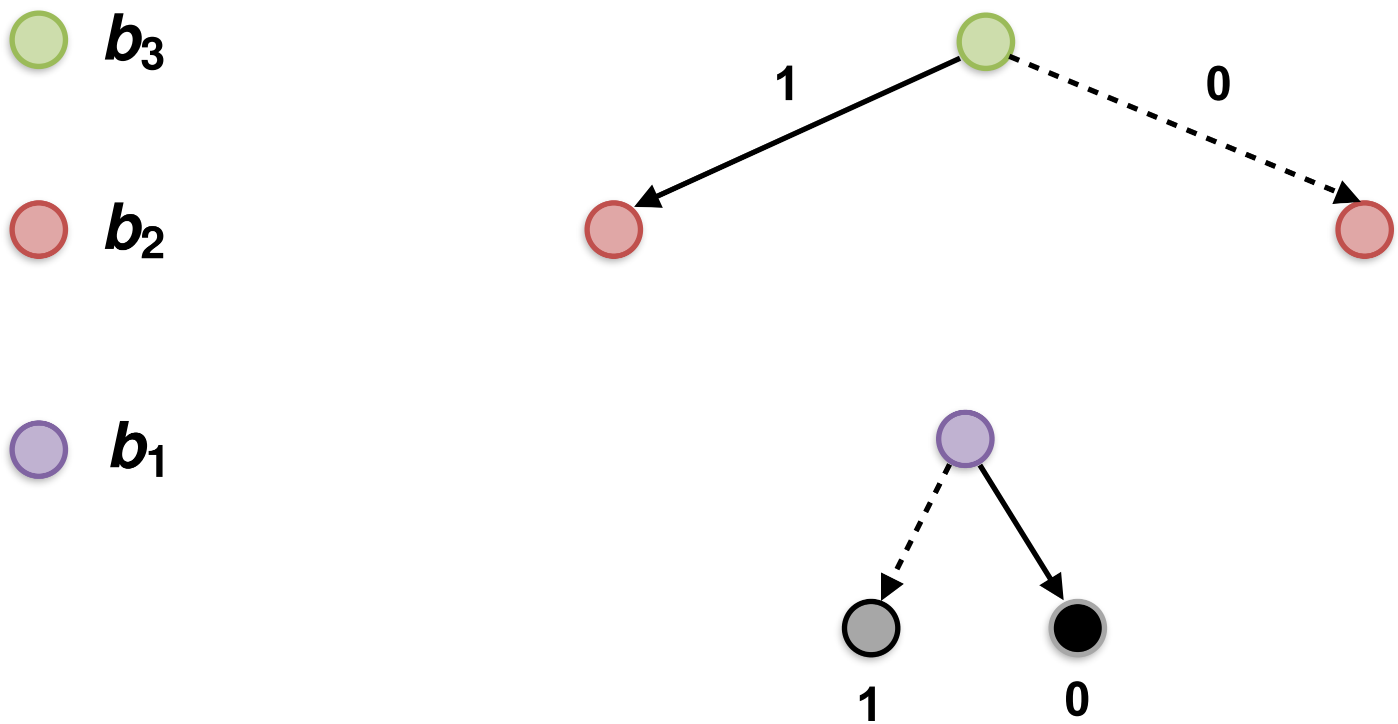
Ordered (Reduced) Binary Decision Diagram



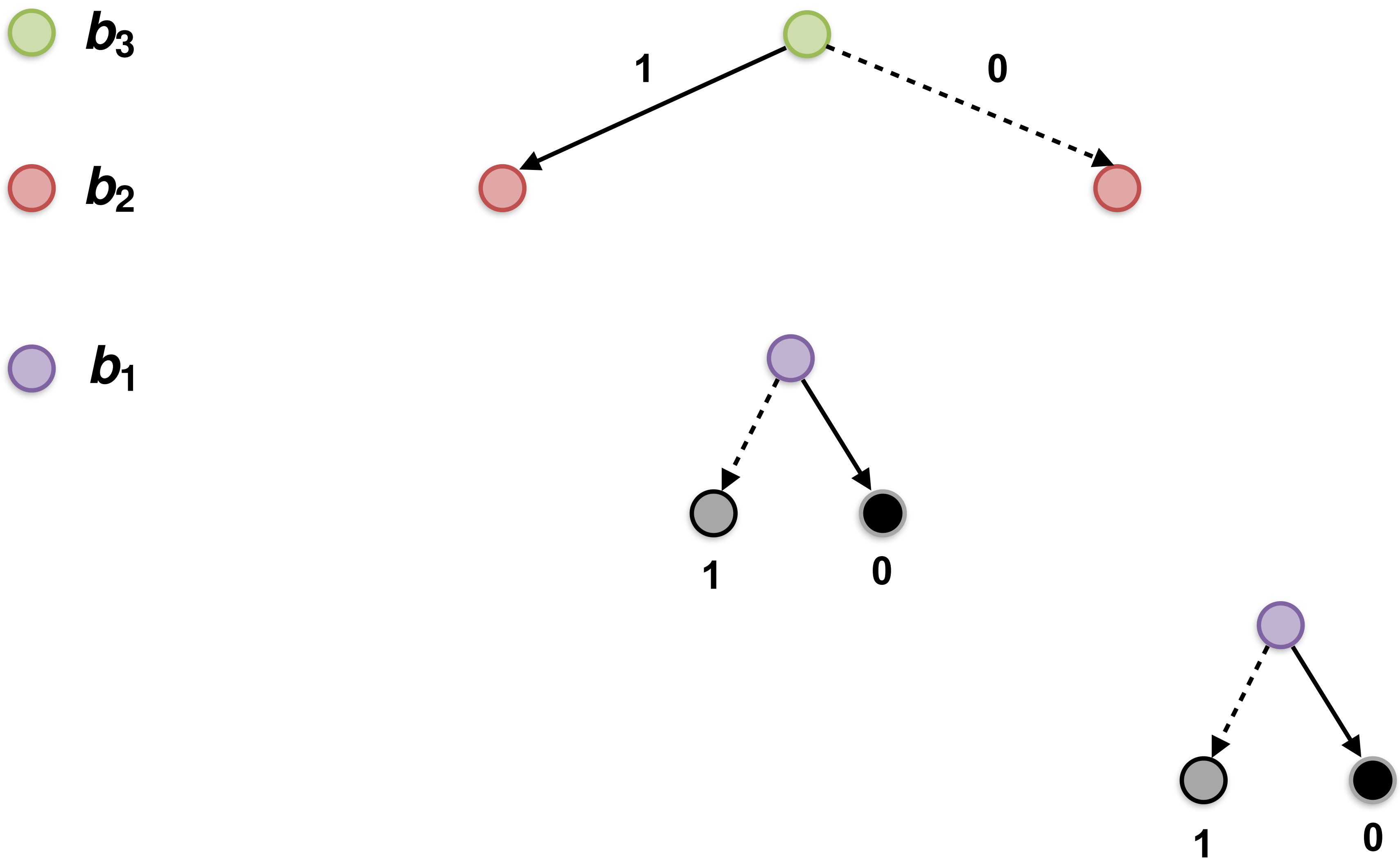
Ordered (Reduced) Binary Decision Diagram



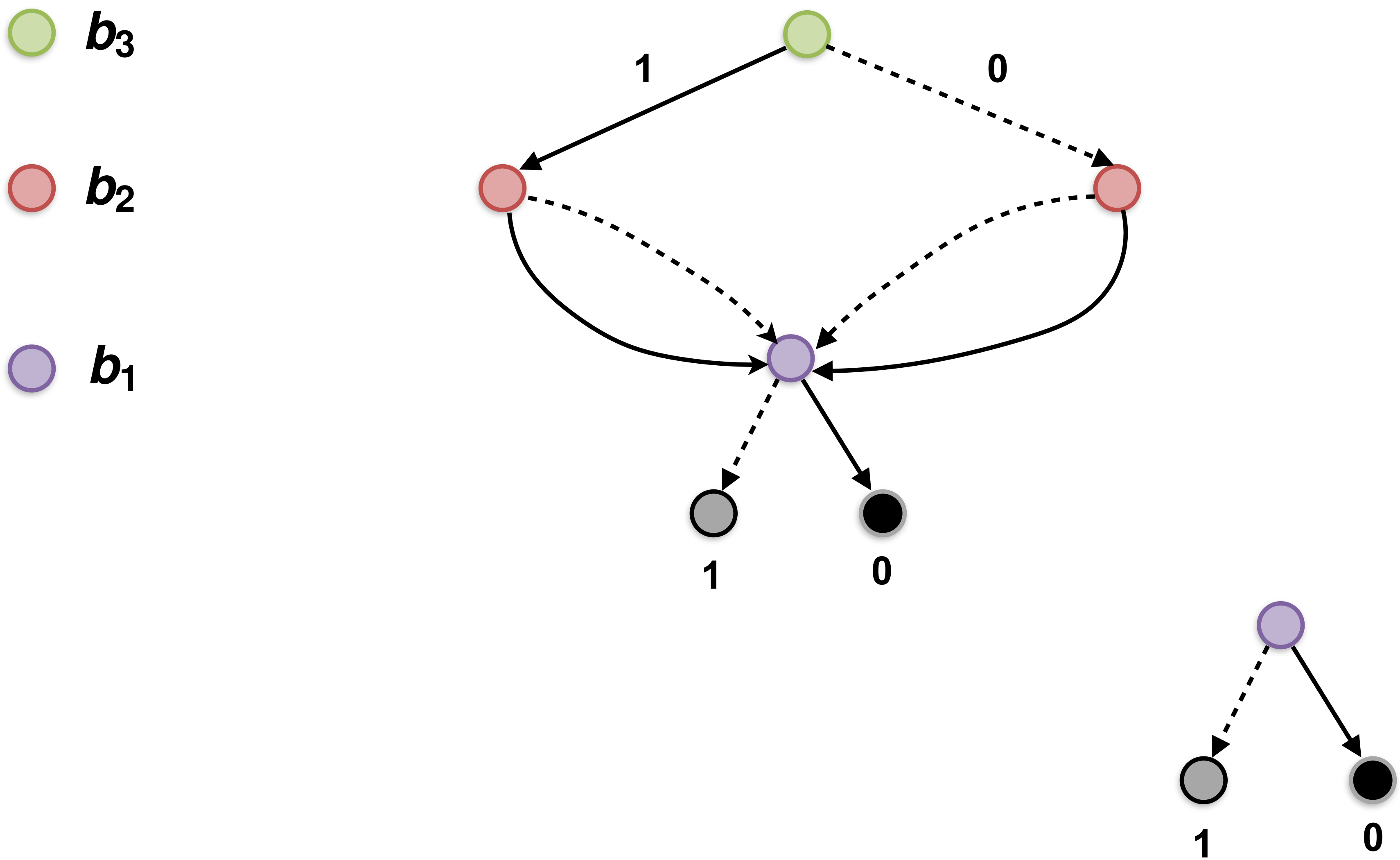
Ordered (Reduced) Binary Decision Diagram



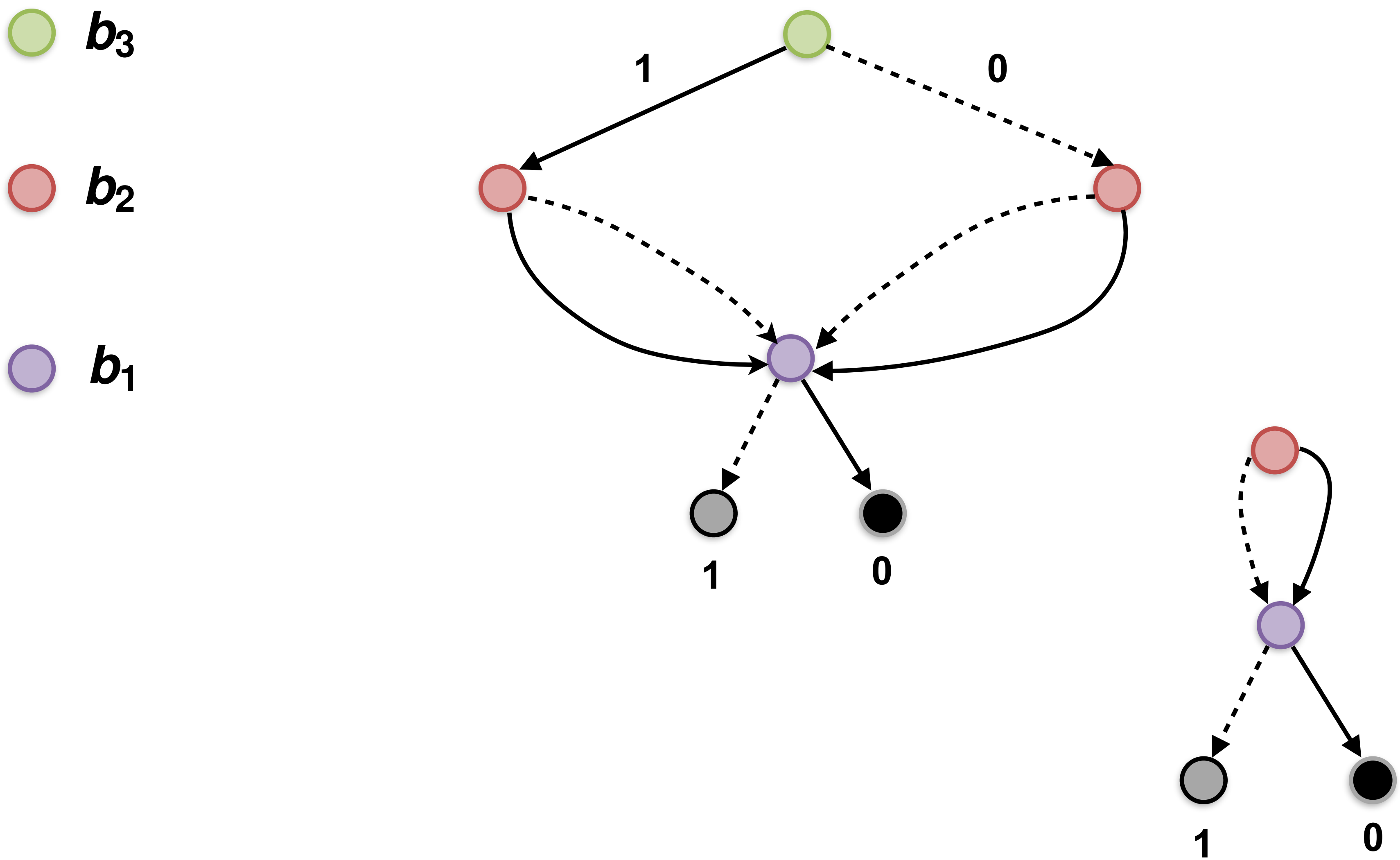
Ordered (Reduced) Binary Decision Diagram



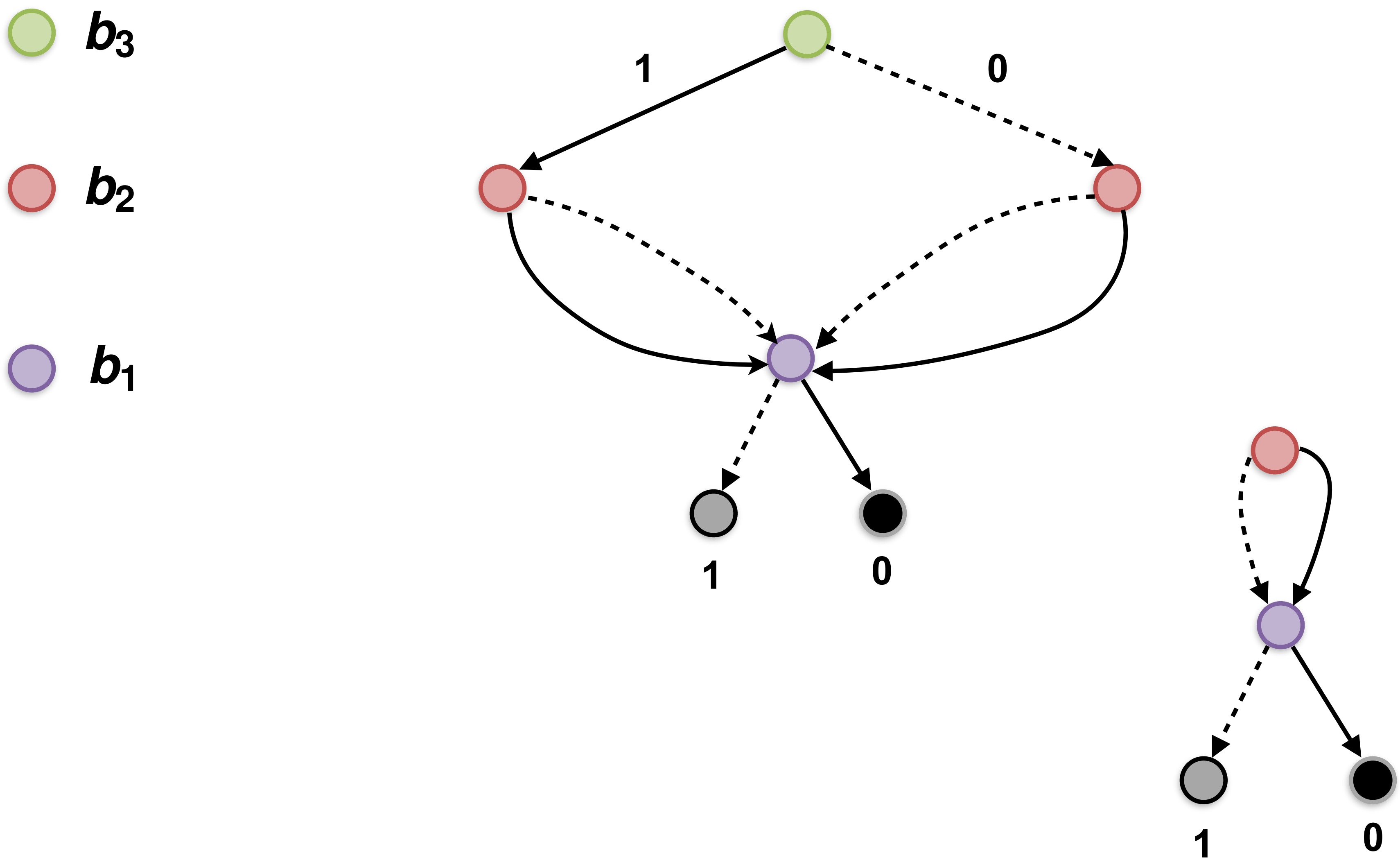
Ordered (Reduced) Binary Decision Diagram



