Binary Decision Diagrams

Bettina Könighofer

IAIK – Graz University of Technology
bettina.koenighofer@iaik.tugraz.at

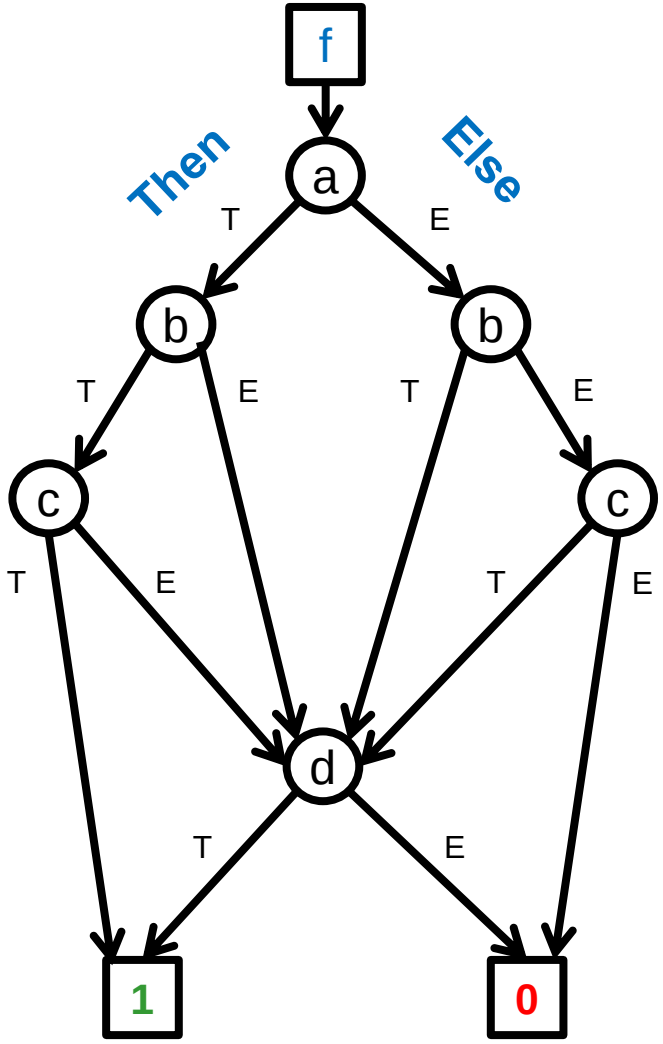
Motivation

- Efficient Representation of Boolean Formula
  - Small, for many practical cases
  - Efficient Manipulation
    - Boolean Operations

Outline

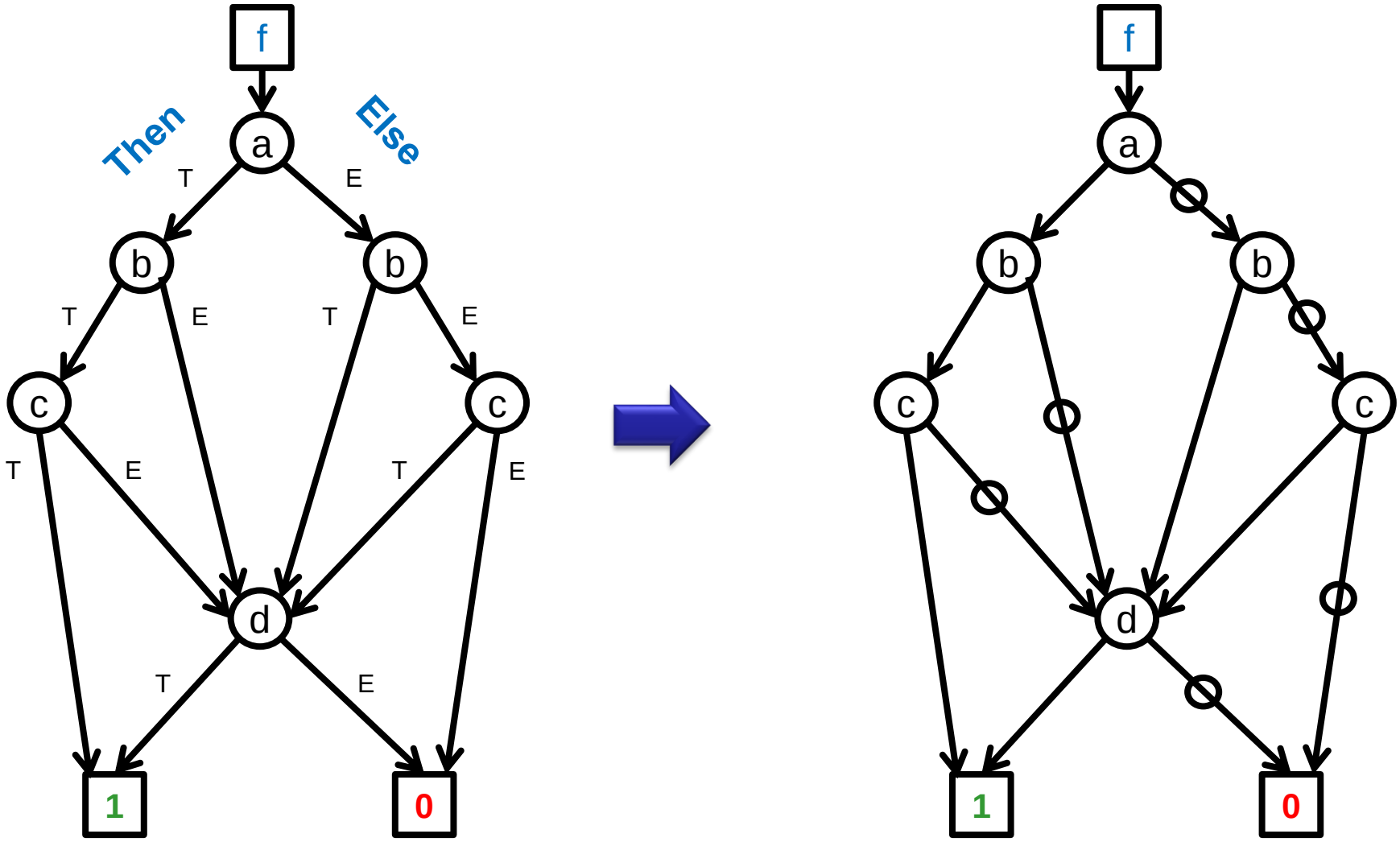
- Intuitive Example
- Formal Definition
- Construct Formula from BDD
- Construct BDD from Formula
- Properties (Advantages, Disadvantages)

