Formal Methods

Lecture VI

Binary Decision Diagrams (BDD's)

Alessandro Artale

Faculty of Computer Science – Free University of Bolzano

artale@inf.unibz.it   http://www.inf.unibz.it/∼artale/

Some material (text, figures) displayed in these slides is courtesy of:
M. Benerecetti, A. Cimatti, M. Fisher, F. Giunchiglia, M. Pistore, M. Roveri, R.Sebastiani.

Summary of Lecture VI

- Motivations.
- Ordered Binary Decision Diagrams (OBDD).
- OBDD's as Canonical Forms.
- Building OBDD's.

State Space Explosion

The bottleneck:

- Exhaustive analysis may require to store all the states of the Kripke structure, and to explore them one-by-one.
- The state space may be exponential in the number of components and variables
  (E.g., 300 boolean vars $\Rightarrow$ up to $2^{300} \approx 10^{100}$ states!)
- State Space Explosion:
  - Too much memory required;
  - Too much CPU time required to explore each state.
- A solution: Symbolic Model Checking.

Symbolic Model Checking: Intuitions

- Symbolic representation of Set of states by formulae in propositional logic.
  - manipulation of sets of states, rather than single states;
  - manipulation of sets of transitions, rather than single transitions.

Summary

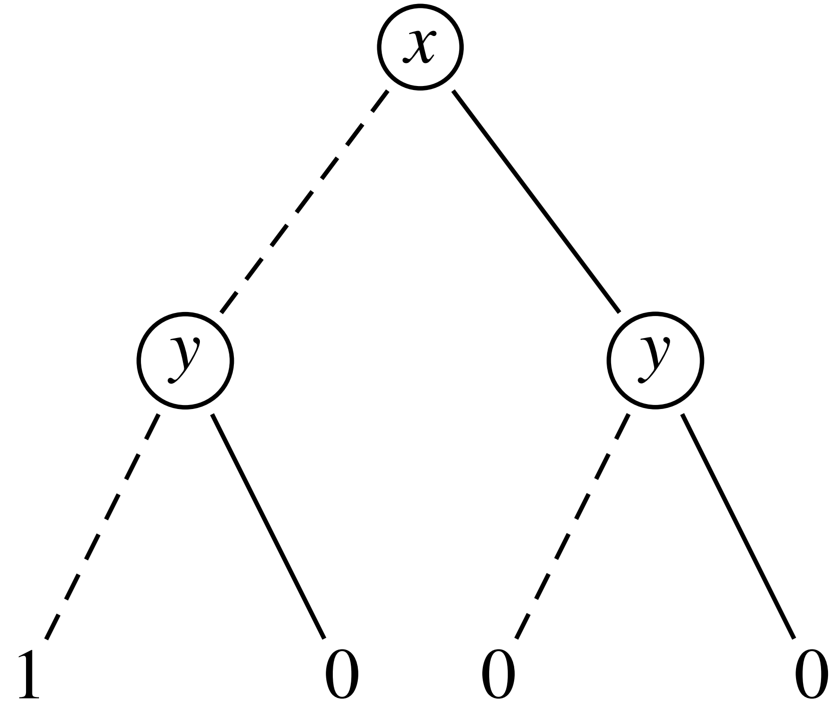
- Motivations.
- Ordered Binary Decision Diagrams (OBDD).
- OBDD's as Canonical Forms.
- Building OBDD's

Ordered Binary Decision Diagrams (OBDD)

- Ordered Binary Decision Diagrams (OBDD) are used to represent formulae in propositional logic.
- A simple version: Binary Decision Trees:
  - Non-Terminal nodes labeled with boolean variables/propositions;
  - Leaves (terminal nodes) are labeled with either 0 or 1;
  - Two kinds of lines: dashed and solid;
  - Paths leading to 1 represent models, while paths leading to 0 represent counter-models.

Binary Decision Trees: An Example

BDT representing the formula: $\varphi = \neg x \wedge \neg y$.



The assignment, $x = 0, y = 0$, makes true the formula.

Binary Decision Trees (BDT)