Ordered (Reduced) Binary Decision Diagram

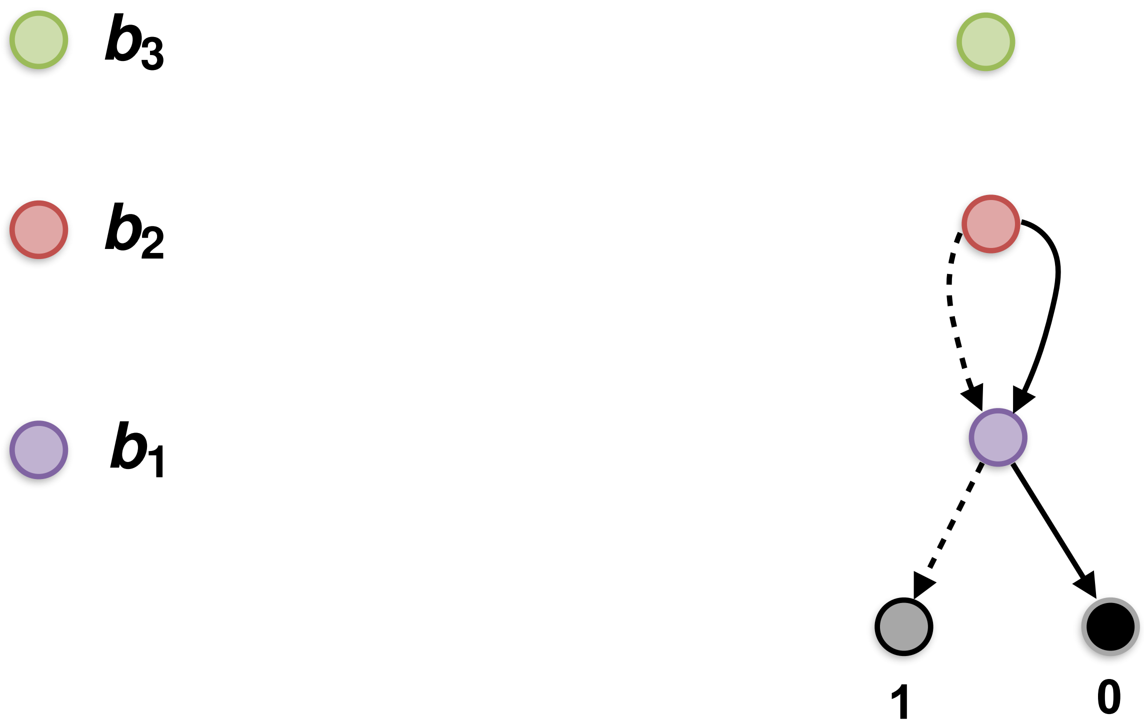


Ordered (Reduced) Binary Decision Diagram

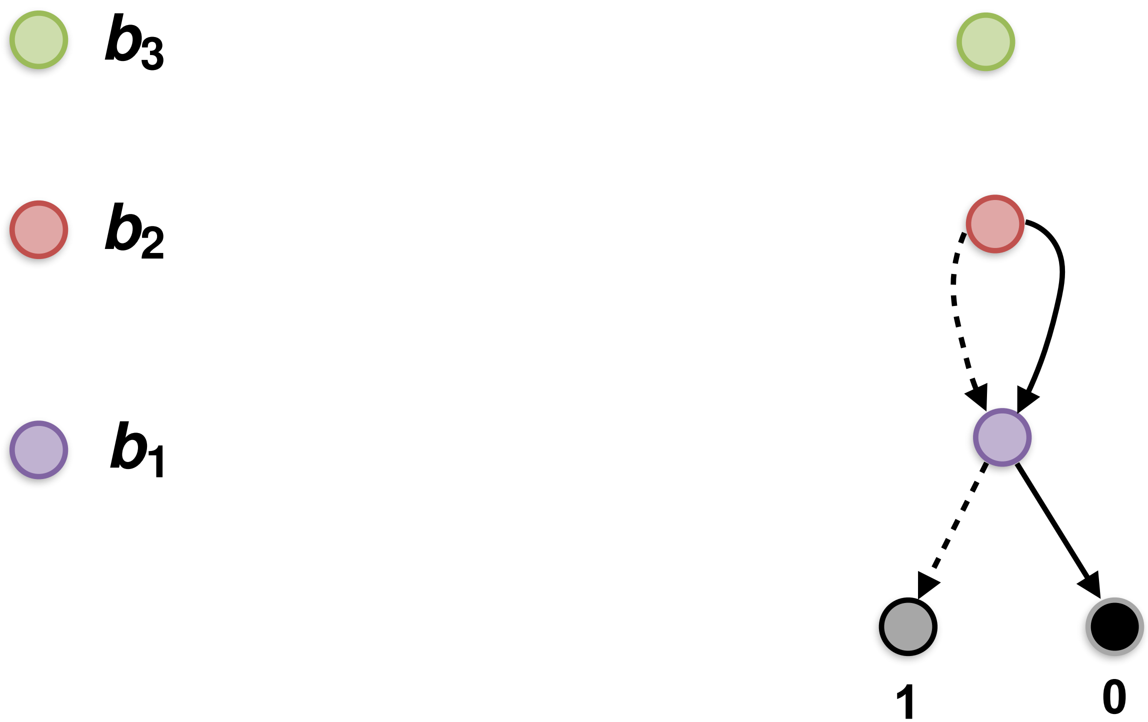


Ordered (Reduced) Binary Decision Diagram

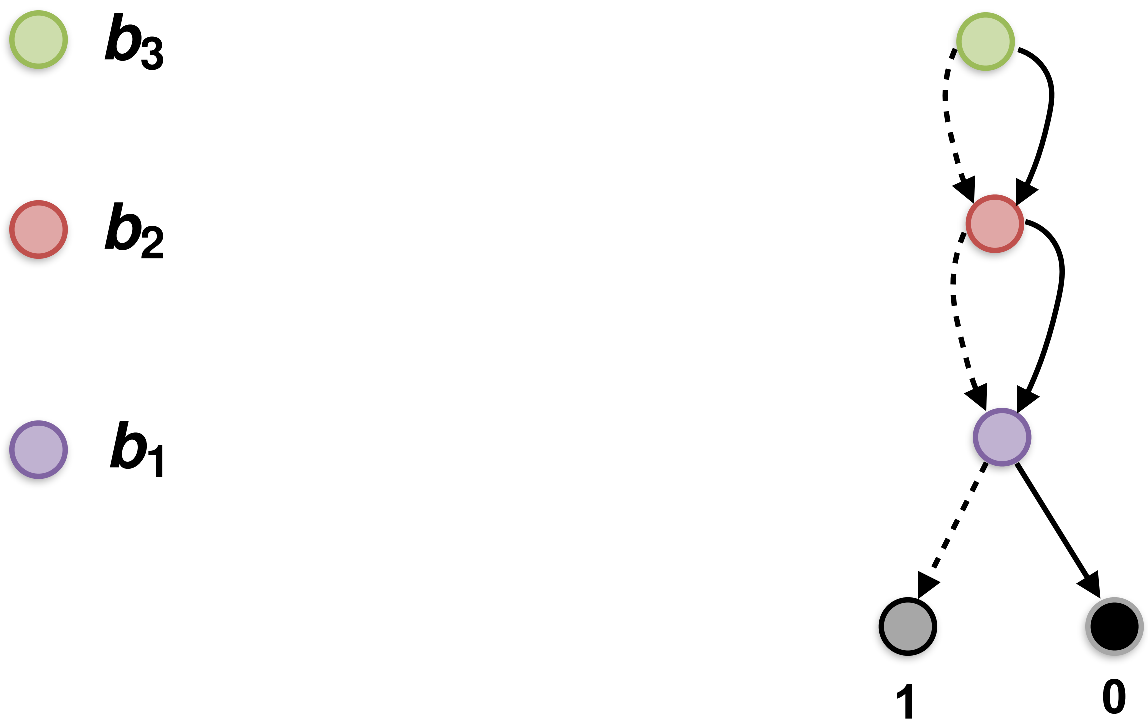
Ordered (Reduced) Binary Decision Diagram



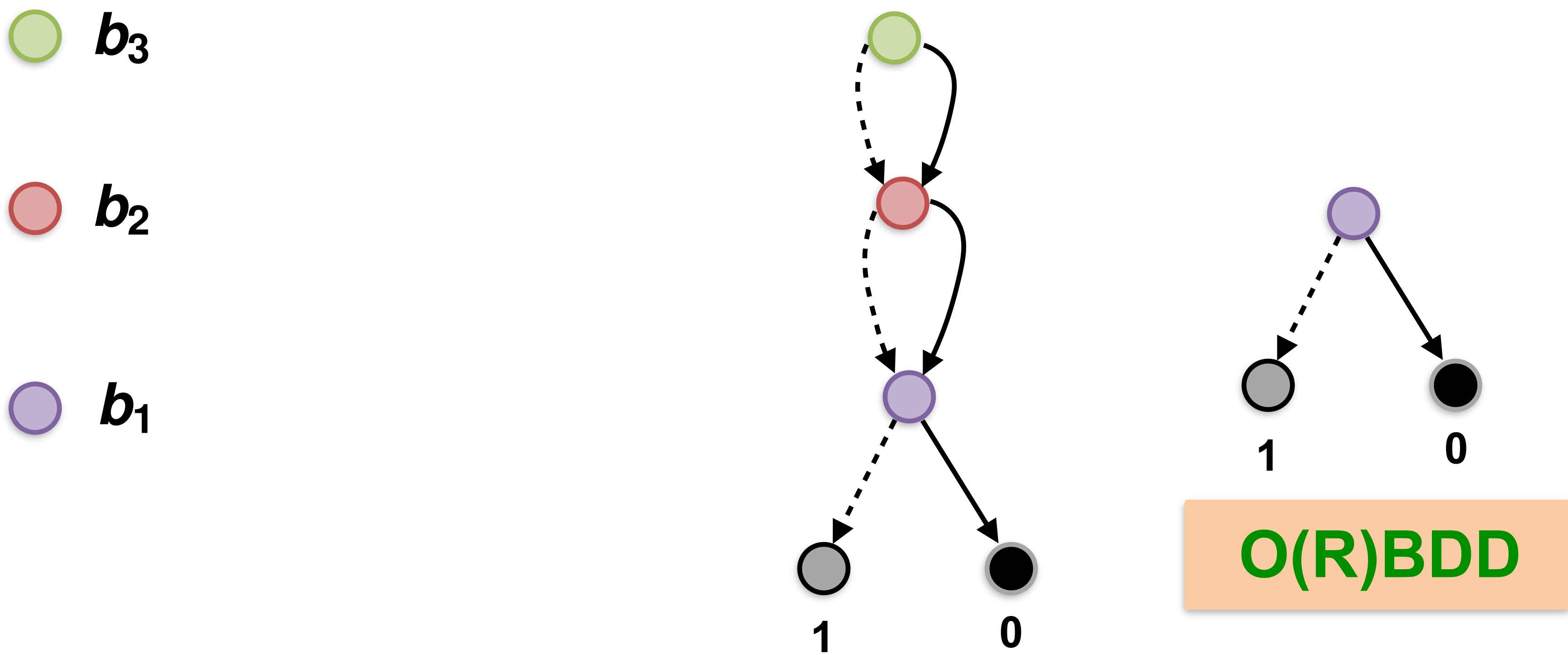
Ordered (Reduced) Binary Decision Diagram



Ordered (Reduced) Binary Decision Diagram



Ordered (Reduced) Binary Decision Diagram

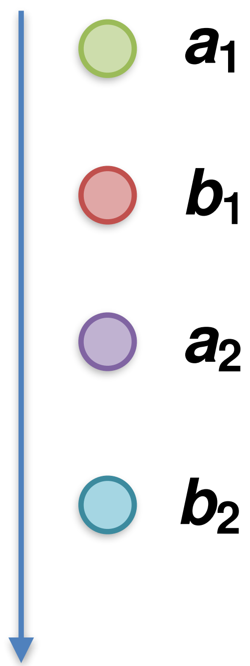


Variables Order (1)

Example: two-bit comparator [Clarke et al.] $a_1 = b_1 \wedge a_2 = b_2$

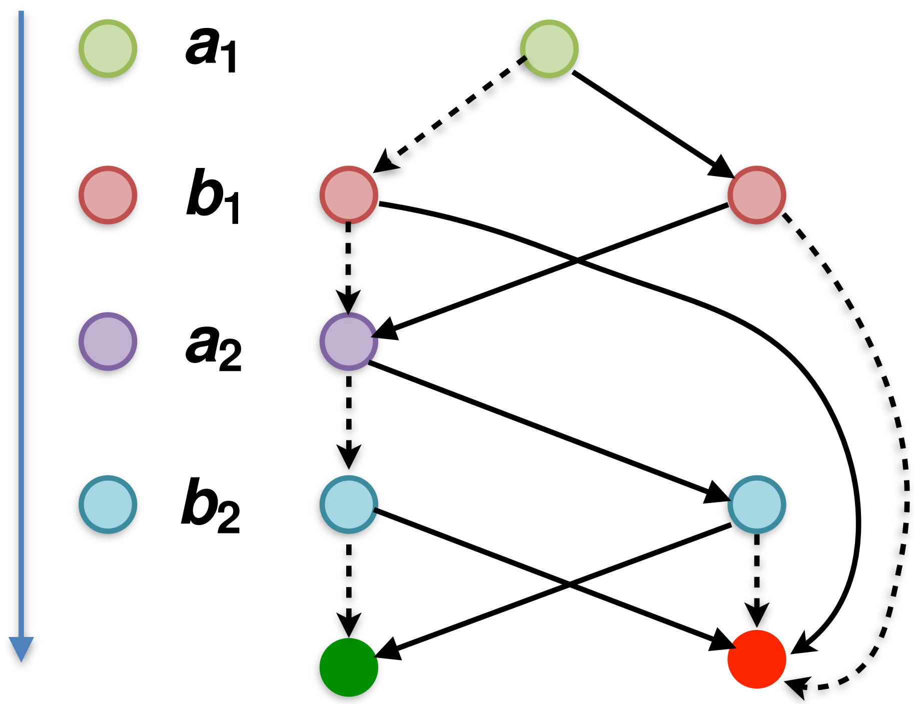
Variables Order (1)

Example: two-bit comparator [Clarke et al.] $a_1 = b_1 \wedge a_2 = b_2$



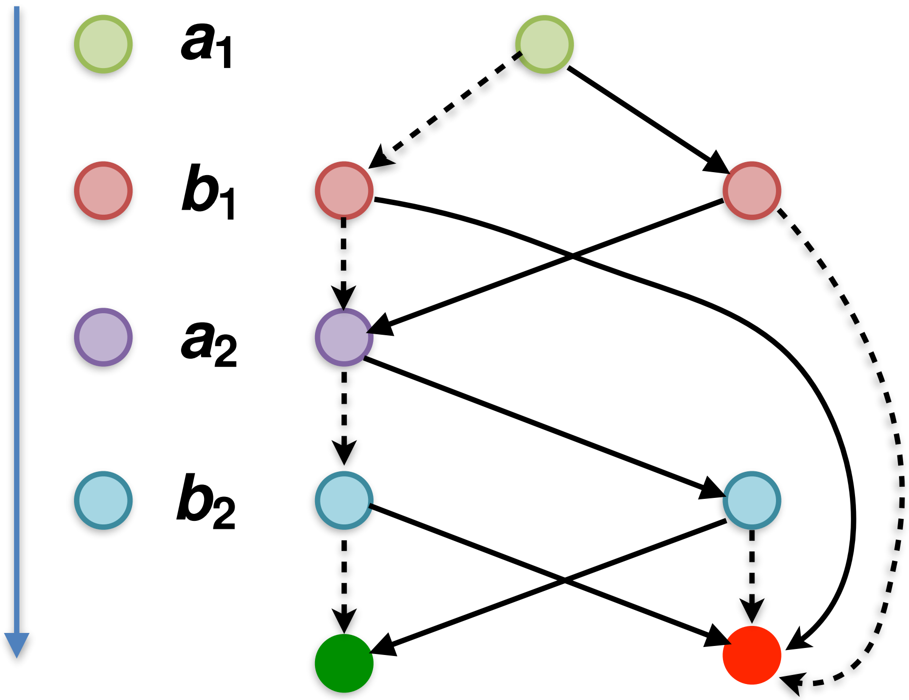
Variables Order (1)

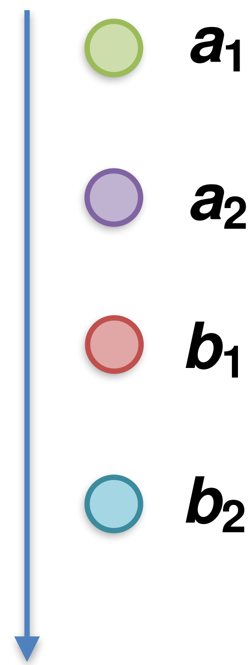
Example: two-bit comparator [Clarke et al.] $a_1 = b_1 \wedge a_2 = b_2$



Variables Order (1)

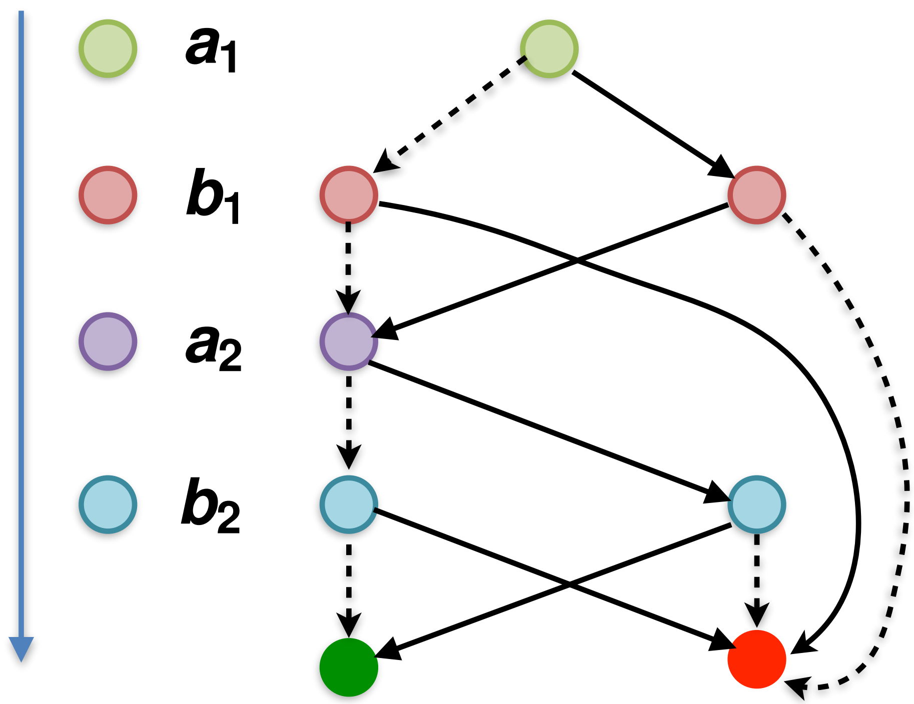
Example: two-bit comparator [Clarke et al.] $a_1 = b_1 \wedge a_2 = b_2$

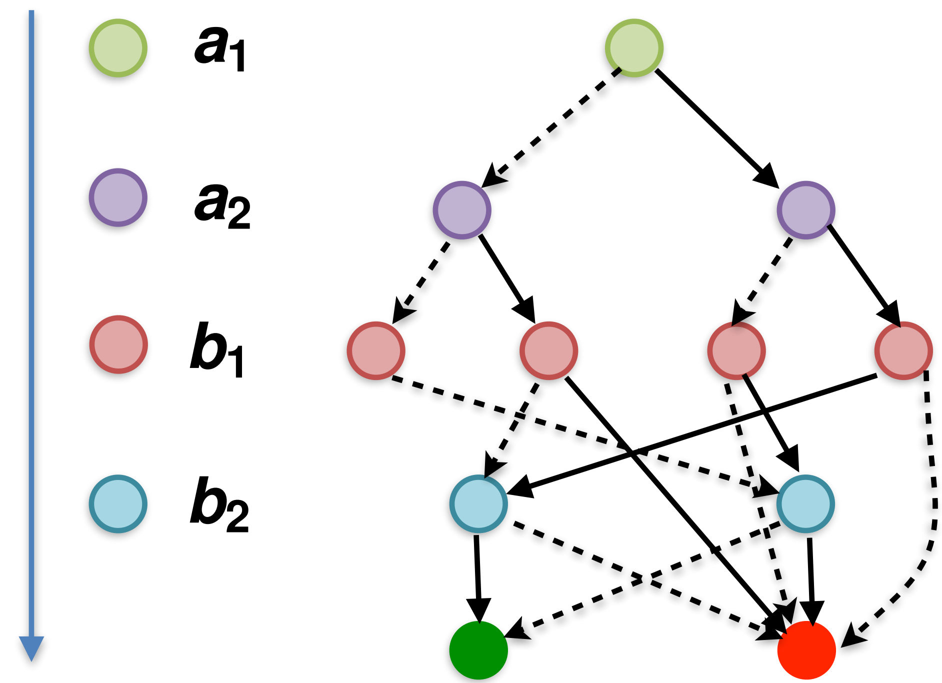




Variables Order (1)

Example: two-bit comparator [Clarke et al.] $a_1 = b_1 \wedge a_2 = b_2$





Variables Order (2)

1. Finding an optimal order is NP-complete
2. There are boolean functions that have exponential size BDDs regardless of order

Solutions:

- smart order (related variables close in BDD)
- dynamic re-ordering

Properties of ORBDDs

Fix an ordering $\boldsymbol{b_1} < \boldsymbol{b_2} < \cdots < \boldsymbol{b_k}$

Boolean function $\boldsymbol{f(\bar{b})}$

Properties of ORBDDs

Fix an ordering $b_1 < b_2 < \cdots < b_k$

Boolean function $f(\bar{b})$

- fixed order for the variables
- maximally reduced

Properties of ORBDDs

Fix an ordering $\boldsymbol{b_1 < b_2 < \cdots < b_k}$

Boolean function $\boldsymbol{f(\bar{b})}$

- fixed order for the variables
- maximally reduced

$\boldsymbol{BDD(f(\bar{b}))}$ is uniquely defined

Properties of ORBDDs

Fix an ordering $b_1 < b_2 < \cdots < b_k$

Boolean function $f(\bar{b})$

- fixed order for the variables
- maximally reduced

$BDD(f(\bar{b}))$ is uniquely defined

or equivalently: there is a canonical form for $BDD(f(\bar{b}))$

Properties of ORBDDs

Fix an ordering $b_1 < b_2 < \cdots < b_k$

Boolean function $f(\bar{b})$

$BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

- fixed order for the variables
- maximally reduced

$BDD(f(\bar{b}))$ is uniquely defined

or equivalently: there is a canonical form for $BDD(f(\bar{b}))$

Operations on BDDs

Encoding Boolean Functions

Complement

Fix an ordering $b_1 < b_2 < \cdots < b_k$

Boolean function $f(\bar{b})$

$BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

Complement

Fix an ordering $b_1 < b_2 < \cdots < b_k$

Boolean function $f(\bar{b})$

$BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

Goal: build BDD for $\neg f(\bar{b})$

Complement

Fix an ordering $b_1 < b_2 < \cdots < b_k$

Boolean function $f(\bar{b})$

$BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

Goal: build BDD for $\neg f(\bar{b})$

Example: two-bit comparator [Clarke et al.]

$a_1 = b_1 \wedge a_2 = b_2$

Complement

Fix an ordering $b_1 < b_2 < \cdots < b_k$

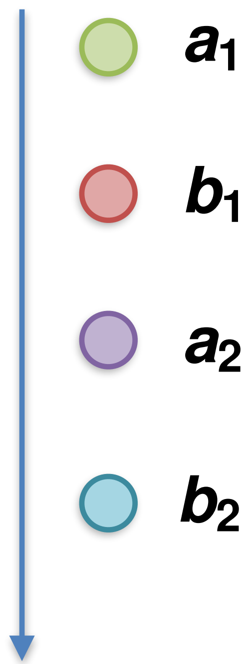
Boolean function $f(\bar{b})$

$BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

Goal: build BDD for $\neg f(\bar{b})$

Example: two-bit comparator [Clarke et al.]