Example: A simple BDD

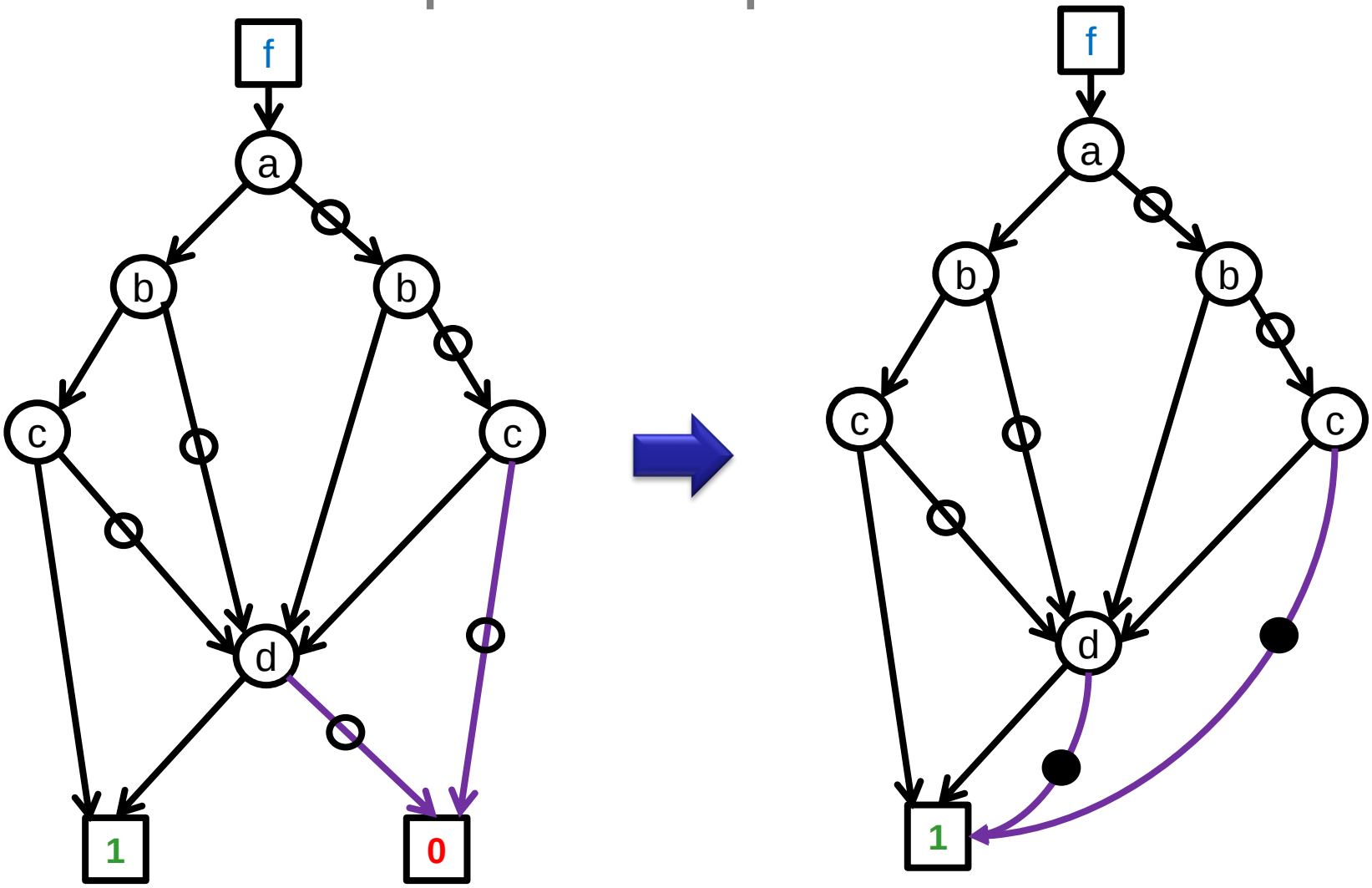


$$M := \{a = T, b = T, c = T, d = T\}$$

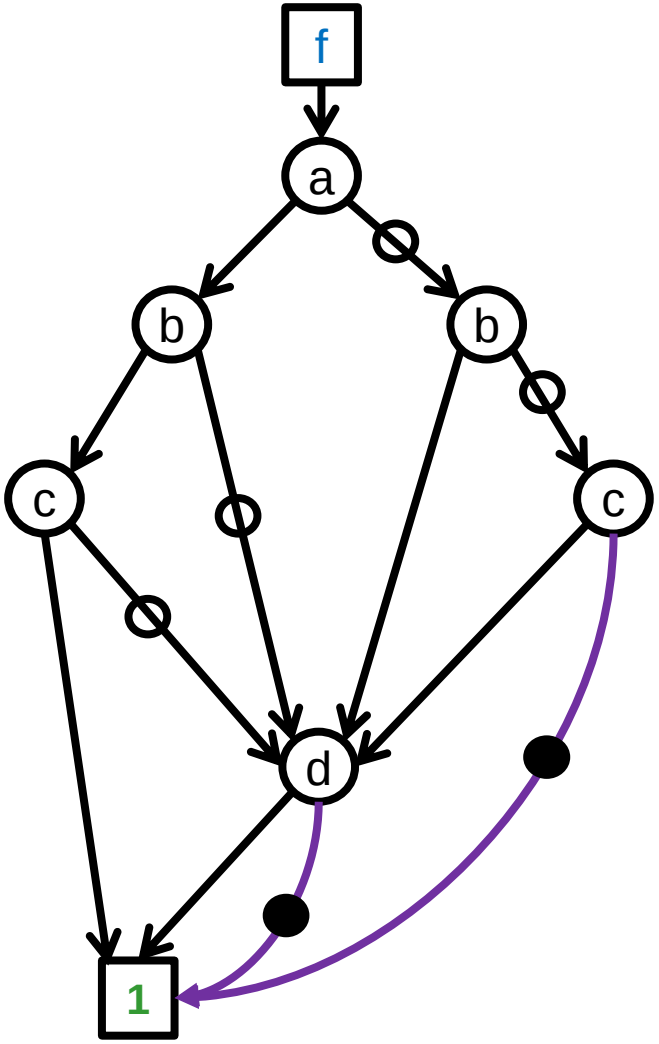
M is a satisfying assignment


f
a
Then
Else
T
E
b
b
T
E
T
E
c
c
T
E
T
E
d
T
E
1
0
f
a
b
b
c
c
d
1
0

Example: Complement Attribute

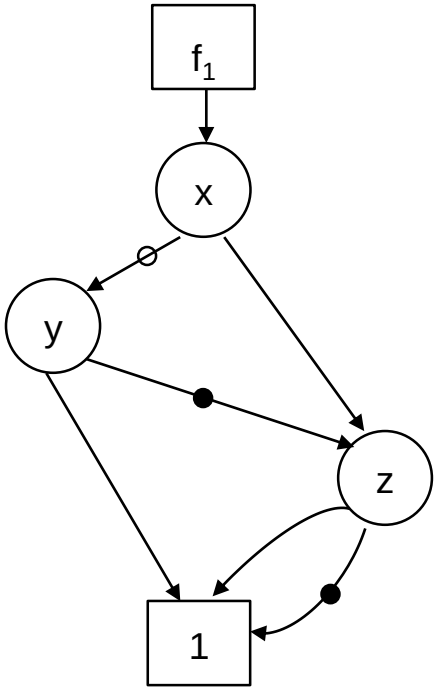


From now on…



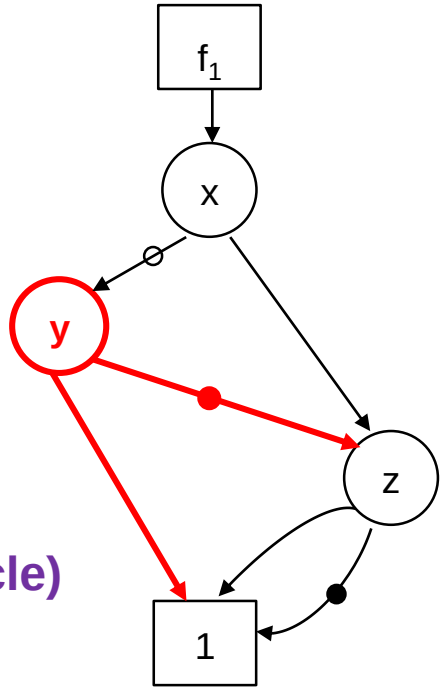
Formal Definition of a BDD

- Directed Acyclic Graph