$a_1 = b_1 \wedge a_2 = b_2$



Complement

Fix an ordering $b_1 < b_2 < \cdots < b_k$

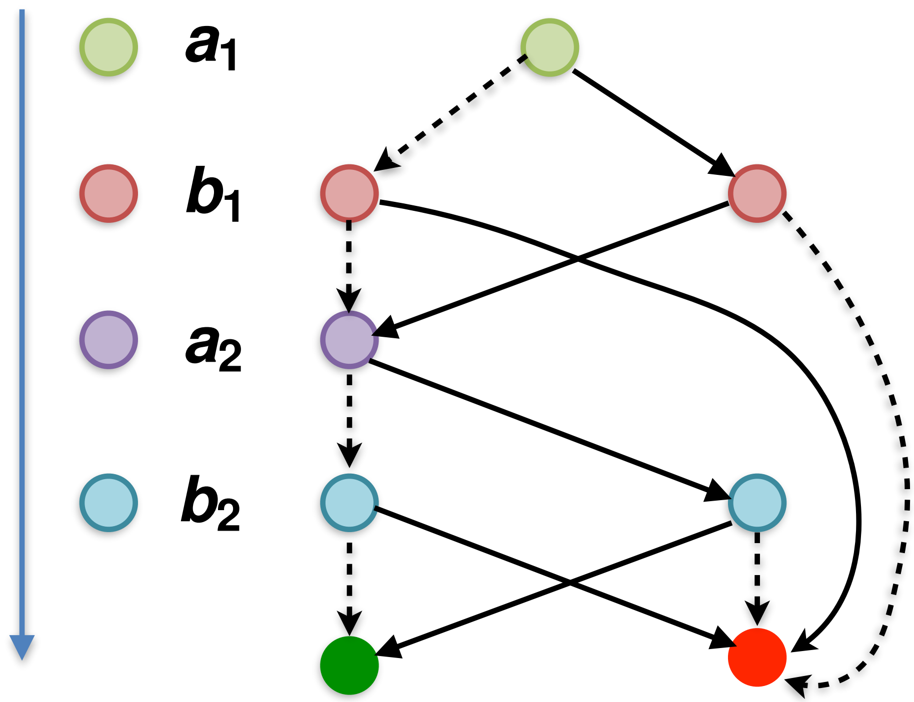
Boolean function $f(\bar{b})$

$BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

Goal: build BDD for $\neg f(\bar{b})$

Example: two-bit comparator [Clarke et al.]

$a_1 = b_1 \wedge a_2 = b_2$



Complement

Fix an ordering $b_1 < b_2 < \cdots < b_k$

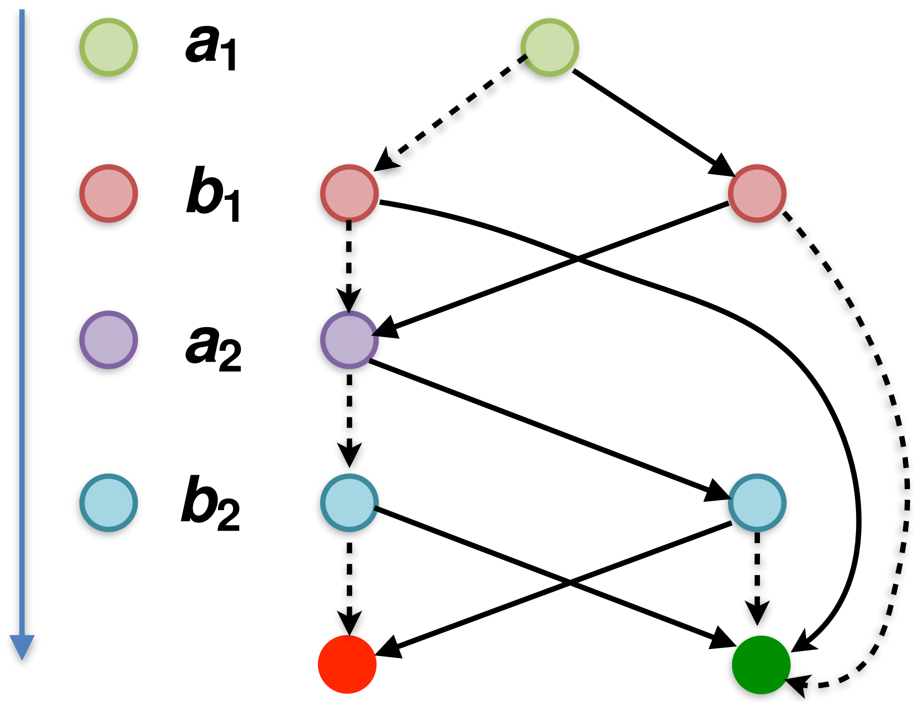
Boolean function $f(\bar{b})$

$BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

Goal: build BDD for $\neg f(\bar{b})$

Example: two-bit comparator [Clarke et al.]

$a_1 = b_1 \wedge a_2 = b_2$



Complement

Fix an ordering $b_1 < b_2 < \cdots < b_k$

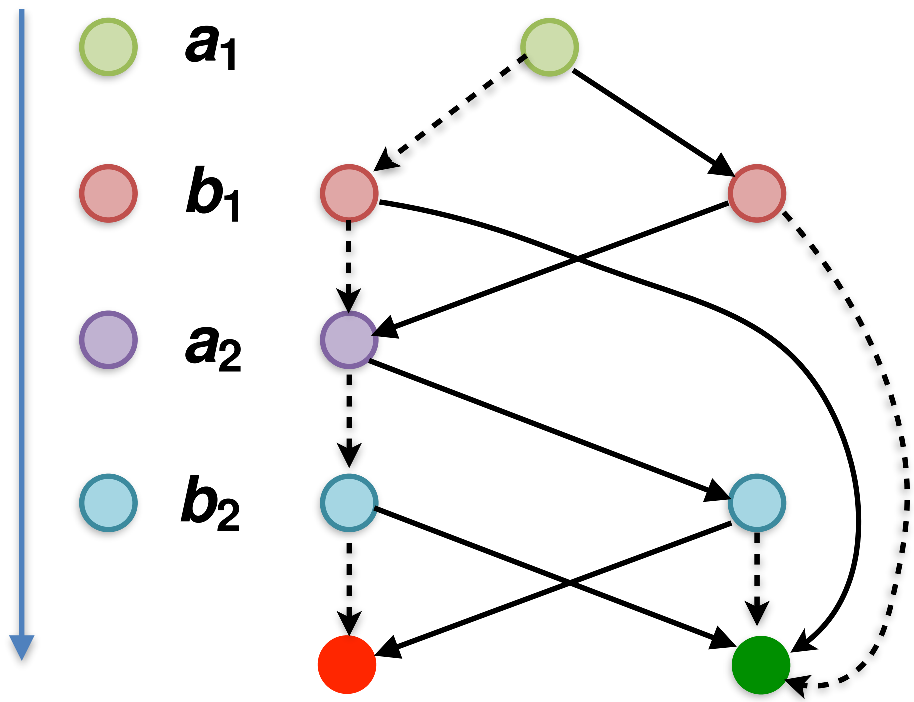
Boolean function $f(\bar{b})$

$BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

Goal: build BDD for $\neg f(\bar{b})$

Example: two-bit comparator [Clarke et al.]

$a_1 = b_1 \wedge a_2 = b_2$



$$BDD(\neg f(\bar{b})) = \overline{BDD(f(\bar{b}))}$$

Restrict

Boolean function $\boldsymbol{f}(\bar{\boldsymbol{b}})$ $\quad$ $\boldsymbol{BDD(f(\bar{b}))}$ the ORBDD for $\boldsymbol{f(\bar{b})}$

Restrict

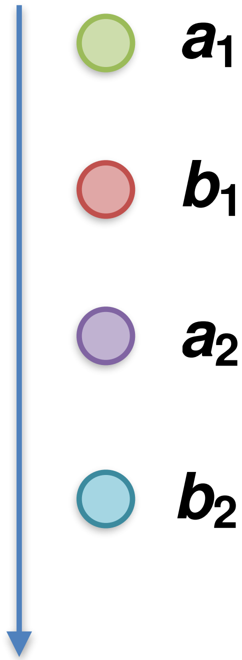
Boolean function $\boldsymbol{f(\bar{b})}$ | $\boldsymbol{BDD(f(\bar{b}))}$ the ORBDD for $\boldsymbol{f(\bar{b})}$

Goal: Build BDD for $\boldsymbol{f(\bar{b})[b_j/0]}$ or $\boldsymbol{f(\bar{b})[b_j/1]}$

Restrict

Boolean function $\boldsymbol{f(\bar{b})}$ $\qquad$ $\boldsymbol{BDD(f(\bar{b}))}$ the ORBDD for $\boldsymbol{f(\bar{b})}$

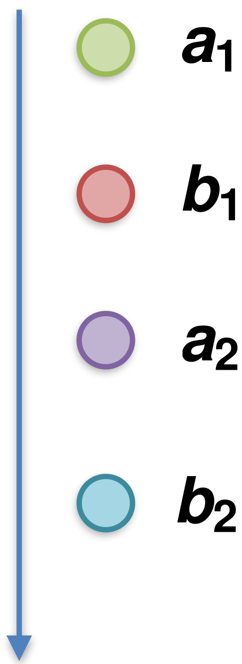
Goal: Build BDD for $\boldsymbol{f(\bar{b})[b_j/0]}$ or $\boldsymbol{f(\bar{b})[b_j/1]}$



Restrict

Boolean function $f(\bar{b})$ $\qquad$ $BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

Goal: Build BDD for $f(\bar{b})[b_j/0]$ or $f(\bar{b})[b_j/1]$

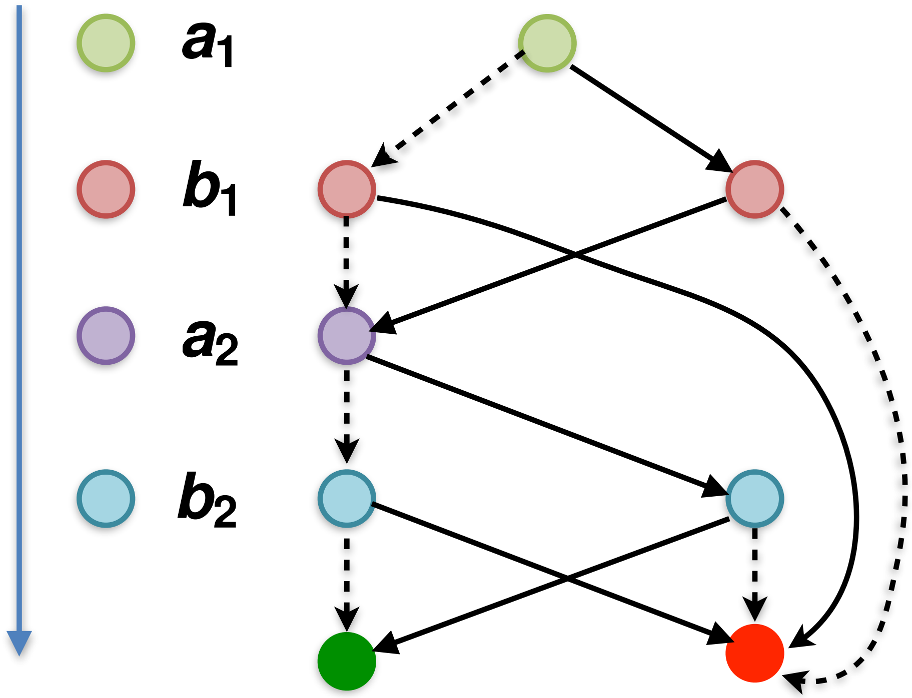


$a_1 = b_1 \wedge a_2 = b_2$

Restrict

Boolean function $f(\bar{b})$ $\quad$ $BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

Goal: Build BDD for $f(\bar{b})[b_j/0]$ or $f(\bar{b})[b_j/1]$

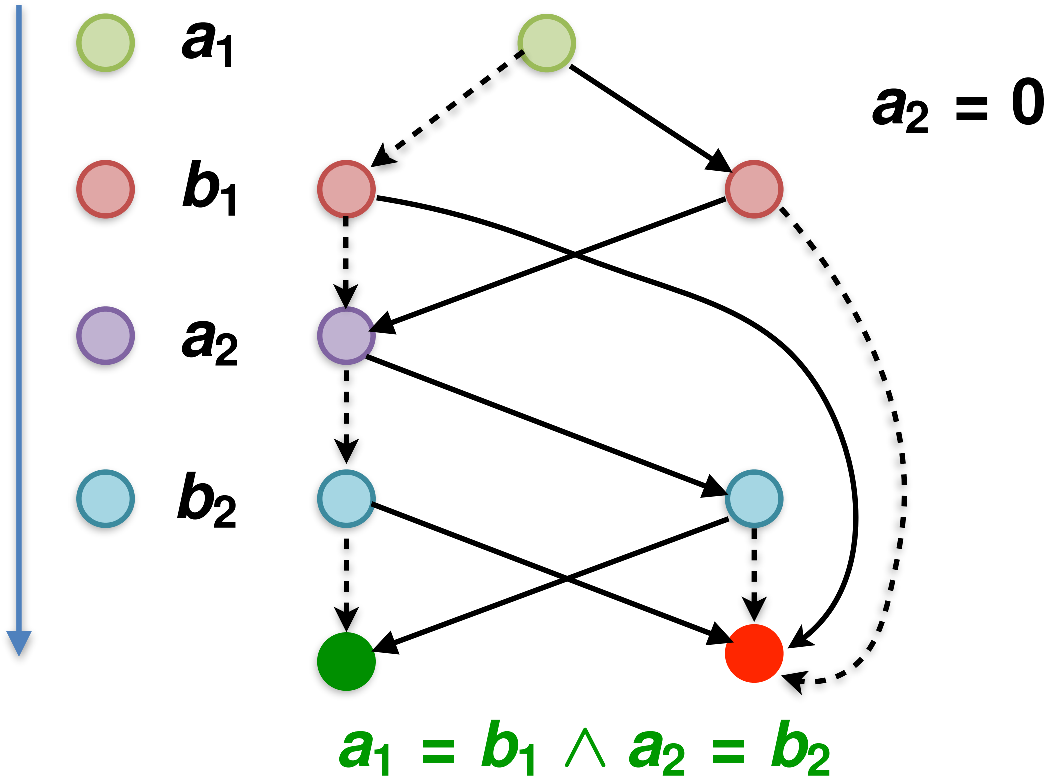


$a_1 = b_1 \wedge a_2 = b_2$

Restrict

Boolean function $\boldsymbol{f(\bar{b})}$ $\qquad$ $\boldsymbol{BDD(f(\bar{b}))}$ the ORBDD for $\boldsymbol{f(\bar{b})}$

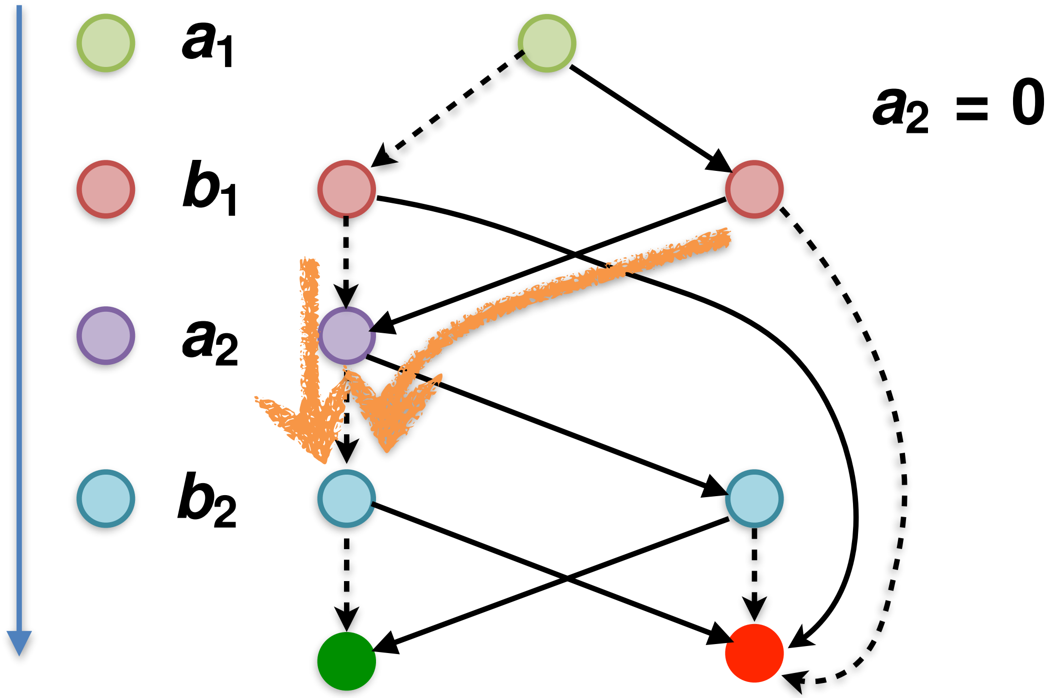
Goal: Build BDD for $\boldsymbol{f(\bar{b})[b_j/0]}$ or $\boldsymbol{f(\bar{b})[b_j/1]}$



Restrict

Boolean function $\boldsymbol{f(\bar{b})}$ $\qquad$ $\boldsymbol{BDD(f(\bar{b}))}$ the ORBDD for $\boldsymbol{f(\bar{b})}$

Goal: Build BDD for $f(\bar{b})[b_j/0]$ or $f(\bar{b})[b_j/1]$

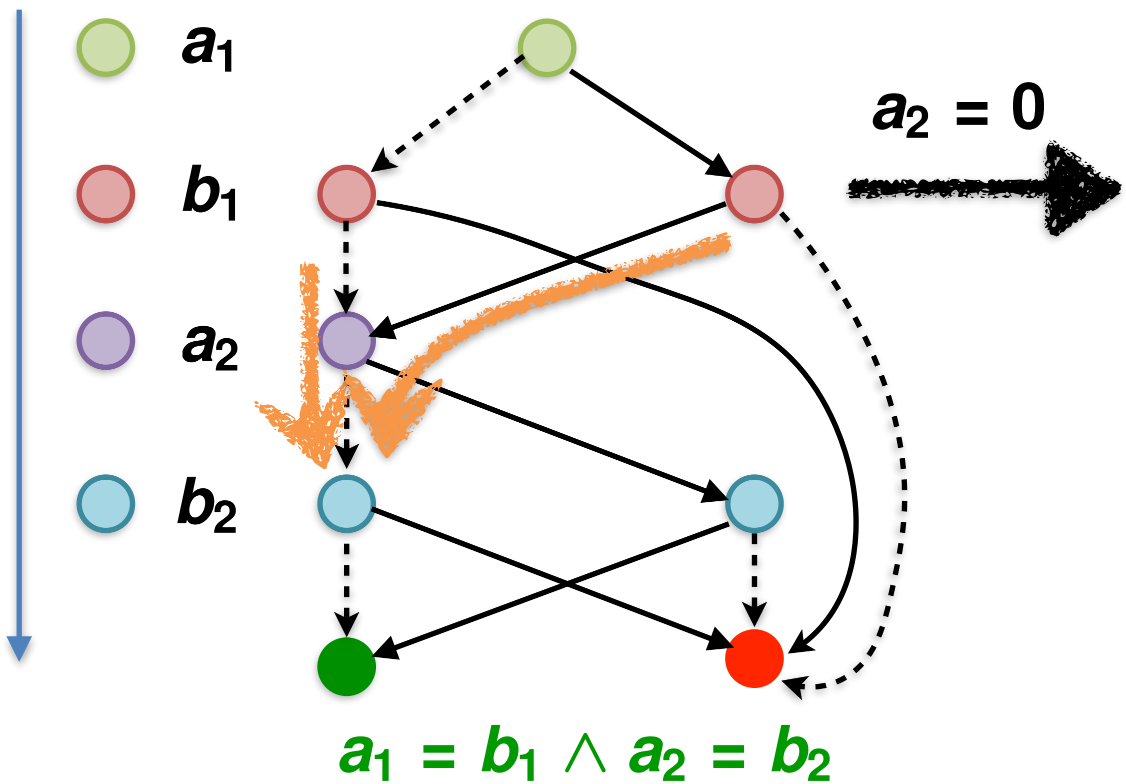


$a_1 = b_1 \land a_2 = b_2$

Restrict

Boolean function $\boldsymbol{f(\bar{b})}$ | $\boldsymbol{BDD(f(\bar{b}))}$ the ORBDD for $\boldsymbol{f(\bar{b})}$

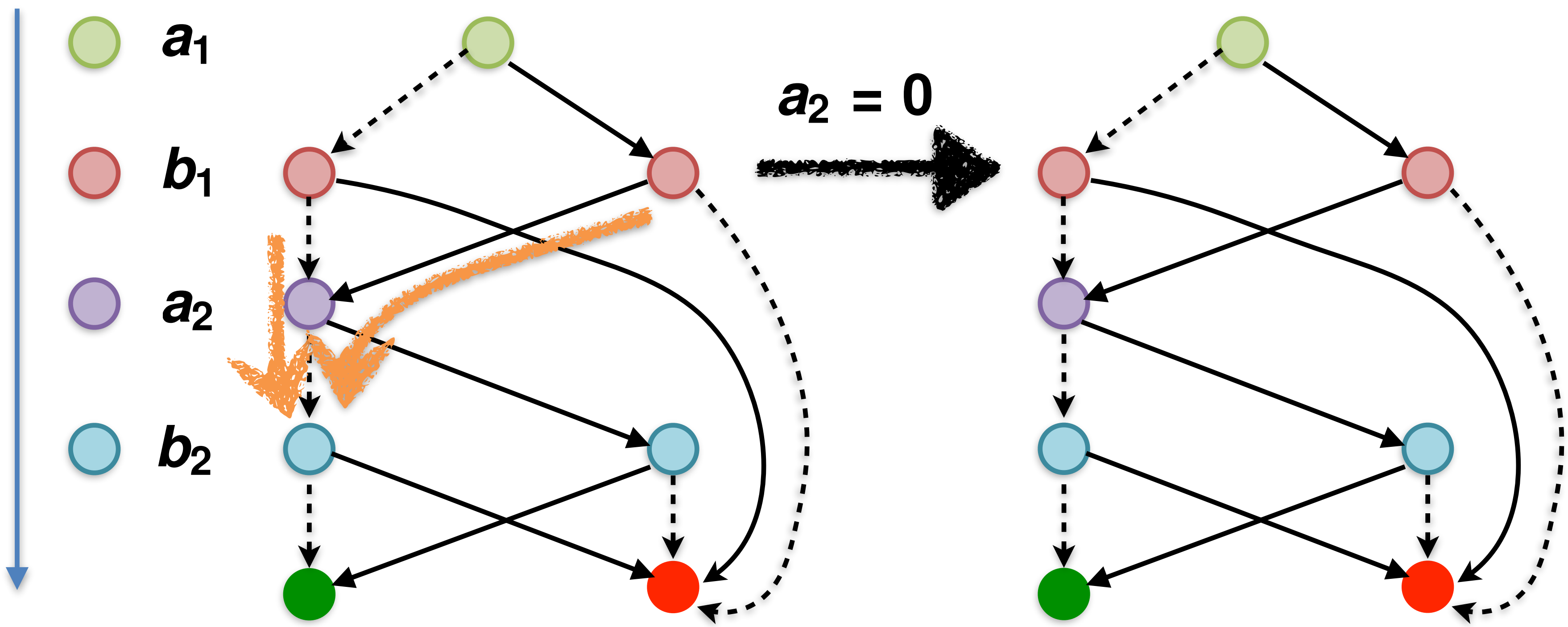
Goal: Build BDD for $\boldsymbol{f(\bar{b})[b_j/0]}$ or $\boldsymbol{f(\bar{b})[b_j/1]}$



Restrict

Boolean function $f(\bar{b})$ $\quad$ $BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

Goal: Build BDD for $f(\bar{b})[b_j/0]$ or $f(\bar{b})[b_j/1]$

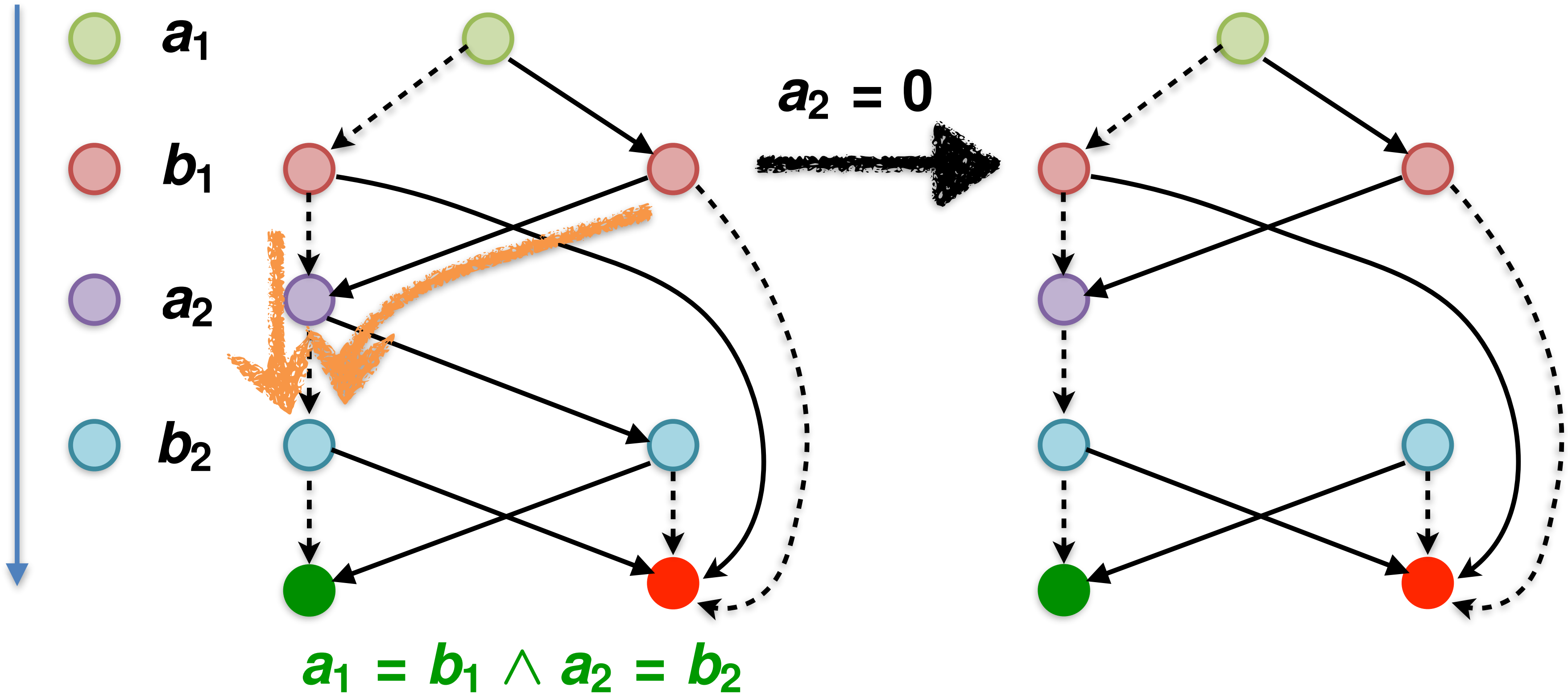


$a_1 = b_1 \wedge a_2 = b_2$

Restrict

Boolean function $\boldsymbol{f(\bar{b})}$ $\quad$ $\boldsymbol{BDD(f(\bar{b}))}$ the ORBDD for $\boldsymbol{f(\bar{b})}$

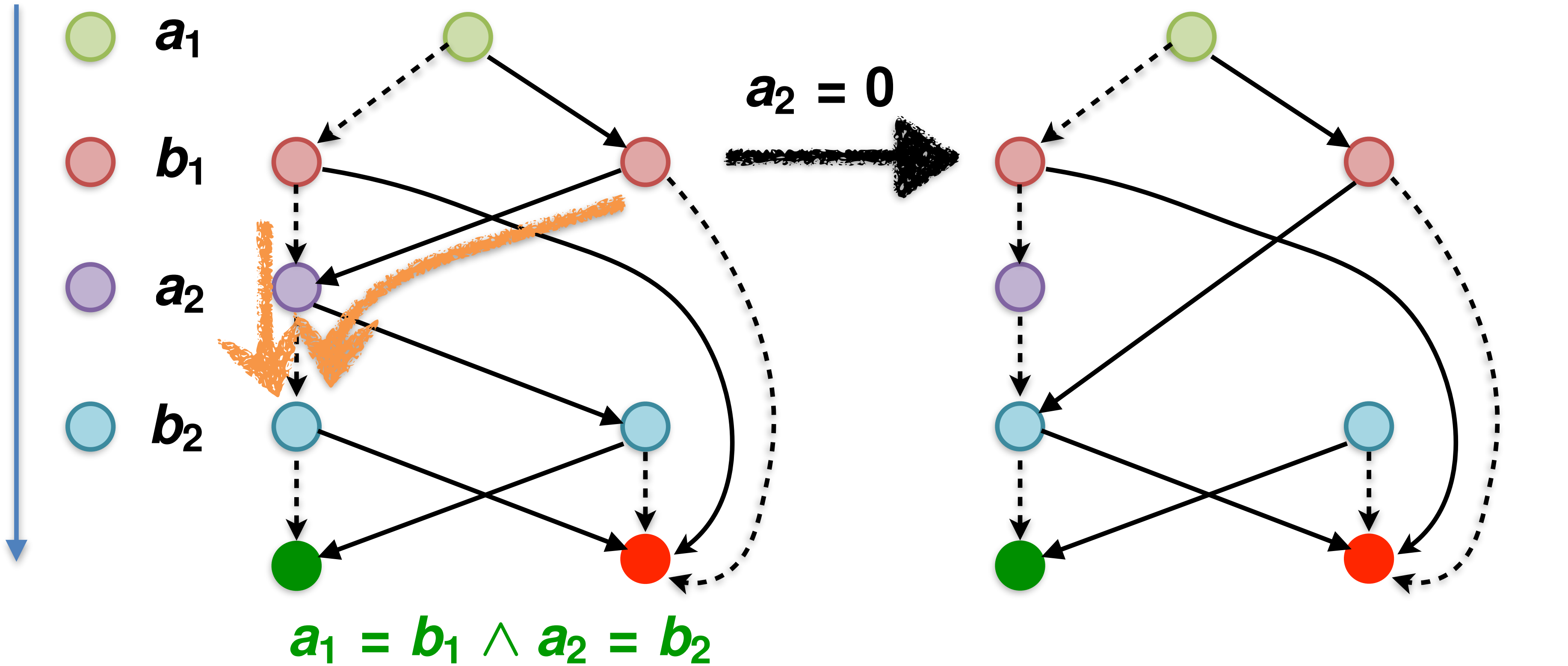
Goal: Build BDD for $\boldsymbol{f(\bar{b})[b_j/0]}$ or $\boldsymbol{f(\bar{b})[b_j/1]}$



Restrict

Boolean function $f(\bar{b})$ — $BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

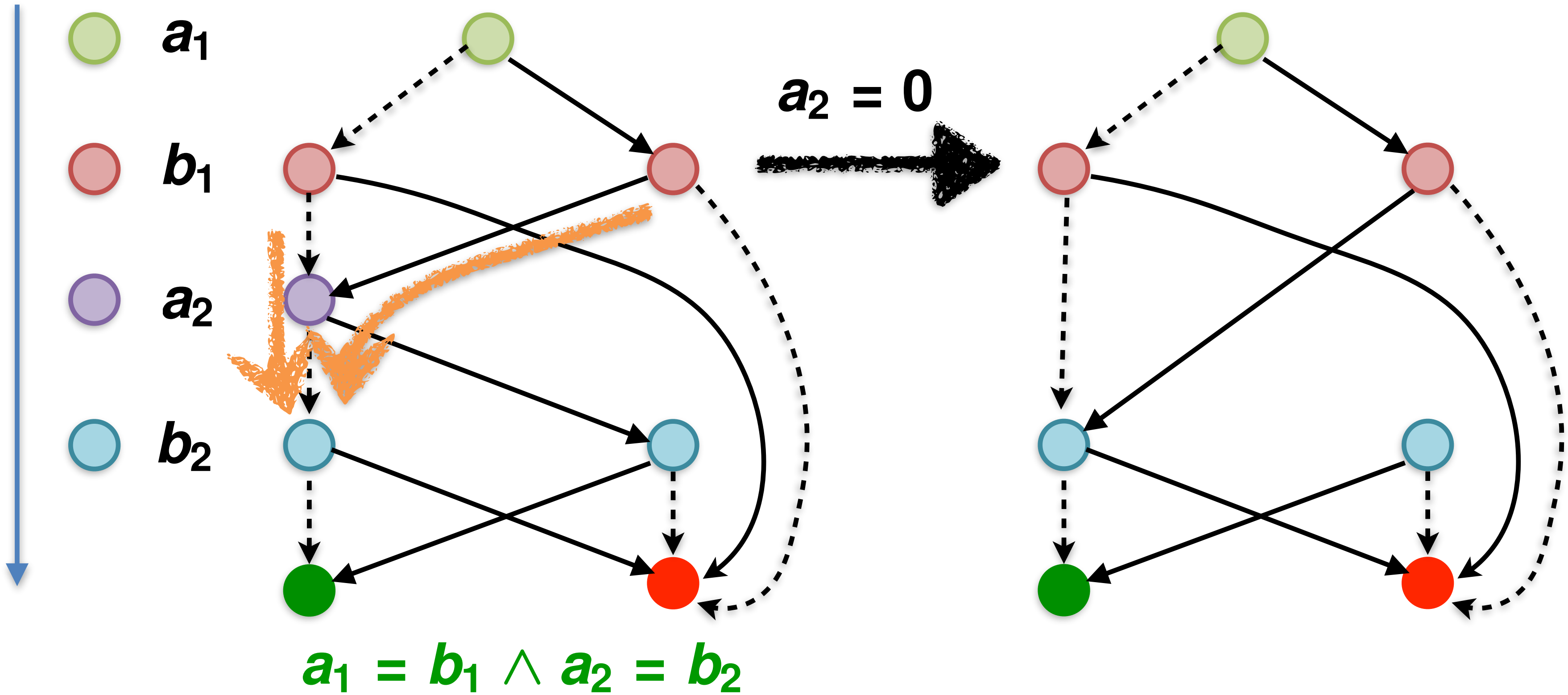
Goal: Build BDD for $f(\bar{b})[b_j/0]$ or $f(\bar{b})[b_j/1]$



Restrict

Boolean function $f(\bar{b})$ $\quad$ $BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

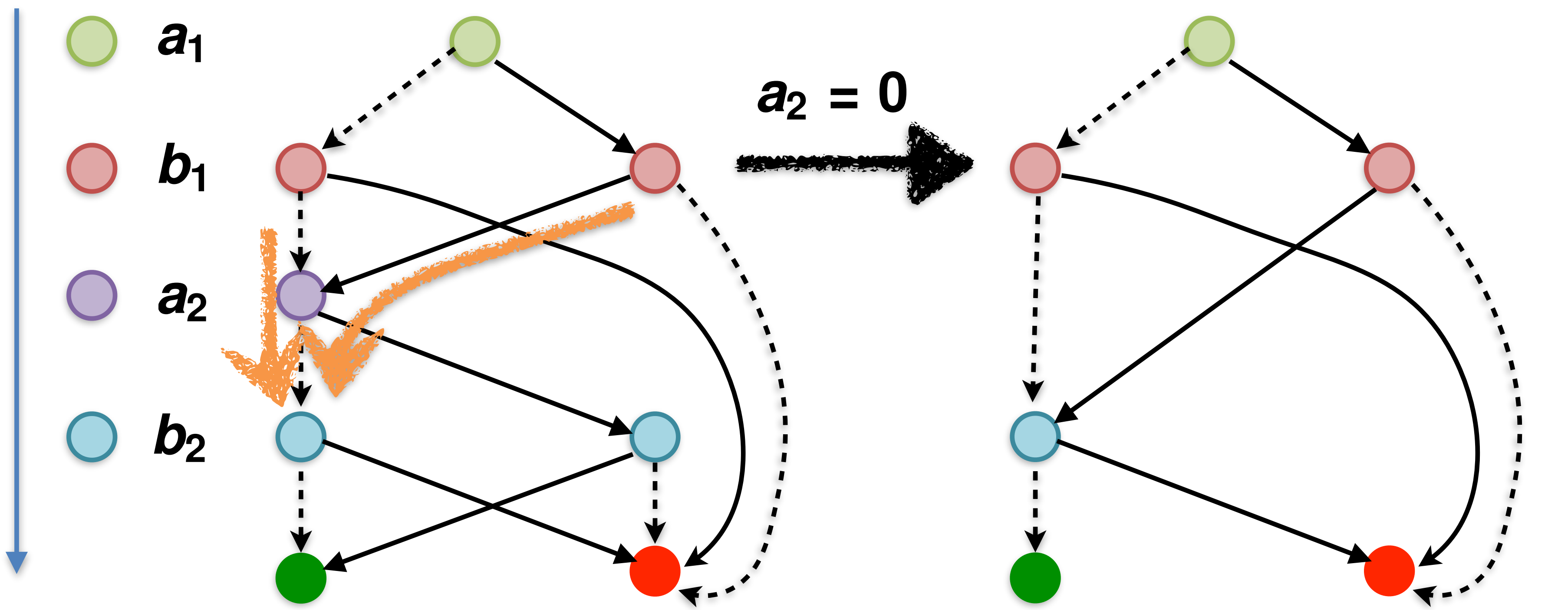
Goal: Build BDD for $f(\bar{b})[b_j/0]$ or $f(\bar{b})[b_j/1]$



Restrict

Boolean function $f(\bar{b})$ $\qquad$ $BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

Goal: Build BDD for $f(\bar{b})[b_j/0]$ or $f(\bar{b})[b_j/1]$

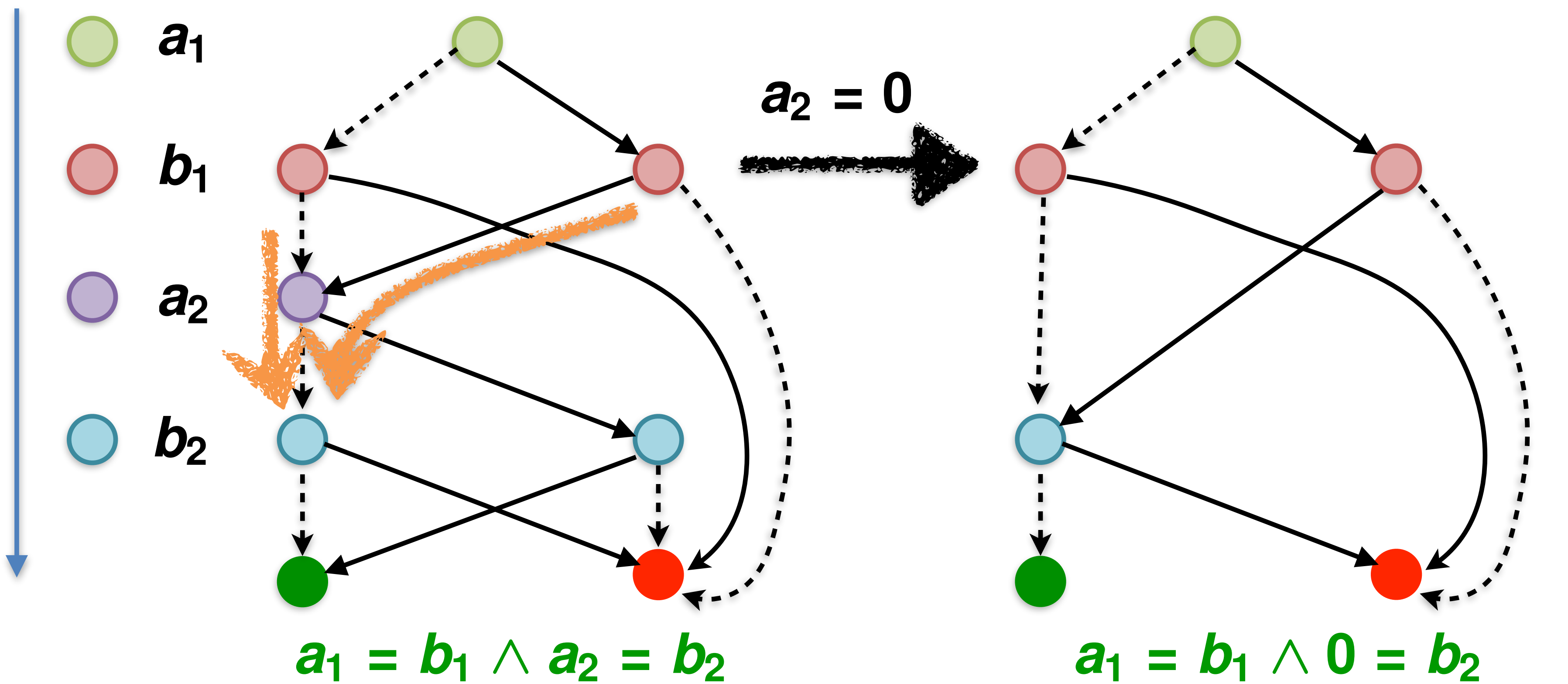


$a_1 = b_1 \wedge a_2 = b_2$

Restrict

Boolean function $f(\bar{b})$ $\quad$ **$BDD(f(\bar{b}))$** the ORBDD for $f(\bar{b})$

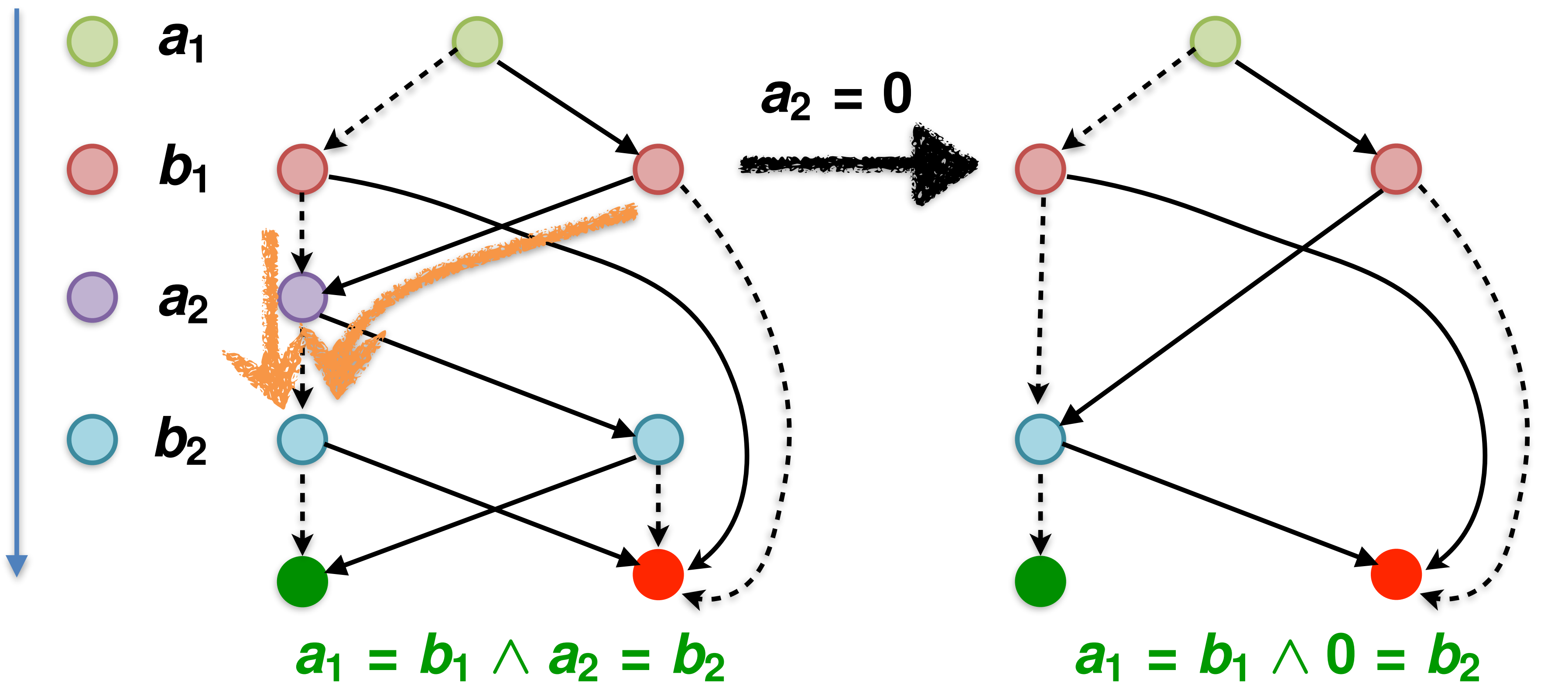
Goal: Build BDD for $f(\bar{b})[b_j/0]$ or $f(\bar{b})[b_j/1]$