- $(V \cup \Phi \cup \{\mathbf{1}\},\ E)$
  - Internal Nodes $v \in V$
  - Function Nodes $f_i \in \Phi$
  - Terminal Node $\mathbf{1}$
  - Edges E
    - "Complement" attribute



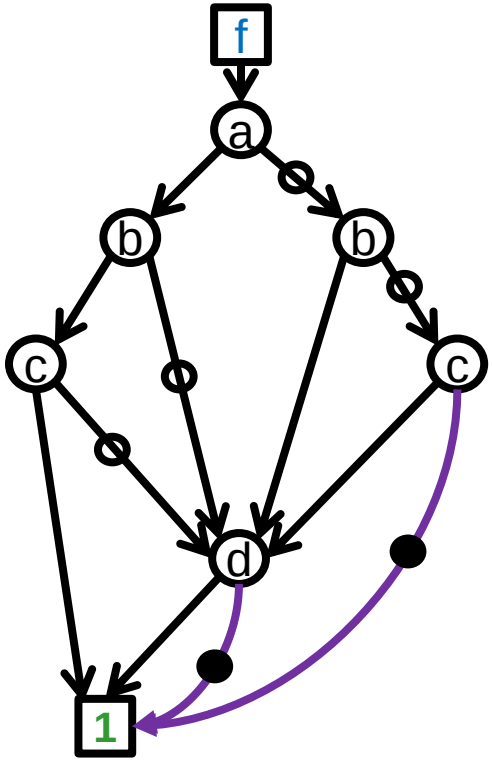
Internal Node

- Label $l(v) \in \{x_1, \ldots, x_n\}$
  - Variables of f
- Out-degree: 2
  - Then-Edge T
  - Else-Edge E
    - Marked with (empty) circle
    - **Can have complement attribute (full circle)**