Restrict

Boolean function $f(\bar{b})$ $BDD(f(\bar{b}))$ the ORBDD for $f(\bar{b})$

Goal: Build BDD for $f(\bar{b})[b_j/0]$ or $f(\bar{b})[b_j/1]$

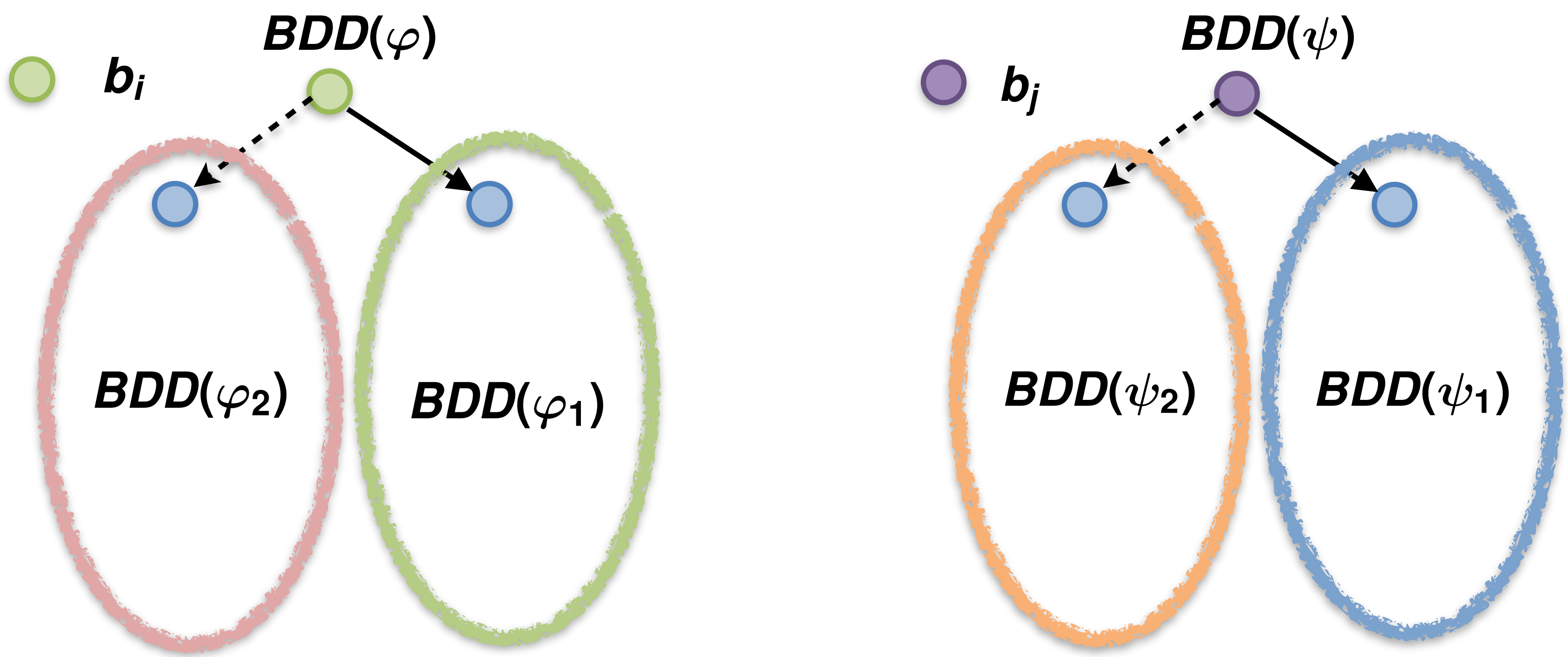


$$BDD(f(\bar{b})[b_j/v]) = \text{Restrict}(BDD(f(\bar{b})), b_j, v)$$

And – Intersection

Fix an ordering $b_1 < b_2 < \cdots < b_k$

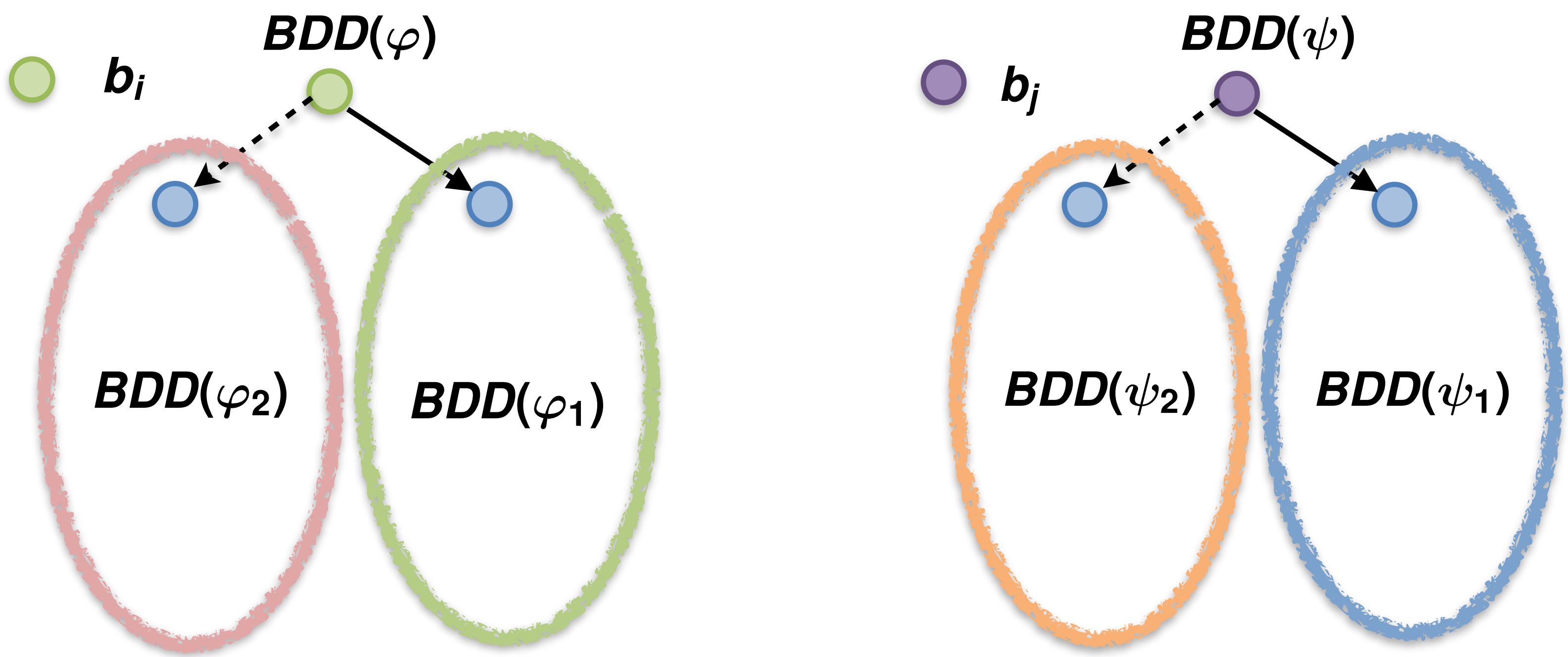
Goal: build BDD for $\varphi \wedge \psi$



And – Intersection

Fix an ordering $b_1 < b_2 < \cdots < b_k$

Goal: build BDD for $\varphi \wedge \psi$

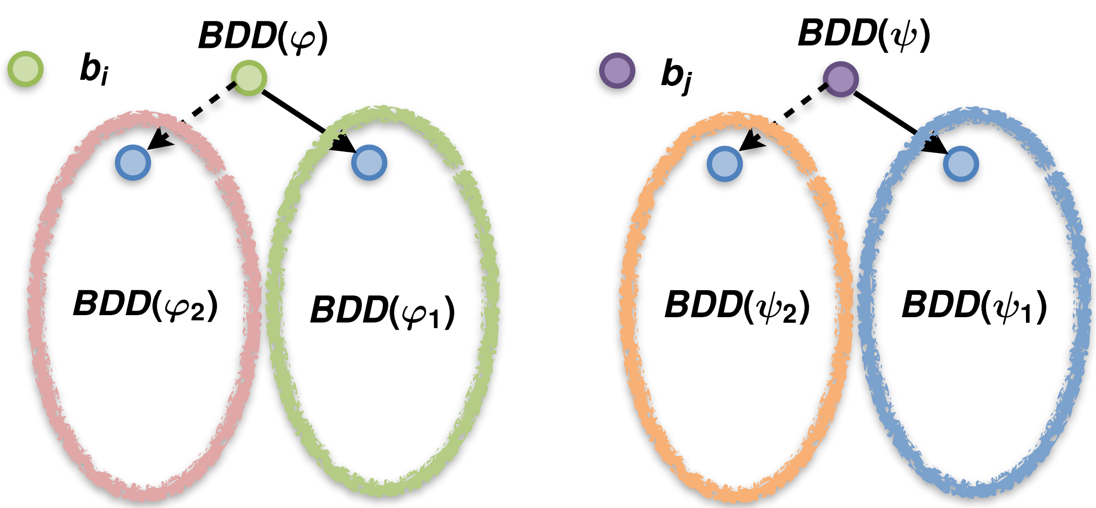


φ = if b_i then φ_1 else φ_2

ψ = if b_j then ψ_1 else ψ_2

And – Intersection

Fix an ordering $b_1 < b_2 < \cdots < b_k$



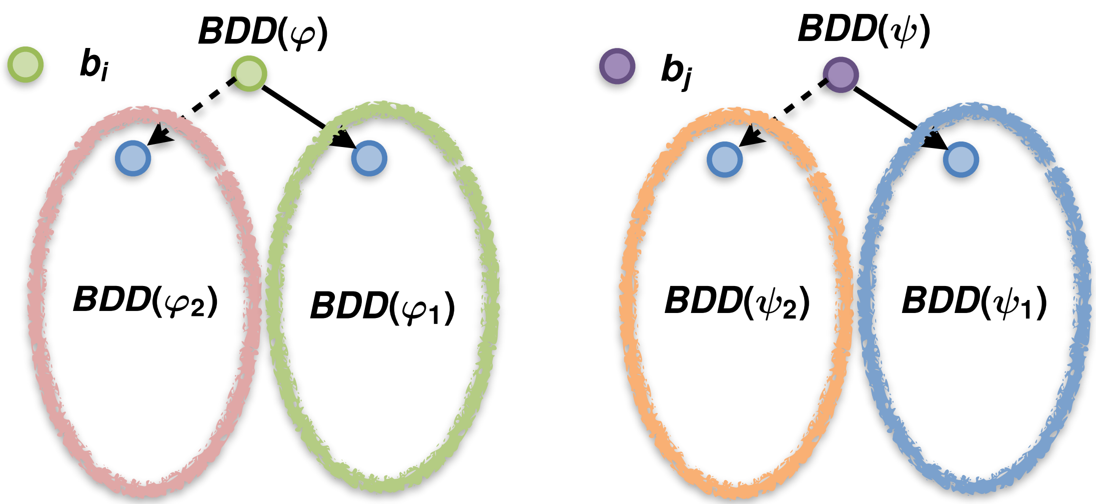
φ = if b_i then φ_1 else φ_2

ψ = if b_j then ψ_1 else ψ_2

And – Intersection

Fix an ordering $b_1 < b_2 < \cdots < b_k$

$i > j$



φ = if b_i then φ_1 else φ_2

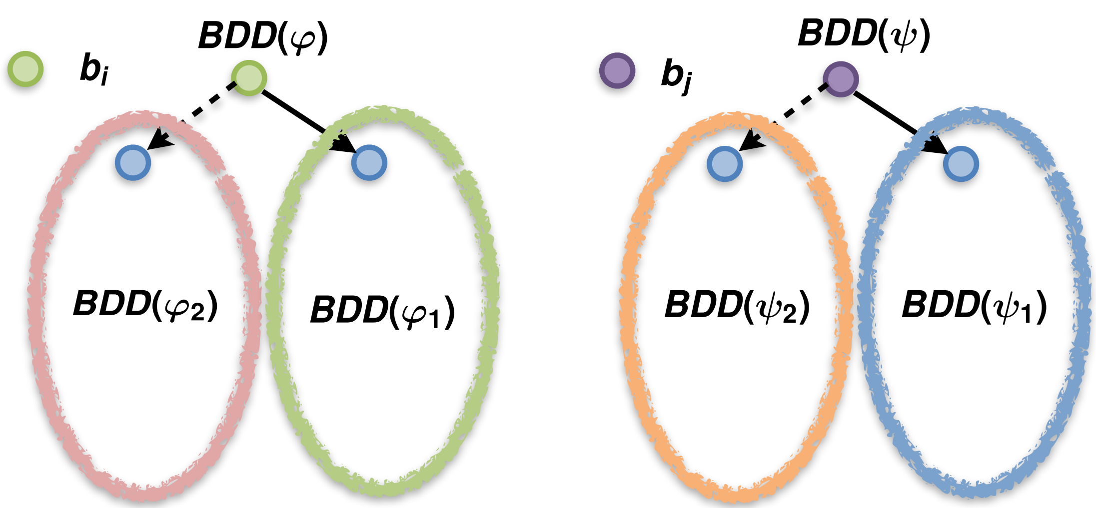
ψ = if b_j then ψ_1 else ψ_2

And – Intersection

Fix an ordering $b_1 < b_2 < \cdots < b_k$

$i > j$

$$\varphi \wedge \psi = \text{if } b_i \text{ then } \varphi_1 \wedge \psi \text{ else } \varphi_2 \wedge \psi$$



φ = if b_i then φ_1 else φ_2

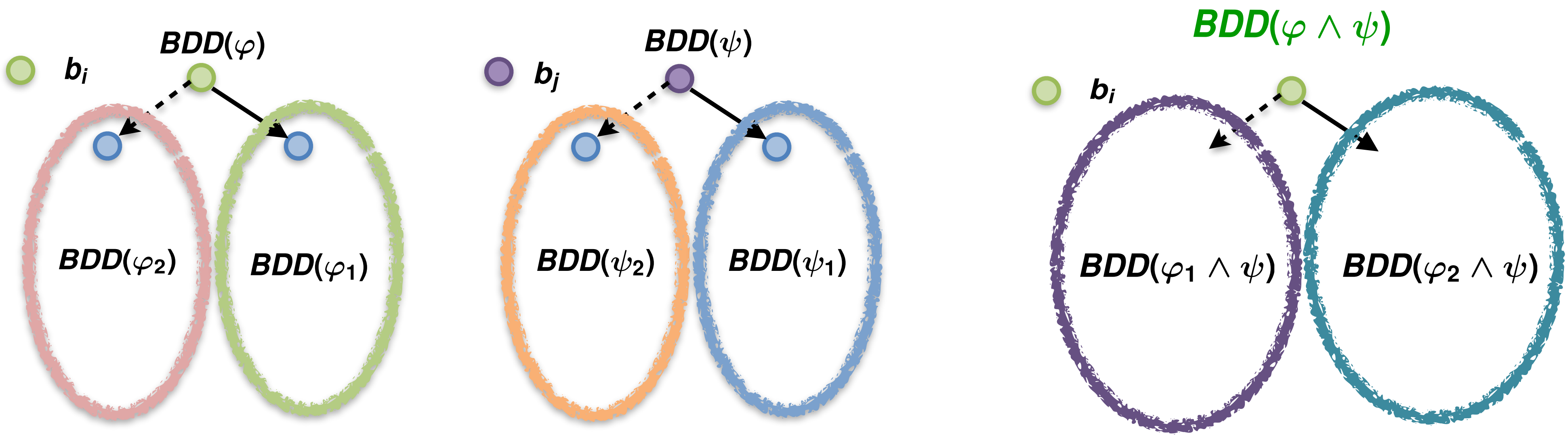
ψ = if b_j then ψ_1 else ψ_2

And – Intersection

Fix an ordering $b_1 < b_2 < \cdots < b_k$

$i > j$

$$\varphi \wedge \psi = \text{if } b_i \text{ then } \varphi_1 \wedge \psi \text{ else } \varphi_2 \wedge \psi$$

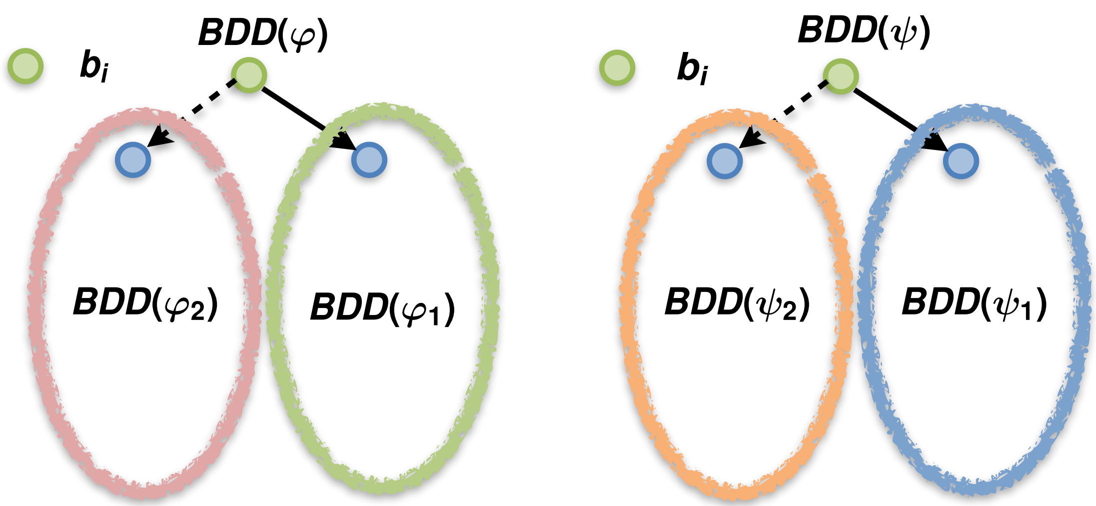


φ = if b_i then φ_1 else φ_2

ψ = if b_j then ψ_1 else ψ_2

And – Intersection

Fix an ordering $b_1 < b_2 < \cdots < b_k$

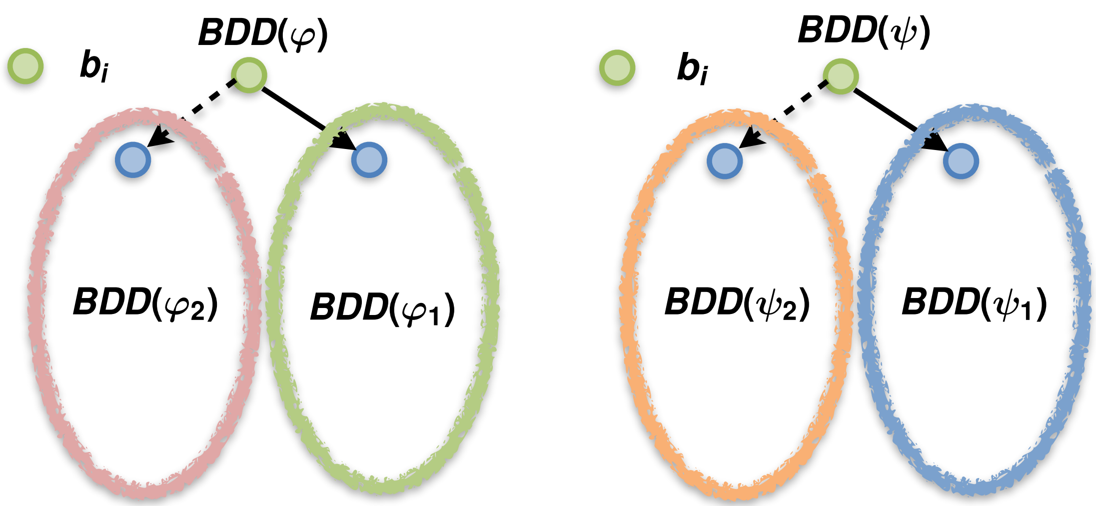


φ = if b_i then φ_1 else φ_2 $\qquad$ ψ = if b_j then ψ_1 else ψ_2

And – Intersection

Fix an ordering $b_1 < b_2 < \cdots < b_k$

$i = j$



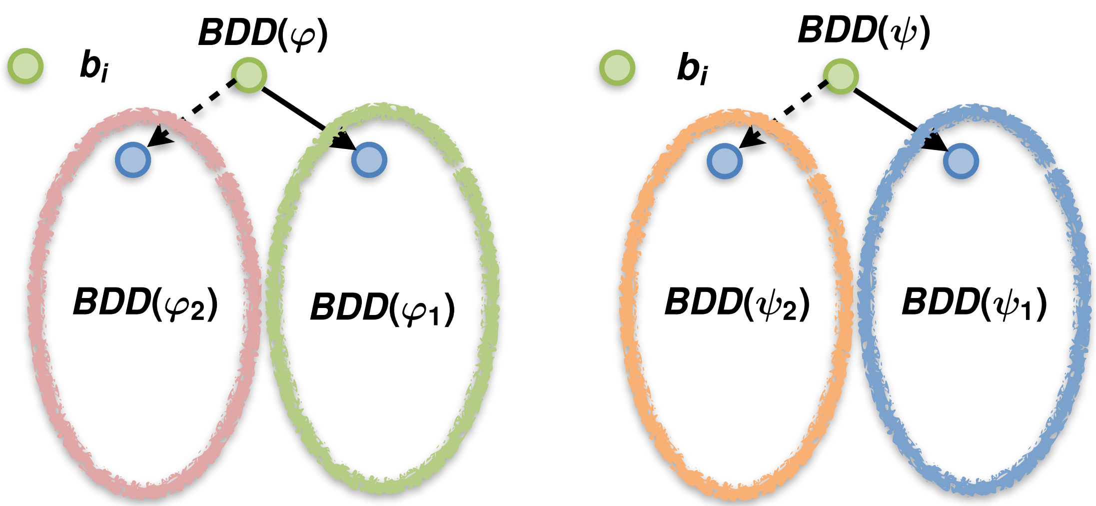
φ = if b_i then φ_1 else φ_2 $\qquad$ ψ = if b_j then ψ_1 else ψ_2

And – Intersection

Fix an ordering $\boldsymbol{b}_1 < \boldsymbol{b}_2 < \cdots < \boldsymbol{b}_k$

$i = j$

$$\varphi \wedge \psi = \text{ if } \boldsymbol{b}_i \text{ then } \varphi_1 \wedge \psi_1 \text{ else } \varphi_2 \wedge \psi_2$$



φ = if $\boldsymbol{b}_i$ then φ_1 else φ_2

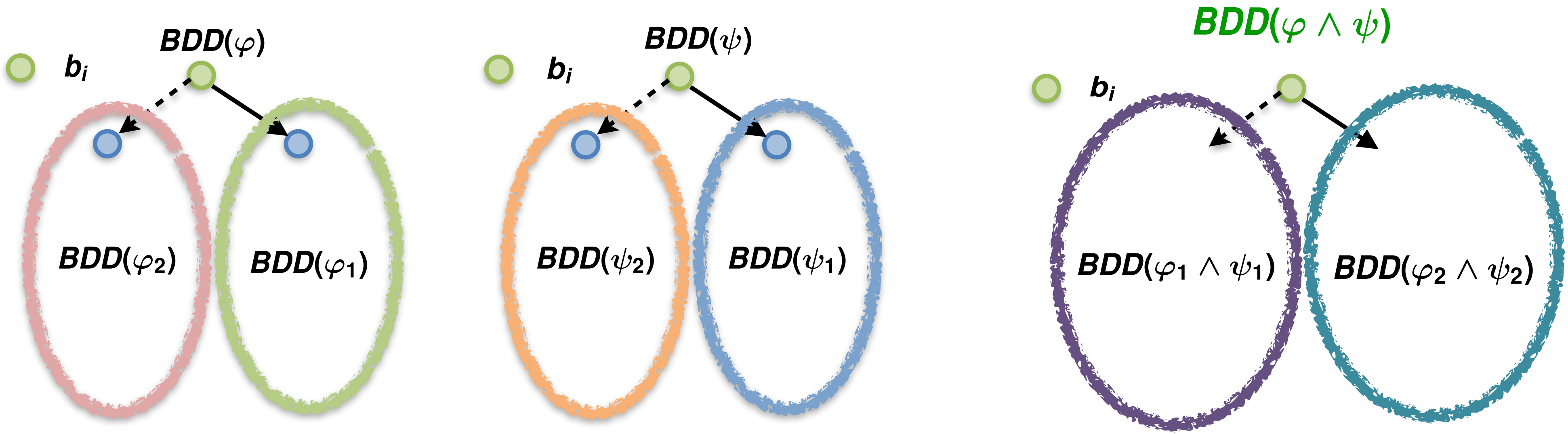
ψ = if $\boldsymbol{b}_j$ then ψ_1 else ψ_2

And – Intersection

Fix an ordering $b_1 < b_2 < \cdots < b_k$

$i = j$

$\varphi \wedge \psi = \text{ if } b_i \text{ then } \varphi_1 \wedge \psi_1 \text{ else } \varphi_2 \wedge \psi_2$



$\varphi = \text{ if } b_i \text{ then } \varphi_1 \text{ else } \varphi_2$

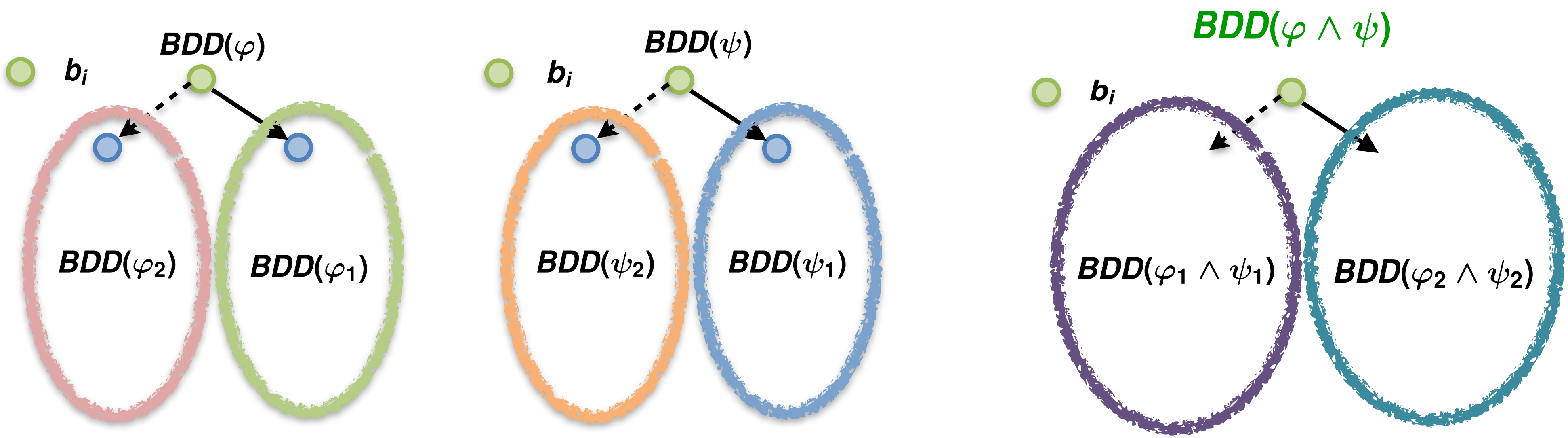
$\psi = \text{ if } b_j \text{ then } \psi_1 \text{ else } \psi_2$

And – Intersection

Fix an ordering $b_1 < b_2 < \cdots < b_k$

$i = j$

$\varphi \wedge \psi =$ if b_i then $\varphi_1 \wedge \psi_1$ else $\varphi_2 \wedge \psi_2$



φ = if b_i then φ_1 else φ_2

ψ = if b_j then ψ_1 else ψ_2

$BDD(f_1(\bar{b}) \wedge f_2(\bar{b})) = BDD(f_1(\bar{b})) \otimes BDD(f_2(\bar{b}))$

OR – Union

Boolean functions $\boldsymbol{f_1(\bar{b})}$ and $\boldsymbol{f_2(\bar{b})}$ ***BDD($f_i(\bar{b})$)*** the ORBDD for $\boldsymbol{f_i(\bar{b})}$

Goal: build BDD for $\boldsymbol{f_1(\bar{b}) \vee f_2(\bar{b})}$

OR – Union

Boolean functions $f_1(\bar{b})$ and $f_2(\bar{b})$ $\quad$ $BDD(f_i(\bar{b}))$ the ORBDD for $f_i(\bar{b})$

Goal: build BDD for $f_1(\bar{b}) \vee f_2(\bar{b})$

$$f_1(\bar{b}) \vee f_2(\bar{b}) = \neg(\neg f_1(\bar{b}) \wedge \neg f_2(\bar{b}))$$

OR – Union

Boolean functions $f_1(\bar{b})$ and $f_2(\bar{b})$ $\quad$ $BDD(f_i(\bar{b}))$ the ORBDD for $f_i(\bar{b})$

Goal: build BDD for $f_1(\bar{b}) \vee f_2(\bar{b})$

$$f_1(\bar{b}) \vee f_2(\bar{b}) = \neg(\neg f_1(\bar{b}) \wedge \neg f_2(\bar{b}))$$

$$BDD(f_1(\bar{b}) \vee f_2(\bar{b})) = BDD(\neg(\neg f_1(\bar{b}) \wedge \neg f_2(\bar{b})))$$

OR – Union

Boolean functions $f_1(\bar{b})$ and $f_2(\bar{b})$ $\quad$ $BDD(f_i(\bar{b}))$ the ORBDD for $f_i(\bar{b})$

Goal: build BDD for $f_1(\bar{b}) \vee f_2(\bar{b})$

$$f_1(\bar{b}) \vee f_2(\bar{b}) = \neg(\neg f_1(\bar{b}) \wedge \neg f_2(\bar{b}))$$

$$BDD(f_1(\bar{b}) \vee f_2(\bar{b})) = BDD(\neg(\neg f_1(\bar{b}) \wedge \neg f_2(\bar{b})))$$

$$BDD(f_1(\bar{b})) \oplus BDD(f_2(\bar{b})) = \overline{\left(\overline{BDD(f_1(\bar{b}))} \otimes \overline{BDD(f_2(\bar{b}))}\right)}$$

Existential Quantification

Boolean function $\boldsymbol{f}(\bar{\boldsymbol{b}})$

$\boldsymbol{BDD(f(\bar{b}))}$ the ORBDD for $\boldsymbol{f(\bar{b})}$

Goal: build BDD for $\exists \boldsymbol{b_j.f(\bar{b})}$

Existential Quantification

Boolean function $\boldsymbol{f(\bar{b})}$

$\boldsymbol{BDD(f(\bar{b}))}$ the ORBDD for $\boldsymbol{f(\bar{b})}$

Goal: build BDD for $\exists \boldsymbol{b_j.f(\bar{b})}$

$$\exists \boldsymbol{b_j.f(\bar{b}) = f(\bar{b})[b_j/0]} \vee \boldsymbol{f(\bar{b})[b_j/1]}$$

Existential Quantification

Boolean function $\boldsymbol{f(\bar{b})}$ $\boldsymbol{BDD(f(\bar{b}))}$ the ORBDD for $\boldsymbol{f(\bar{b})}$

Goal: build BDD for $\boldsymbol{\exists b_j . f(\bar{b})}$

$$\exists b_j . f(\bar{b}) = f(\bar{b})[b_j/0] \vee f(\bar{b})[b_j/1]$$

$$BDD(\exists b_j . f(\bar{b})) = BDD(f(\bar{b})[b_j/0]) \oplus BDD(f(\bar{b})[b_j/1])$$

Existential Quantification

Boolean function $\boldsymbol{f(\bar{b})}$

$\boldsymbol{BDD(f(\bar{b}))}$ the ORBDD for $\boldsymbol{f(\bar{b})}$

Goal: build BDD for $\exists \boldsymbol{b_j.f(\bar{b})}$

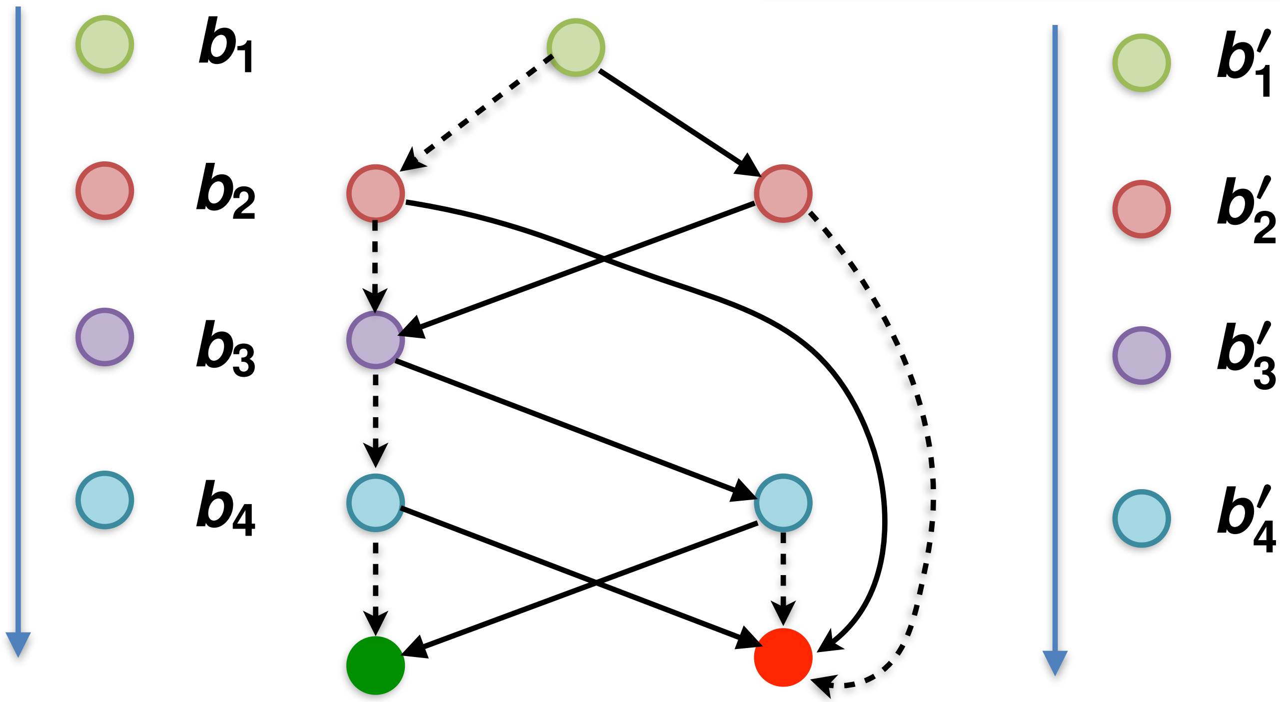
$$\exists b_j.f(\bar{b}) = f(\bar{b})[b_j/0] \vee f(\bar{b})[b_j/1]$$

$$BDD(\exists b_j.f(\bar{b})) = BDD(f(\bar{b})[b_j/0]) \oplus BDD(f(\bar{b})[b_j/1])$$

$$\exists b_j.BDD(f(\bar{b})) = BDD(f(\bar{b})[b_j/0]) \oplus BDD(f(\bar{b})[b_j/1])$$

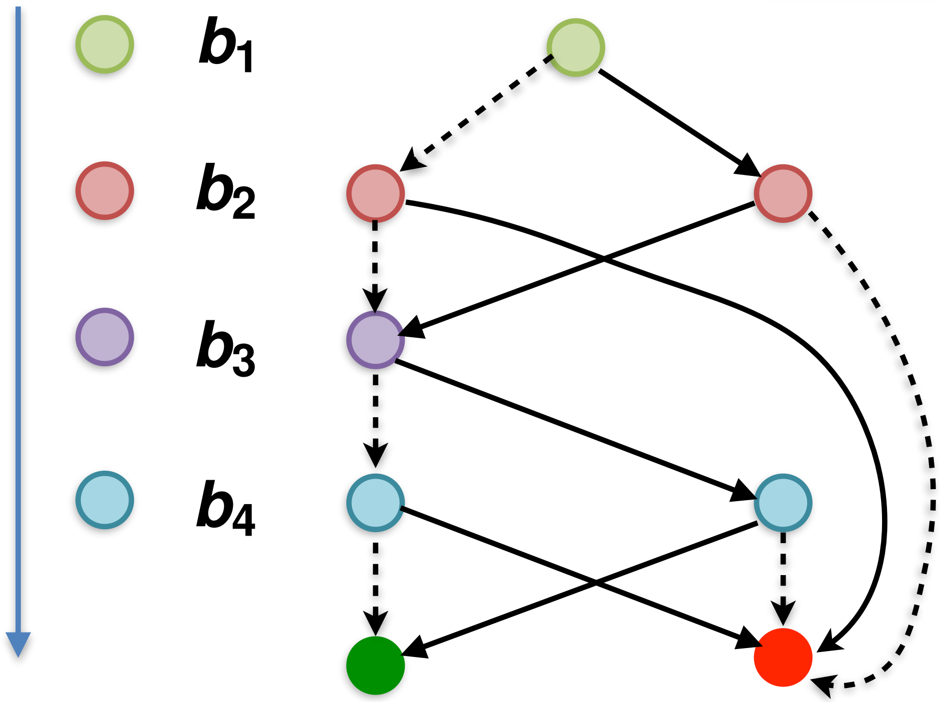
Renaming

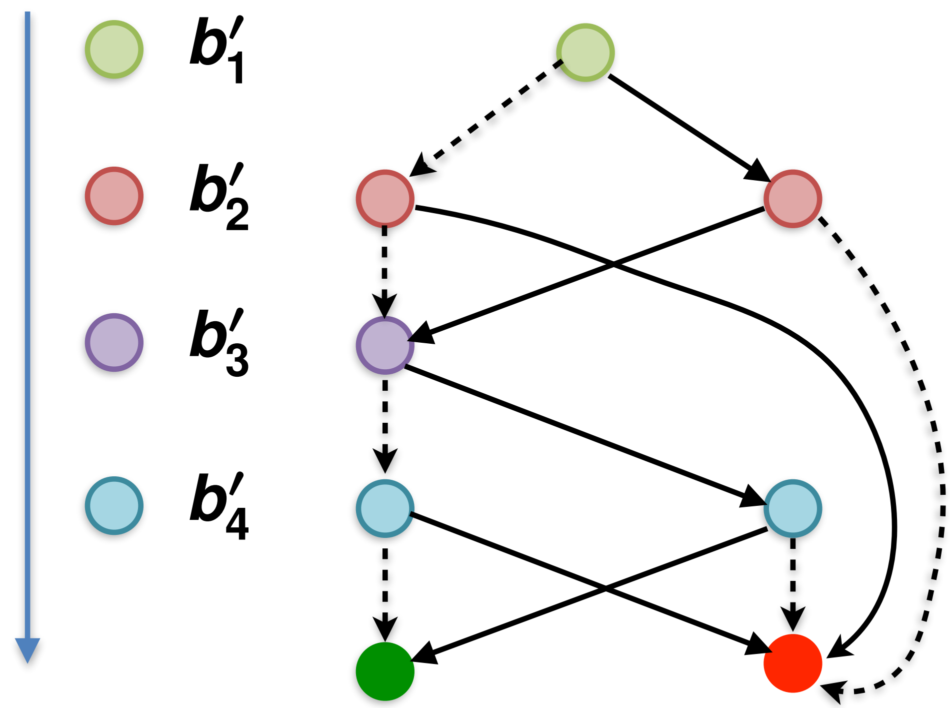
Rename $\bar{\boldsymbol{b}}$: $\bar{\boldsymbol{b}}'$