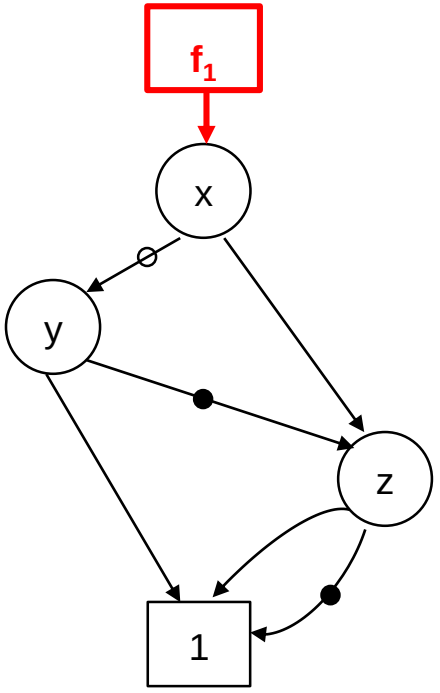
Internal Node

- Label $l(v) \in \{x_1, \ldots, x_n\}$
  - Variables of f
- Out-degree: 2
  - Then-Edge T
  - Else-Edge E
    - Marked with (empty) circle
    - **Can have complement attribute (full circle)**



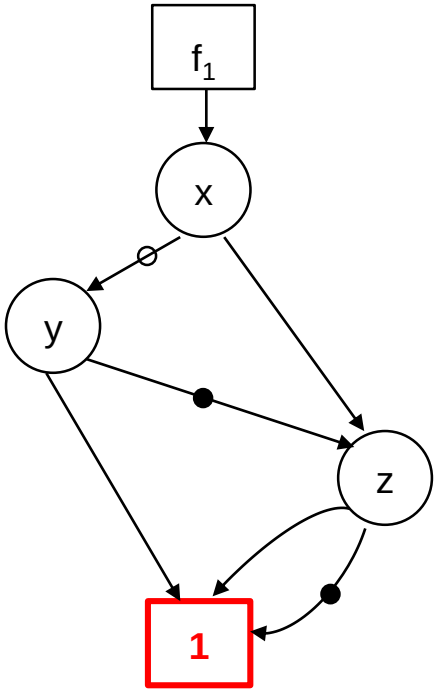
Function Node

- Single Boolean Function f_i
- In-degree: 0
- Out-degree: 1
  - Edge can have complement attribute

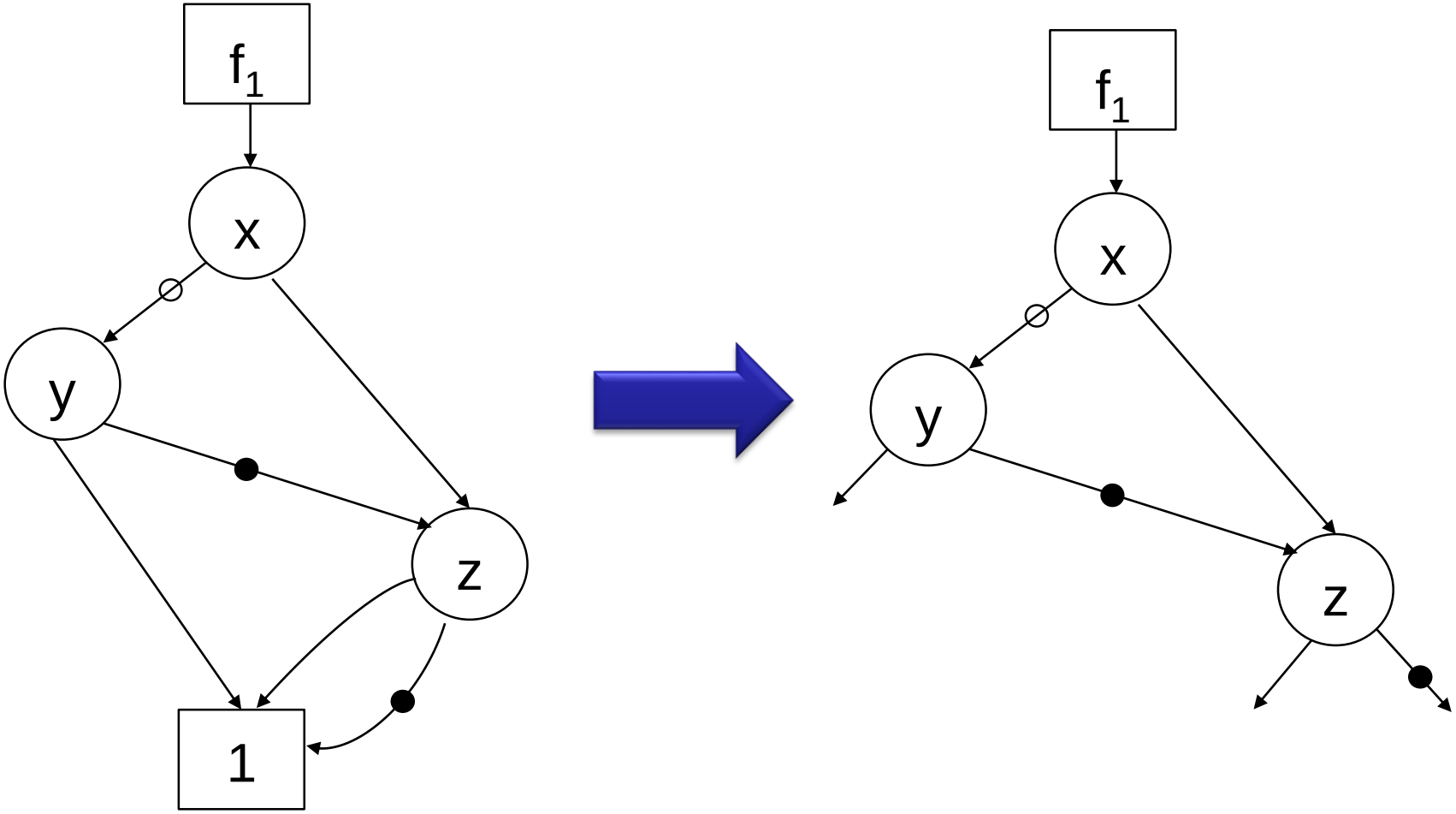


Terminal Node

- Constant Function **TRUE**
- Out-degree: 0



BDD with tangling Edges



Size of BDD

- Worst case expoenential

- Often: BDDs contain a lot of redundancy. Eliminating redundancy results in small BDD.

Reduced Ordered BDDs (ROBDD)

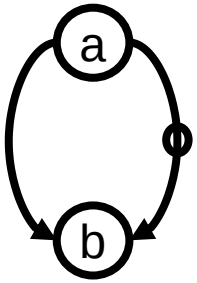
- No duplicate sub-BDDs
- No redundant nodes

- **Canonical**
  - For given variable ordering

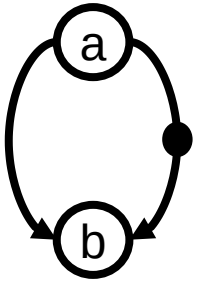
Henceforth: BDD = ROBDD
(unless explicitly stated otherwise)

Redundancy

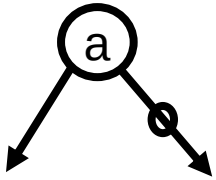
Redundant



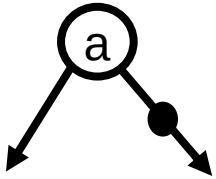
NOT Redundant



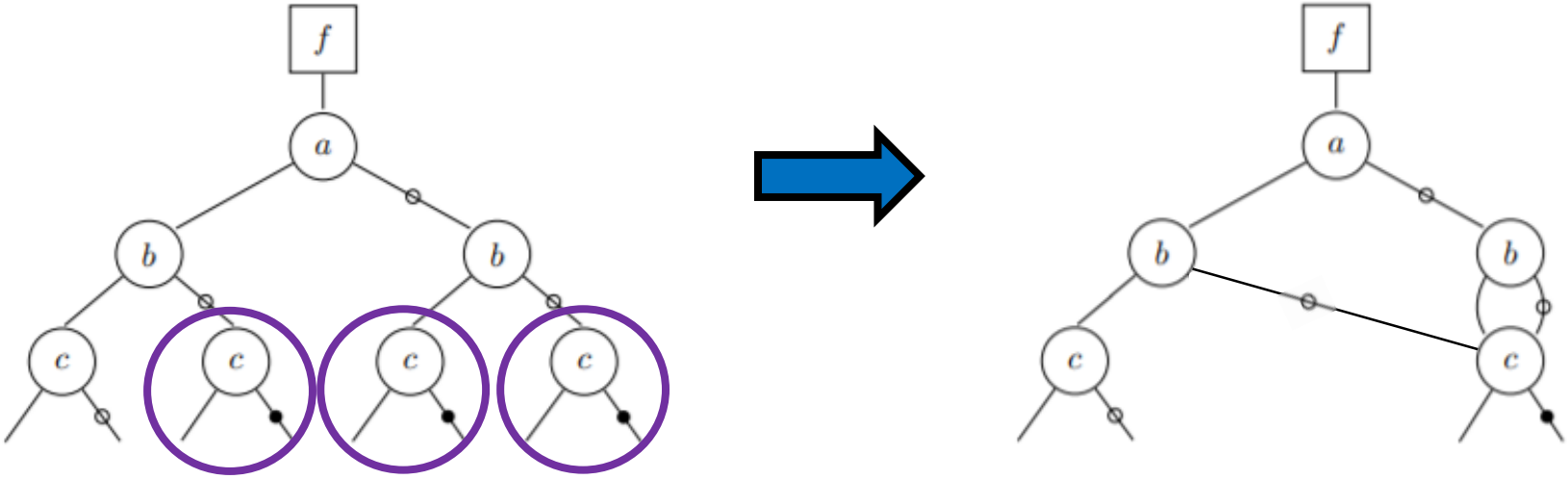
Redundant
(special case)



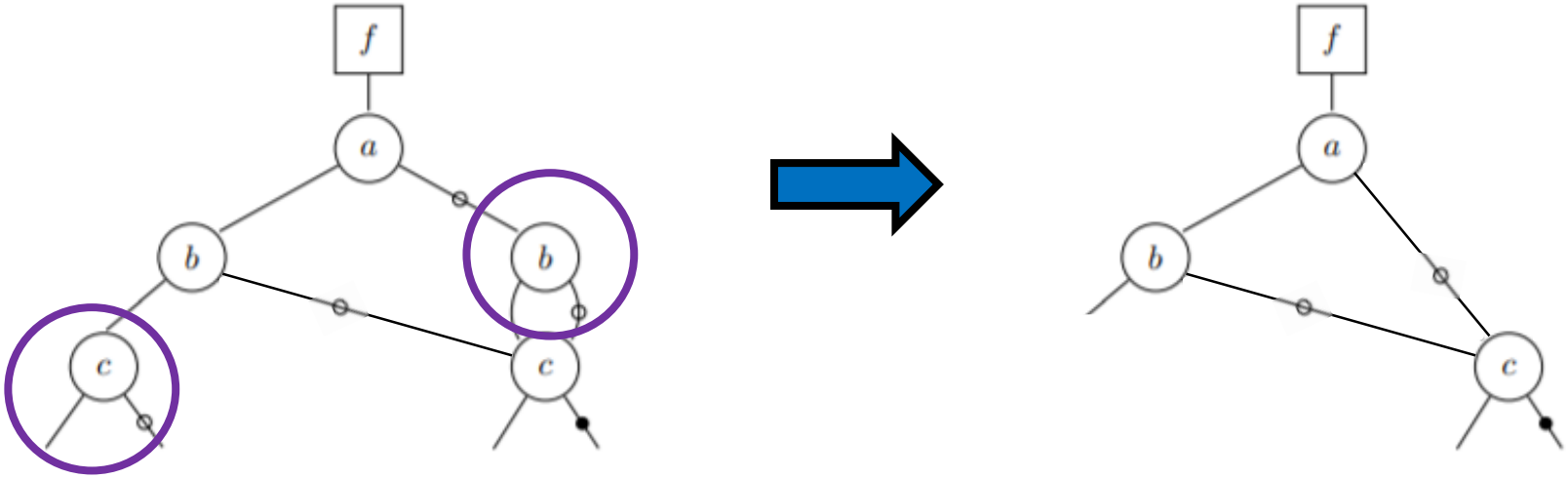
NOT Redundant
(special case)