Renaming

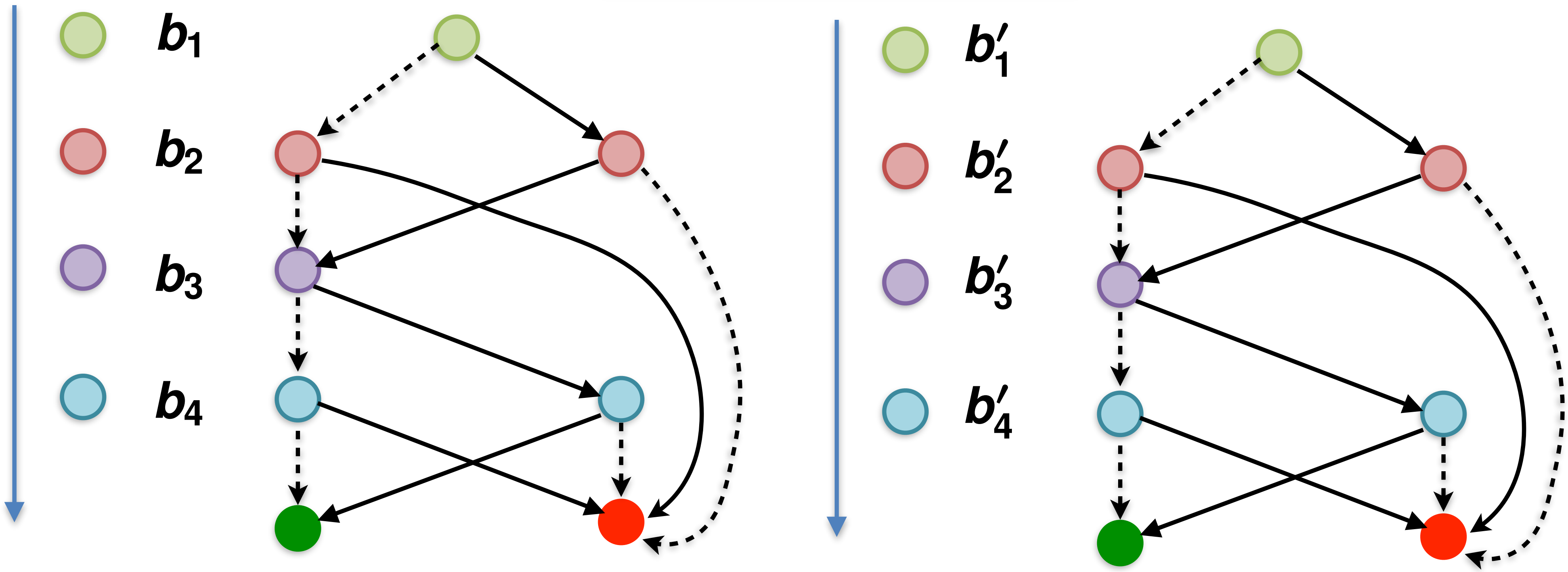
Rename $\bar{b}$: $\bar{b}'$





Renaming

Rename $\bar{b}$: $\bar{b}'$



$$Rename[\bar{b}/\bar{b}'](BDD(f(\bar{b}))) = BDD(f[\bar{b}/\bar{b}'])$$

Operations on BDDs

$\overline{\boldsymbol{BDD}(\boldsymbol{bdd}_1)}$	swap leaves **0** and **1**
Restrict$(\boldsymbol{bdd}_1, \boldsymbol{b}_j, \boldsymbol{v})$	Instantiate $\boldsymbol{b}_j$ with $\boldsymbol{v}$
$\boldsymbol{bdd}_1 \otimes \boldsymbol{bdd}_2$	Conjunction
$\boldsymbol{bdd}_1 \oplus \boldsymbol{bdd}_2$	Disjunction
$\exists \boldsymbol{b}_j.\boldsymbol{bdd}_1$	Existantial quantification on $\boldsymbol{b}_j$
Rename$[\boldsymbol{b}/\boldsymbol{b}'](\boldsymbol{bdd}_1)$	Variables renaming

Properties & Transition Relation

Encoding with BDDs

Atomic Properties as Boolean Formulas

$\boldsymbol{A} = (\boldsymbol{Q}, \boldsymbol{q_0}, \delta, \boldsymbol{L})$

$\boldsymbol{p} \in \mathcal{P}, [\![\boldsymbol{p}]\!] = \boldsymbol{L}^{-1}(\boldsymbol{p})$

Atomic Properties as Boolean Formulas

$\boldsymbol{A = (Q, q_0, \delta, L)}$

$\boldsymbol{p \in \mathcal{P}, [\![p]\!] = L^{-1}(p)}$

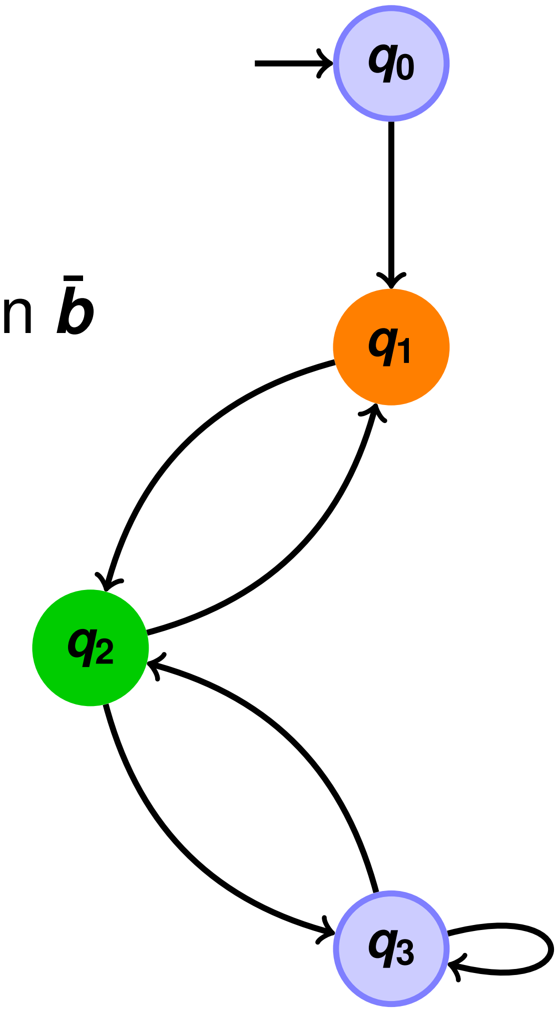
Let $\boldsymbol{BinEnc(v, \bar{b})}$ **=** binary encoding of $\boldsymbol{v}$ on $\boldsymbol{\bar{b}}$

Atomic Properties as Boolean Formulas

$\boldsymbol{A} = (\boldsymbol{Q}, \boldsymbol{q_0}, \delta, \boldsymbol{L})$

$\boldsymbol{p} \in \mathcal{P}, [\![\boldsymbol{p}]\!] = \boldsymbol{L}^{-1}(\boldsymbol{p})$

Let ***BinEnc*(v, $\bar{b}$)** = binary encoding of $\boldsymbol{v}$ on $\bar{\boldsymbol{b}}$



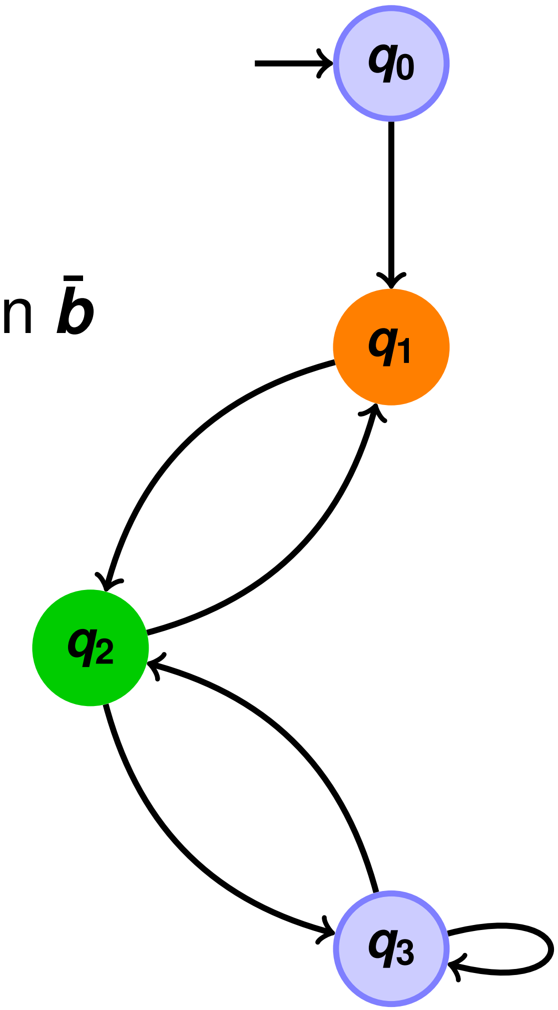
Atomic Properties as Boolean Formulas

$A = (Q, q_0, \delta, L)$

$p \in \mathcal{P}, [\![p]\!] = L^{-1}(p)$

Let $\textit{BinEnc}(v, \bar{b})$ = binary encoding of v on $\bar{b}$

$$f_{[\![\bullet]\!]} \;\equiv\; b_2 = 0 \wedge b_1 = 1$$

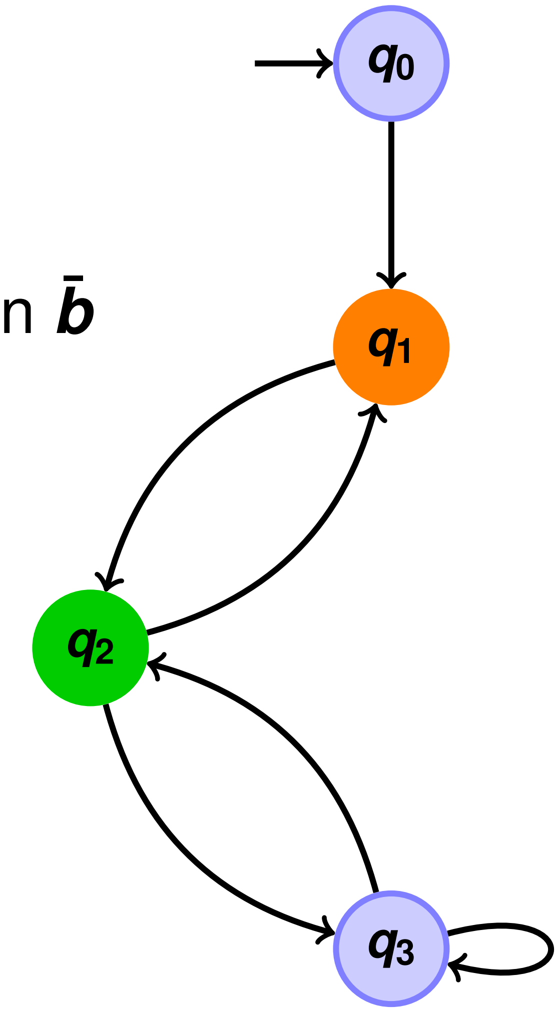


Atomic Properties as Boolean Formulas

$\boldsymbol{A = (Q, q_0, \delta, L)}$

$\boldsymbol{p \in \mathcal{P}, [\![p]\!] = L^{-1}(p)}$

Let ***BinEnc*(*v*, $\bar{b}$)** = binary encoding of ***v*** on $\boldsymbol{\bar{b}}$



$$f_{[\![\textcolor{orange}{\bullet}]\!]} \equiv b_2 = 0 \wedge b_1 = 1$$

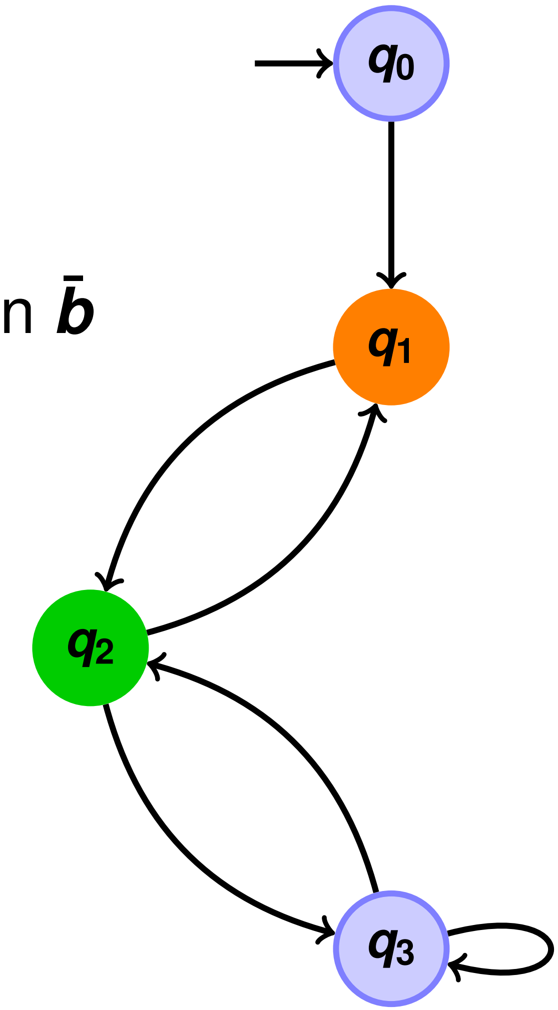
$$f_{[\![\textcolor{green}{\bullet}]\!]} \equiv b_2 = 1 \wedge b_1 = 0$$

Atomic Properties as Boolean Formulas

$$A = (Q, q_0, \delta, L)$$

$$p \in \mathcal{P}, [\![p]\!] = L^{-1}(p)$$

Let $BinEnc(v, \bar{b})$ = binary encoding of v on $\bar{b}$



$$f_{[\![\textcolor{orange}{\bullet}]\!]} \equiv b_2 = 0 \wedge b_1 = 1$$

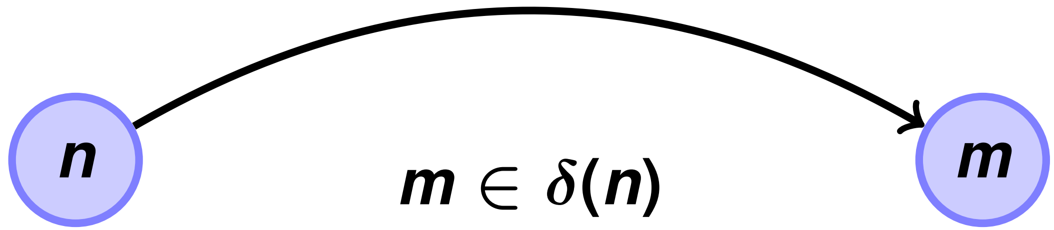
$$f_{[\![\textcolor{green}{\bullet}]\!]} \equiv b_2 = 1 \wedge b_1 = 0$$

$$f_{[\![\textcolor{lavender}{\bullet}]\!]} \equiv (b_2 = b_1 = 0) \vee (b_2 = b_1 = 1)$$

One Transition as Boolean Formula

$\boldsymbol{A = (Q, q_0, \delta, L)}$

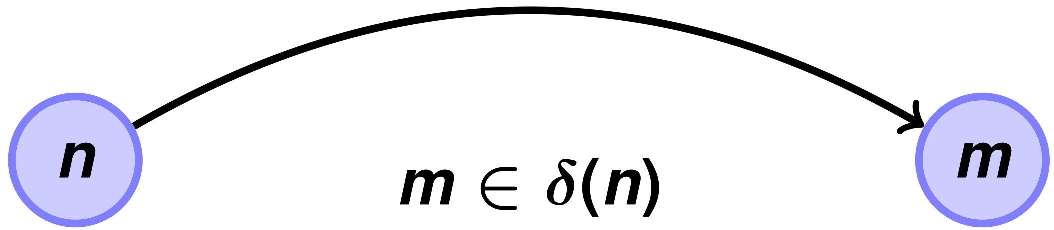
$\delta \subseteq \boldsymbol{Q} \times \boldsymbol{Q}$



One Transition as Boolean Formula

$\boldsymbol{A = (Q, q_0, \delta, L)}$

$\boldsymbol{\delta \subseteq Q \times Q}$

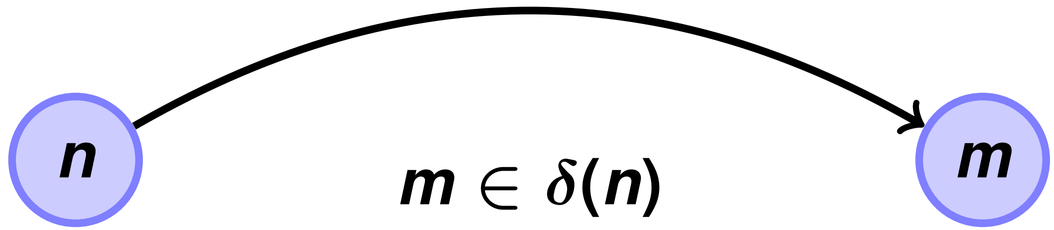


$\boldsymbol{f_\delta(n, m) = BinEnc(n, \bar{b}) \wedge BinEnc(m, \bar{b}')}$

One Transition as Boolean Formula

$A = (Q, q_0, \delta, L)$

$\delta \subseteq Q \times Q$



$f_\delta(n, m) = BinEnc(n, \bar{b}) \wedge BinEnc(m, \bar{b}')$

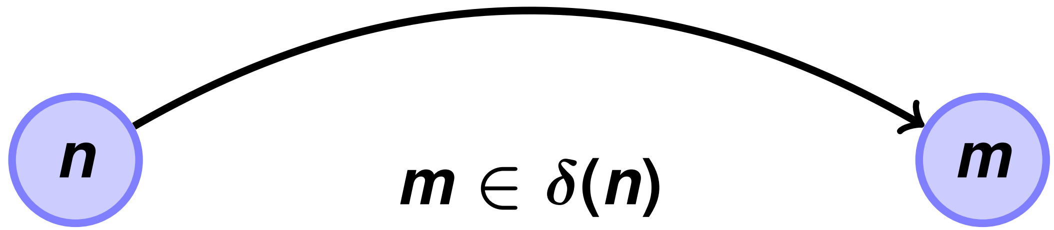
	b_3	b_2	b_1
6	1	1	0

	b'_3	b'_2	b'_1
3	0	1	1

One Transition as Boolean Formula

$A = (Q, q_0, \delta, L)$

$\delta \subseteq Q \times Q$



$f_\delta(n, m) = BinEnc(n, \bar{b}) \wedge BinEnc(m, \bar{b}')$

	b_3	b_2	b_1
6	1	1	0

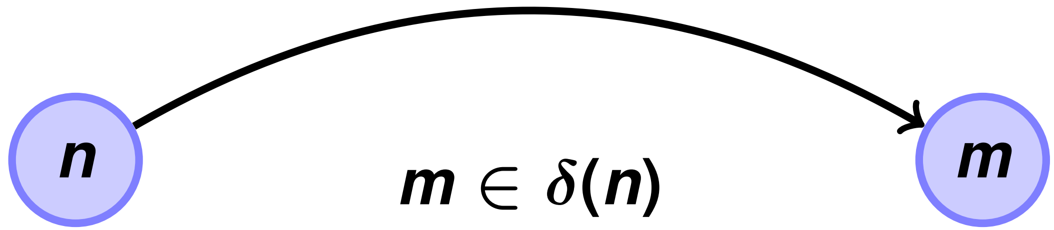
	b'_3	b'_2	b'_1
3	0	1	1

$n = 6, m = 3$

One Transition as Boolean Formula

$A = (Q, q_0, \delta, L)$

$\delta \subseteq Q \times Q$



$f_\delta(n, m) = BinEnc(n, \bar{b}) \wedge BinEnc(m, \bar{b}')$

	b_3	b_2	b_1		b_3'	b_2'	b_1'
6	1	1	0	3	0	1	1

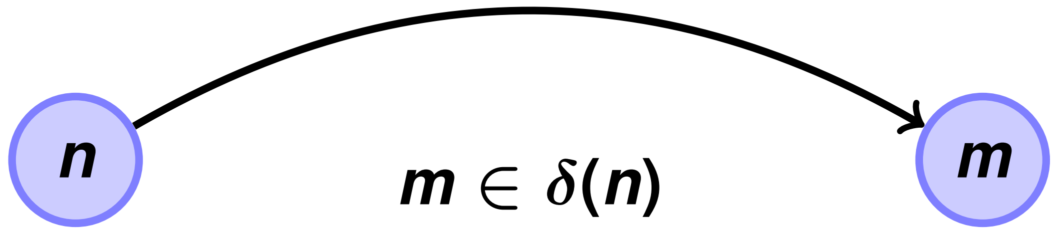
$n = 6, m = 3$

$b_3 \wedge b_2 \wedge \neg b_1 \wedge \neg b_3' \wedge b_2' \wedge b_1'$

One Transition as Boolean Formula

$A = (Q, q_0, \delta, L)$

$\delta \subseteq Q \times Q$



$f_\delta(n, m) = BinEnc(n, \bar{b}) \wedge BinEnc(m, \bar{b}')$

	b_3	b_2	b_1		b'_3	b'_2	b'_1
6	1	1	0	3	0	1	1

$n = 6, m = 3$

$b_3 \wedge b_2 \wedge \neg b_1 \wedge \neg b'_3 \wedge b'_2 \wedge b'_1$

$k = \log_2 |Q|$

$2 \cdot k$ variables $\bar{b}, \bar{b}'$

Transition Relation as Boolean Formula

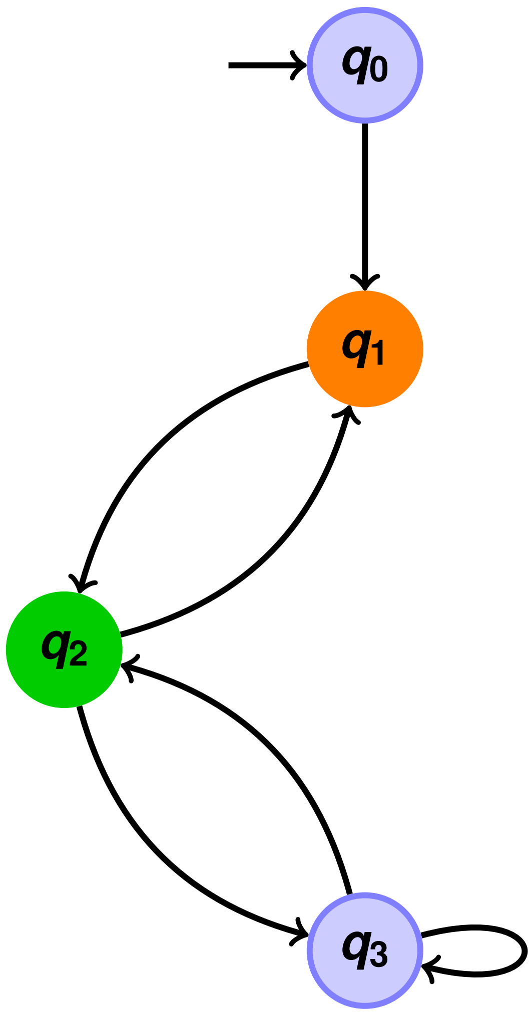
$$\boldsymbol{f_\delta = \bigvee_{n \in Q, m \in \delta(n)} f_\delta(n, m)}$$