No Duplicate Sub-BDD



No Redundant Nodes

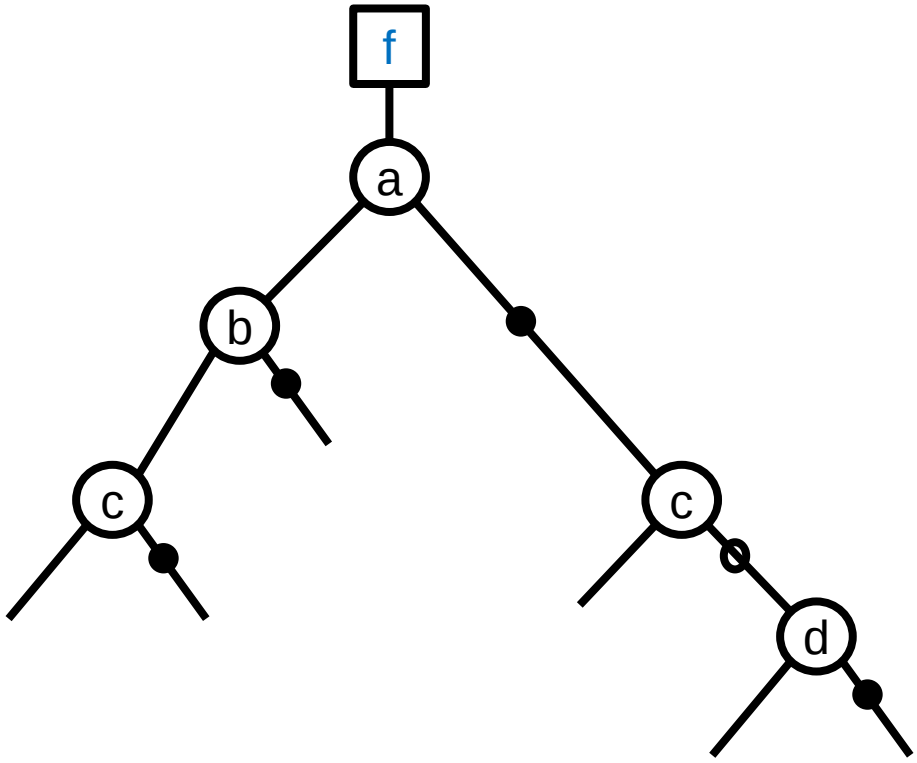


Ordered BDD

- Ordering on the variables along any path
  - e.g., $a < b < c <$ ...

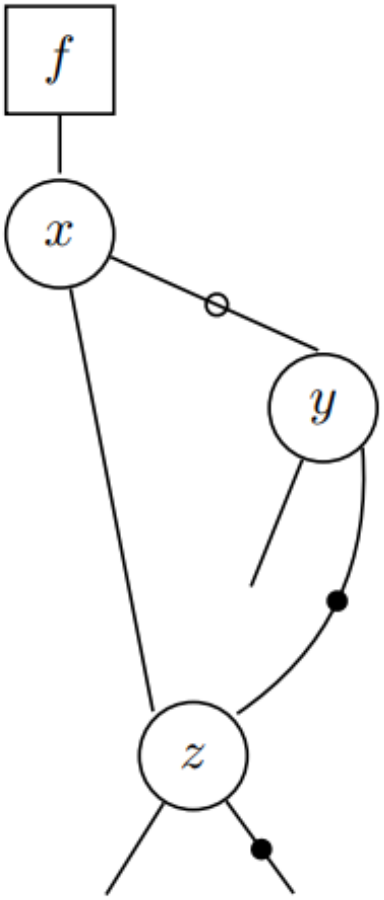
- Variable order has big impact on BDD size

From ROBDD to Formula



$$f = (a \wedge b \wedge c) \vee (\neg a \wedge \neg c \wedge \neg d)$$

From ROBDD to Formula



$$f := (\neg x \wedge y) \vee (\neg x \wedge \neg y \wedge \neg z) \vee (x \wedge z).$$

From Formula to ROBDD

1. Compute all Cofactors
2. Draw BDD from Cofactors.
3. Shift Negations Upwards

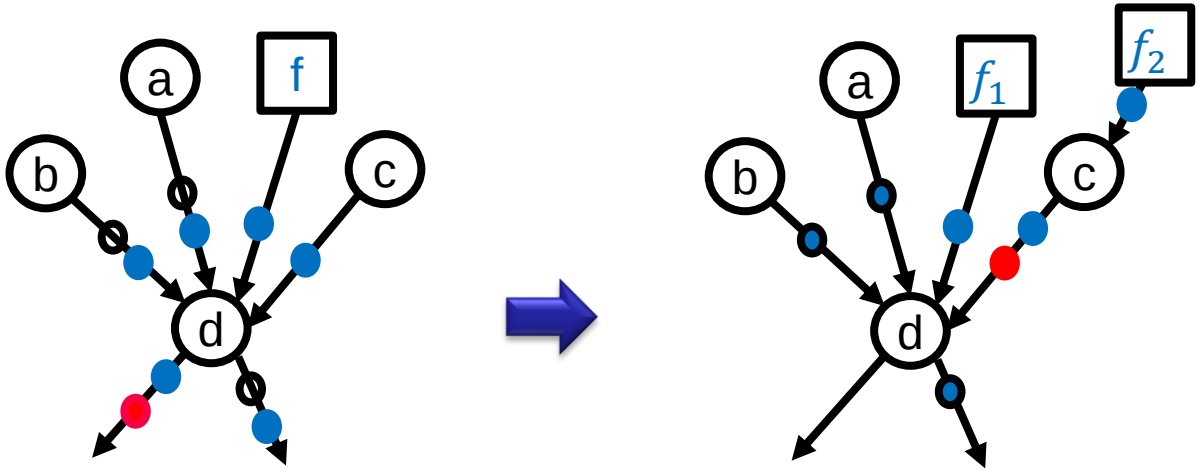
Step 1: Cofactors

- Boolean Formula f
- w.r.t. a variable x
  - Positive Cofactor f_x: f with x set to $\top$
  - Negative Cofactor $f_{\neg x}$: f with x set to $\bot$

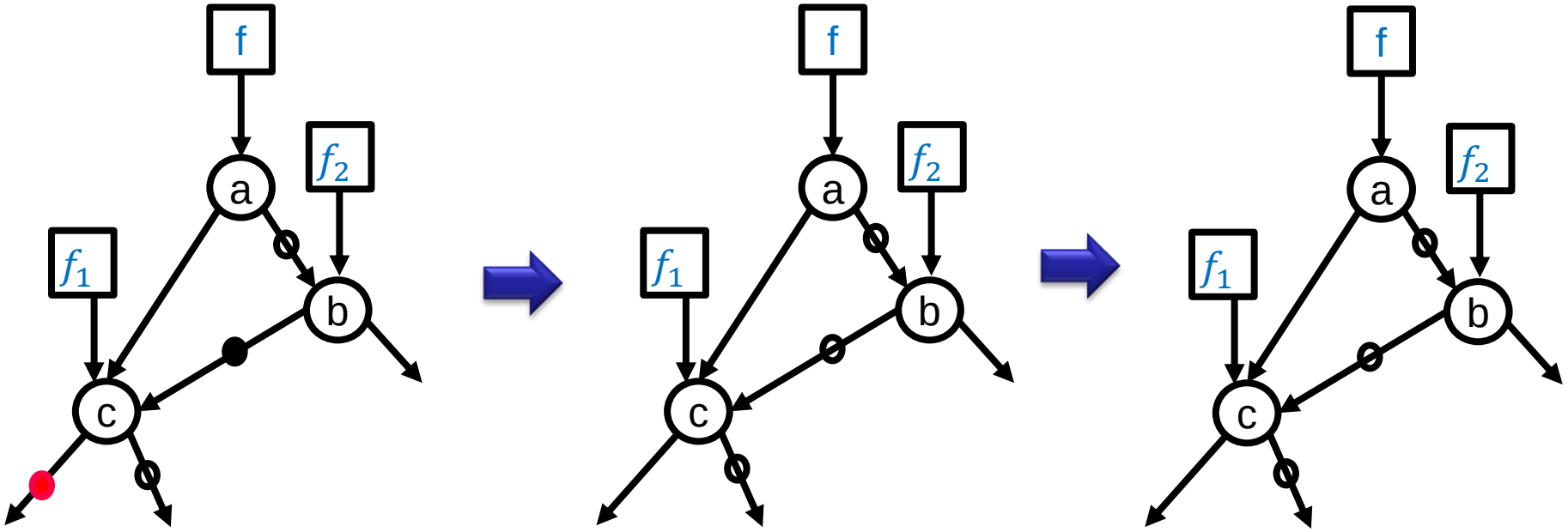
- Example:
  - $f = (x \wedge y) \vee (\neg x \wedge z)$
    - $f_x = y$
    - $f_{\neg x} = z$

Step 3: Dealing with illegal Complements

- „Problem“ pushed upwards
- →Repeat



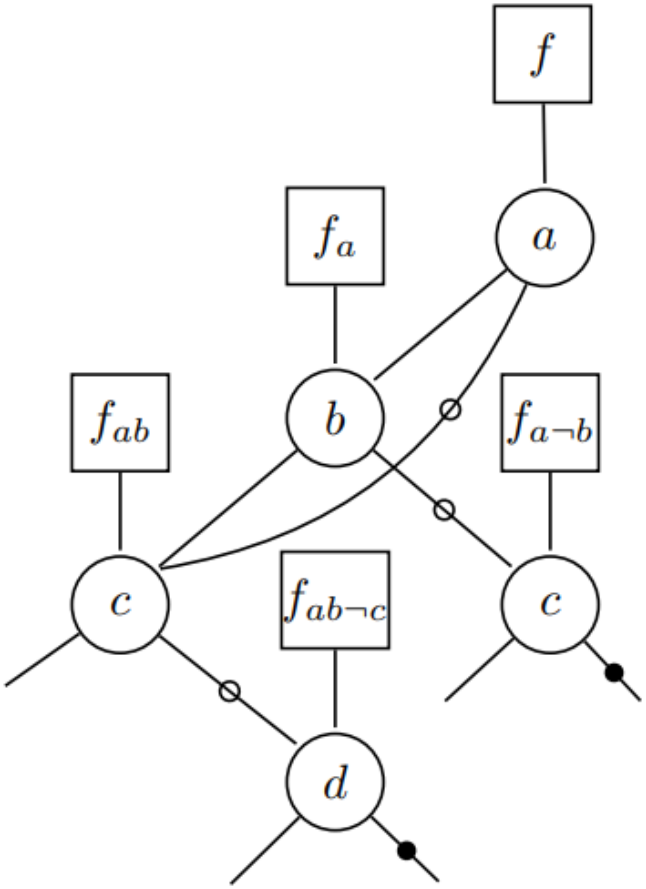
Step 3: Dealing with illegal Complements