Transition Relation as Boolean Formula

$$f_\delta = \bigvee_{n \in Q, m \in \delta(n)} f_\delta(n, m)$$

$$f_\delta(\bar{b}, \bar{b}') = True \iff \bar{b}' \in \delta(\bar{b})$$

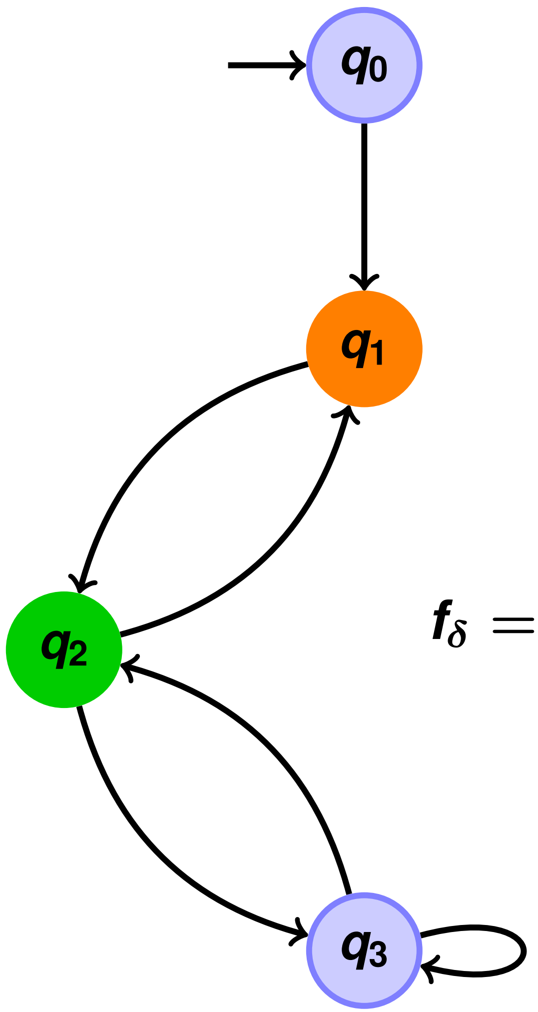
Transition Relation as Boolean Formula



$$f_\delta = \bigvee_{n \in Q, m \in \delta(n)} f_\delta(n, m)$$

$$f_\delta(\bar{b}, \bar{b}') = \mathit{True} \iff \bar{b}' \in \delta(\bar{b})$$

Transition Relation as Boolean Formula



$$f_\delta = \bigvee_{n \in Q, m \in \delta(n)} f_\delta(n, m)$$

$$f_\delta(\bar{b}, \bar{b}') = \mathit{True} \iff \bar{b}' \in \delta(\bar{b})$$

$$\begin{array}{llr}
f_\delta = & (b_2 = b_1 = 0 \wedge b_2' = 0 \wedge b_1' = 1) & (q_0, q_1) \\
\vee & (b_2 = 0 \wedge b_1 = 1 \wedge b_2' = 1 \wedge b_1' = 0) & (q_1, q_2) \\
\vee & (b_2 = 1 \wedge b_1 = 0 \wedge b_2' = 0 \wedge b_1' = 1) & (q_2, q_1) \\
\vee & (b_2 = 1 \wedge b_1 = 0 \wedge b_2' = b_1' = 1) & (q_2, q_3) \\
\vee & (b_2 = b_1 = 1 \wedge b_2' = 1 \wedge b_1' = 0) & (q_3, q_2) \\
\vee & (b_2 = b_1 = 1 \wedge b_2' = b_1' = 1) & (q_3, q_3)
\end{array}$$

Encoding Image

$\boldsymbol{A} = (\boldsymbol{Q}, \boldsymbol{q_0}, \delta, \boldsymbol{L})$

Goal: compute BDD for $\boldsymbol{\delta(X)}$

Encoding Image

$\boldsymbol{A} = (\boldsymbol{Q}, \boldsymbol{q_0}, \delta, \boldsymbol{L})$

Goal: compute BDD for $\delta(\boldsymbol{X})$

$\boldsymbol{X}$ $\quad$ $f_X(\bar{b})$ $\quad$ bdd_X

Encoding Image

$\boldsymbol{A = (Q, q_0, \delta, L)}$

Goal: compute BDD for $\boldsymbol{\delta(X)}$

$\boldsymbol{X}$	$\boldsymbol{f_X(\bar{b})}$	$\boldsymbol{bdd_X}$
δ	$\boldsymbol{f_\delta(\bar{b}, \bar{b}')}$	$\boldsymbol{bdd_\delta}$

Encoding Image

$\boldsymbol{A = (Q, q_0, \delta, L)}$

Goal: compute BDD for $\delta(X)$

X	$f_X(\bar{b})$	bdd_X
δ	$f_\delta(\bar{b}, \bar{b}')$	bdd_δ

$$f_{\delta(X)} = \big(\exists \bar{b}.(f_X(\bar{b}) \wedge f_\delta)\big)[\bar{b}' / \bar{b}]$$

Encoding Image

$A = (Q, q_0, \delta, L)$

Goal: compute BDD for $\delta(X)$

X	$f_X(\bar{b})$	bdd_X
δ	$f_\delta(\bar{b}, \bar{b}')$	bdd_δ

$$f_{\delta(X)} = \left(\exists \bar{b}.(f_X(\bar{b}) \wedge f_\delta)\right)[\bar{b}' / \bar{b}]$$

$$bdd_{\delta(X)} = \text{Rename}[\bar{b}' / \bar{b}]\left(\exists \bar{b}.(bdd_X \otimes bdd_\delta)\right)$$

Encoding Pre-Image

$A = (Q, q_0, \delta, L)$

Goal: compute BDD for $\delta^{-1}(X)$

Encoding Pre-Image

$A = (Q, q_0, \delta, L)$

Goal: compute BDD for $\delta^{-1}(X)$

X $\quad f_X(\bar{b})$ $\quad bdd_X$

Encoding Pre-Image

$\boldsymbol{A = (Q, q_0, \delta, L)}$

Goal: compute BDD for $\delta^{-1}(X)$

X	$f_X(\bar{b})$	bdd_X
δ	$f_\delta(\bar{b}, \bar{b}')$	bdd_δ

Encoding Pre-Image

$\boldsymbol{A = (Q, q_0, \delta, L)}$

Goal: compute BDD for $\delta^{-1}(X)$

X	$f_X(\bar{b})$	bdd_X
δ	$f_\delta(\bar{b}, \bar{b}')$	bdd_δ

$$f_{\delta^{-1}(X)} = \exists \bar{b}'.(f_X[\bar{b}/\bar{b}'] \wedge f_\delta)$$

Encoding Pre-Image

$\boldsymbol{A} = (\boldsymbol{Q}, \boldsymbol{q_0}, \delta, \boldsymbol{L})$

Goal: compute BDD for $\delta^{-1}(X)$

X	$f_X(\bar{b})$	bdd_X
δ	$f_\delta(\bar{b}, \bar{b}')$	bdd_δ

$$f_{\delta^{-1}(X)} = \exists \bar{b}'.(f_X[\bar{b}/\bar{b}'] \wedge f_\delta)$$

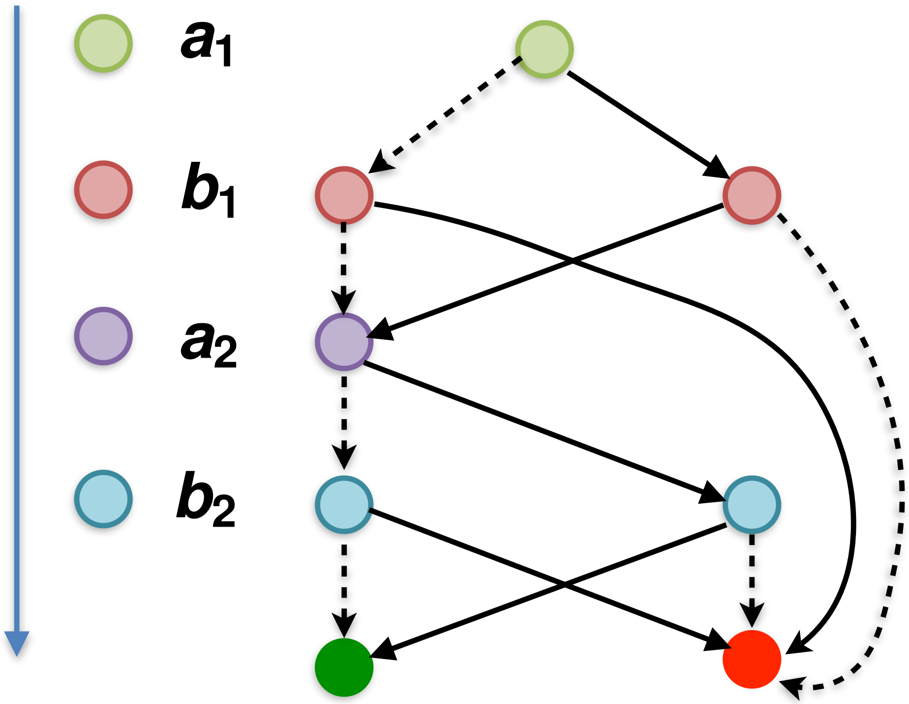
$$bdd_{\delta^{-1}(X)} = \exists \bar{b}'.(\mathrm{Rename}[\bar{b}/\bar{b}'](bdd_X) \otimes bdd_\delta)$$

Model Checking with BDDs

Checking Equality

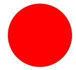
***BDD*(*False*) =**

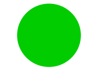
***BDD*(*True*) =**

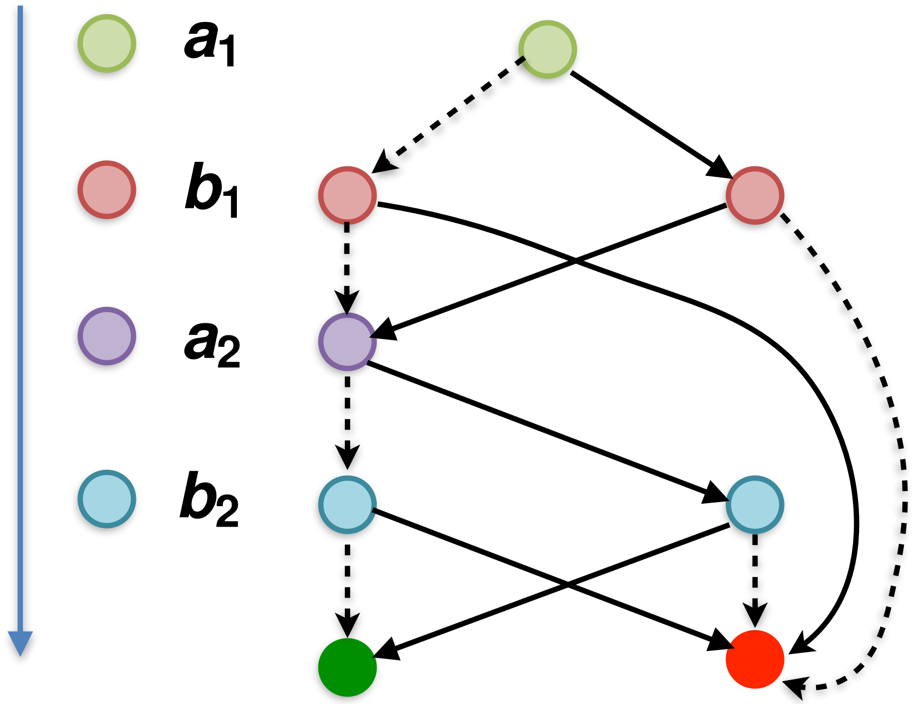


$a_1 = b_1 \wedge a_2 = b_2$

Checking Equality

$BDD(False) =$ 

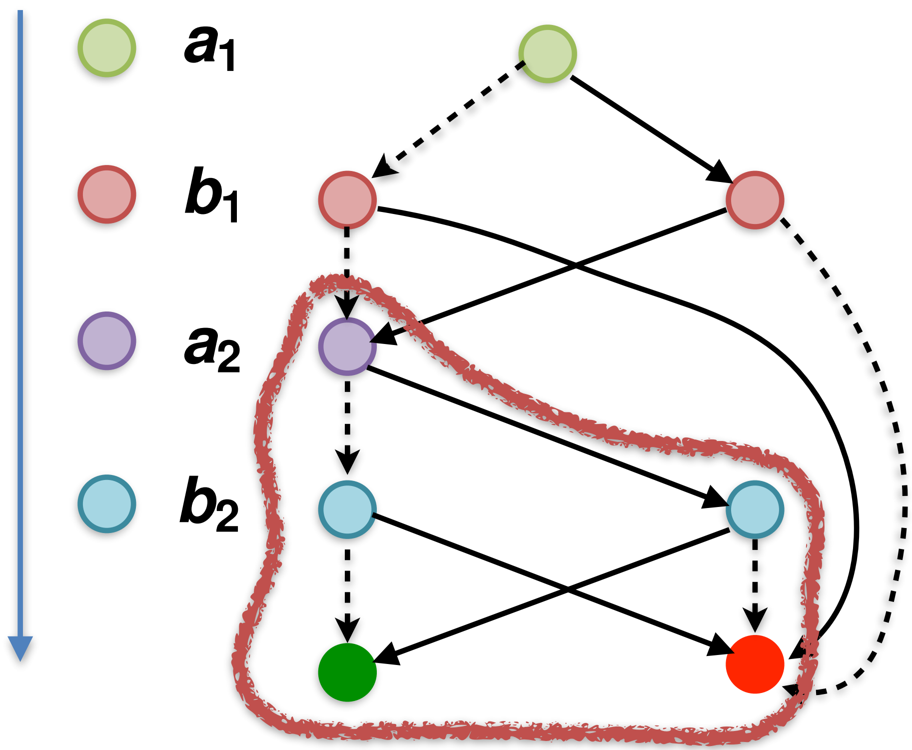
$BDD(True) =$ 



$a_1 = b_1 \wedge a_2 = b_2$

Checking Equality

$BDD(False) =$ ●　　　　$BDD(True) =$ ●

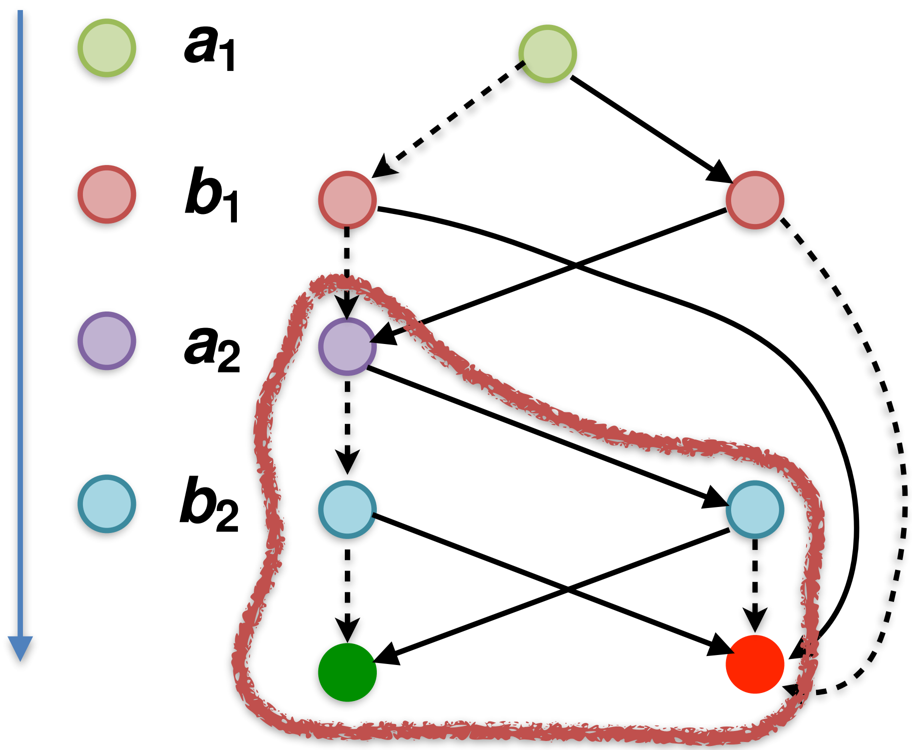


$a_1 = b_1 \wedge a_2 = b_2$

Checking Equality

***BDD*(*False*) =** ●

***BDD*(*True*) =** ●

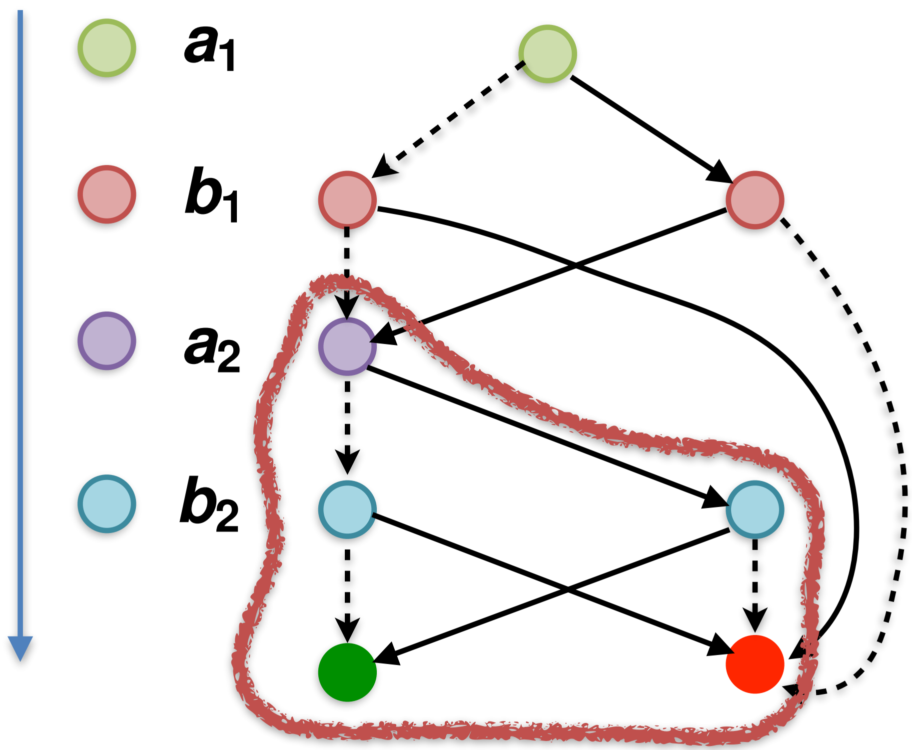


$a_1 = b_1 \wedge a_2 = b_2$

Each BDD uniquely represented in memory

Checking Equality

***BDD*(*False*) =**   ***BDD*(*True*) =**



$a_1 = b_1 \wedge \boxed{a_2 = b_2}$

Consequences:

- Checking equality: constant time
- Hash table to store previously computed BDDs

Each BDD uniquely represented in memory

BDD-based Model Checking Algorithm

Given: $\boldsymbol{A = (Q, q_0, \delta, L)}$ and $\varphi \in \boldsymbol{CTL}$

BDD-based Model Checking Algorithm

Given: $\boldsymbol{A = (Q, q_0, \delta, L)}$ and $\varphi \in \boldsymbol{CTL}$

SMC(φ) computes $[\![\varphi]\!] = \{\boldsymbol{q} \in \boldsymbol{Q} \mid \boldsymbol{q} \models \varphi\}$

BDD-based Model Checking Algorithm

Given: $\boldsymbol{A} = (\boldsymbol{Q}, \boldsymbol{q_0}, \delta, \boldsymbol{L})$ and $\varphi \in \boldsymbol{CTL}$

$\boldsymbol{SMC}(\varphi)$ computes $[\![\varphi]\!] = \{\boldsymbol{q} \in \boldsymbol{Q} \mid \boldsymbol{q} \models \varphi\}$

$\boldsymbol{SMC}$-$\boldsymbol{BDD}(\varphi)$ computes $\boldsymbol{BDD}([\![\varphi]\!])$

BDD-based Model Checking Algorithm

Given: $\boldsymbol{A = (Q, q_0, \delta, L)}$ and $\varphi \in \boldsymbol{CTL}$

$\boldsymbol{SMC}(\varphi)$ computes $[\![\varphi]\!] = \{\boldsymbol{q} \in \boldsymbol{Q} \mid \boldsymbol{q} \models \varphi\}$

$\boldsymbol{SMC\text{-}BDD}(\varphi)$ computes $\boldsymbol{BDD}([\![\varphi]\!])$

$$\boldsymbol{A} \models \varphi \iff \boldsymbol{q_0} \models \varphi \iff \boldsymbol{q_0} \in [\![\varphi]\!]$$

BDD-based Model Checking Algorithm

Given: $\boldsymbol{A = (Q, q_0, \delta, L)}$ and $\varphi \in \boldsymbol{CTL}$

$\boldsymbol{SMC}(\varphi)$ computes $[\![\varphi]\!] = \{\boldsymbol{q} \in \boldsymbol{Q} \mid \boldsymbol{q} \models \varphi\}$

$\boldsymbol{SMC\text{-}BDD}(\varphi)$ computes $\boldsymbol{BDD}([\![\varphi]\!])$

$$\boldsymbol{A} \models \varphi \iff \boldsymbol{q_0} \models \varphi \iff \boldsymbol{q_0} \in [\![\varphi]\!]$$

$$\boldsymbol{q_0} \in [\![\varphi]\!] \iff \boldsymbol{BDD}(f_{q_0}) \otimes \boldsymbol{BDD}([\![\varphi]\!]) \neq \boldsymbol{BDD}(\boldsymbol{False})$$