Compute BDD from Formula

$$f = (a \wedge b \vee \neg a) \wedge \neg c \wedge d \vee c$$

$$f = (a \wedge b \vee \neg a) \wedge \neg c \wedge d \vee c$$
$$f_a = b \wedge \neg c \wedge d \vee c$$
$$f_{ab} = \neg c \wedge d \vee c$$
$$f_{abc} = \top$$
$$f_{ab\neg c} = d$$
$$f_{ab\neg cd} = \top$$
$$f_{ab\neg c\neg d} = \bot$$
$$f_{a\neg b} = c$$
$$f_{a\neg bc} = \top$$
$$f_{a\neg b\neg c} = \bot$$
$$f_{\neg a} = \neg c \wedge d \vee c = f_{ab}$$



Compute BDD from Formula

$$f = (a \wedge \neg c) \vee (\neg a \wedge (b \vee (\neg b \wedge c)))$$

$$f = (a \wedge \neg c) \vee (\neg a \wedge (b \vee (\neg b \wedge c)))$$

$$f_a = \neg c$$

$$f_{ab} = \neg c = f_a$$

$$f_{abc} = \bot$$

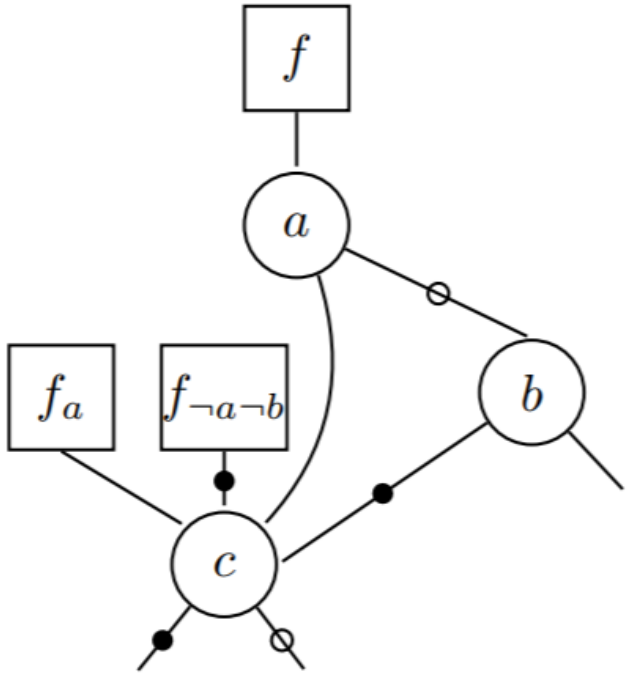
$$f_{ab\neg c} = \top$$

$$f_{a\neg b} = \neg c = f_a$$

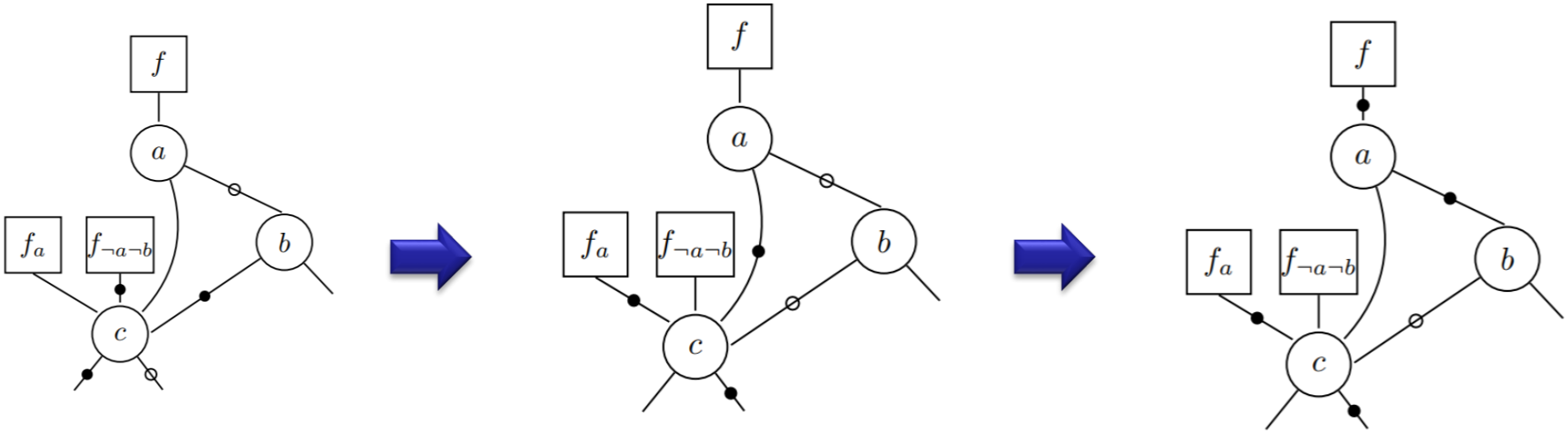
$$f_{\neg a} = b \vee (\neg b \wedge c)$$

$$f_{\neg ab} = \top$$

$$f_{\neg a\neg b} = c = \neg f_{ab}$$



Compute BDD from Formula

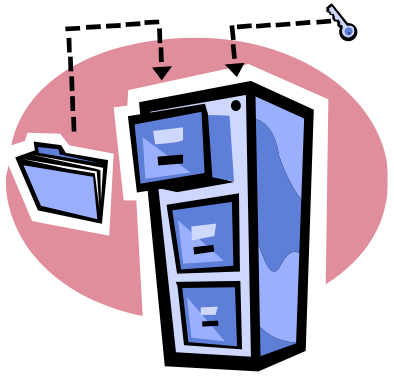


Advantages/ Disadvantages of BDDs

+ Size-Efficiency
  - Worst case: exponential
  - Efficient for many practical examples
+ Efficient Operations
  - e.g. AND, OR: Polynomial time
  - Equivalence Check: Constant time
  - Satisfiability and Validity Check: Constant Time

– Variable order
  - Big impact
  - Hard to optimize

Summary



- BDD = Data structure for Boolean Formulas
- BDD Construction
  - Via Cofactors
- BDD Properties
  - Canonicity, No redundancy
  - Efficient manipulations