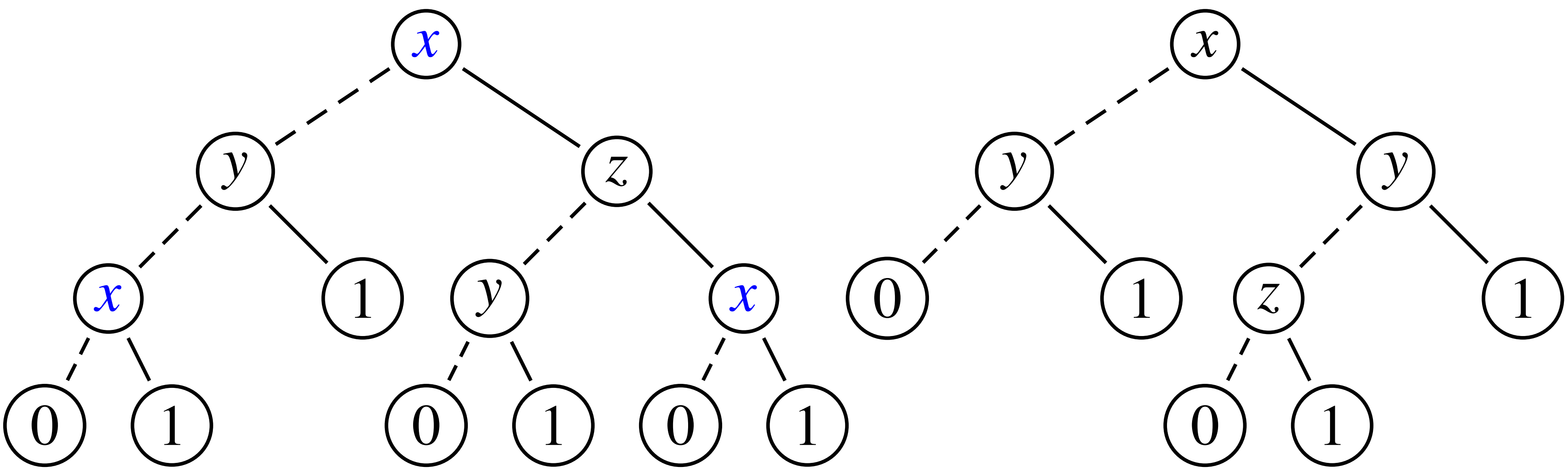
Let T be a BDT, then T determines a unique boolean formula in the following way:

- Fixed an assignment for the variables in T we start at the root and:
  - If the value of the variable in the current node is 1 we follow the solid line;
  - Otherwise, we follow the dashed line;
  - The truth value of the formula is given by the value of the leaf we reach.

Binary Decision Trees (Cont.)

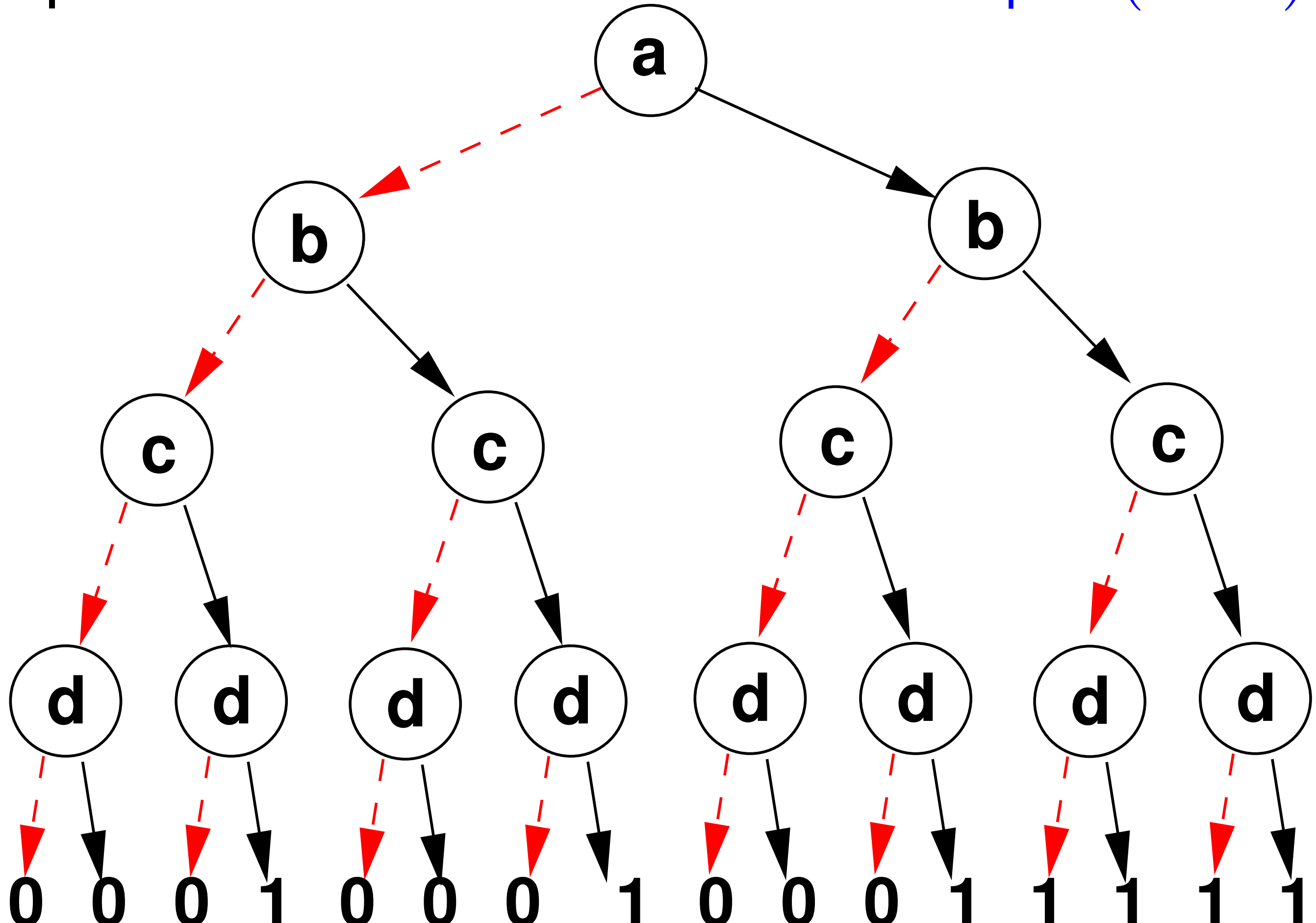
BDT's with multiple occurrences of a variable along a path are:

1. Rather inefficient (Redundant paths);
2. Difficult to check whether they represent the same formula (equivalence test). Example of two equivalent BDT's



Ordered Decision Trees

- Ordered Decision Tree (OBDT): from root to leaves variables are encountered always in the same order without repetitions along paths.
- Example: Ordered Decision tree for $\varphi = (a \wedge b) \vee (c \wedge d)$



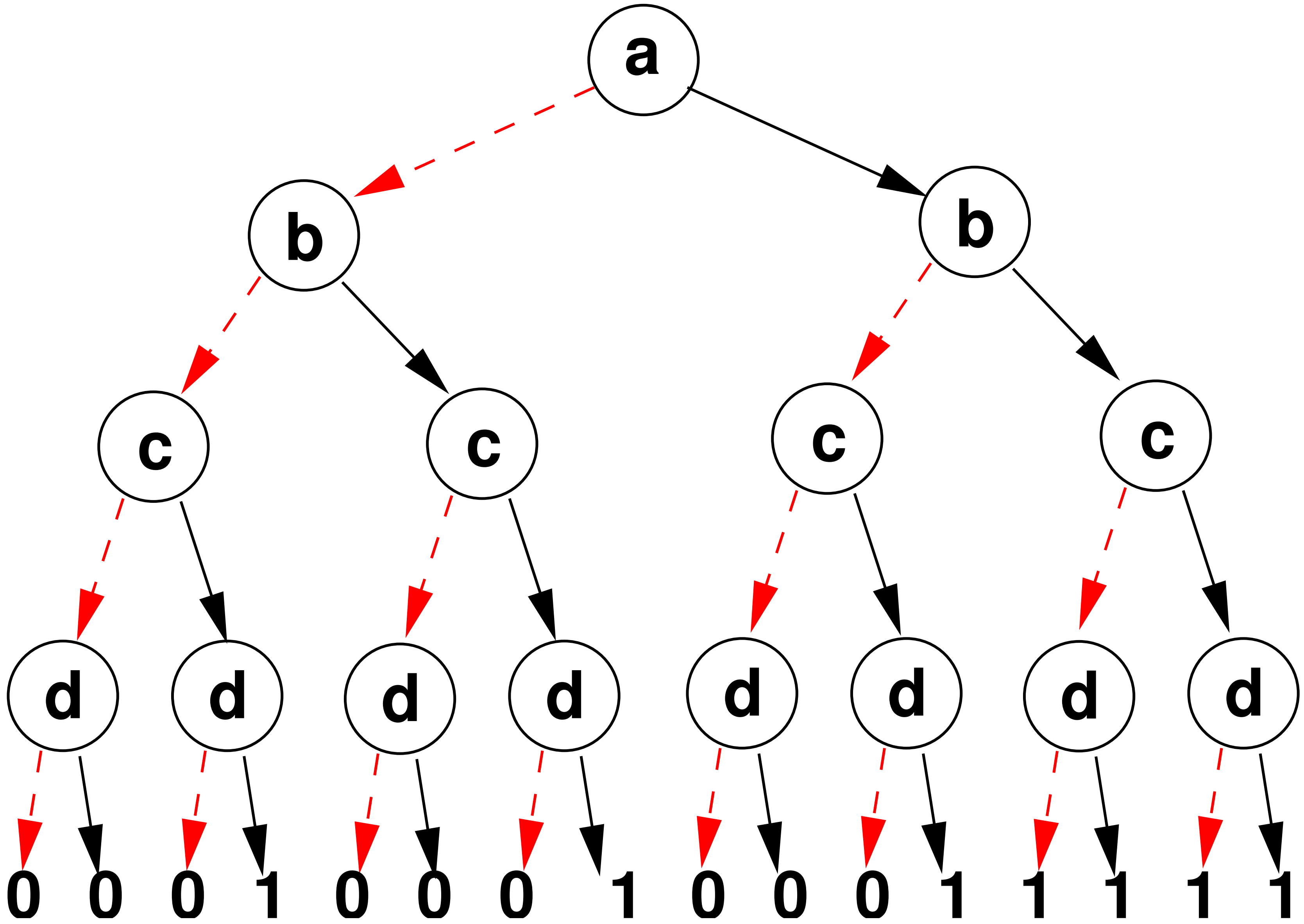
Summary

- Motivations.
- Ordered Binary Decision Diagrams (OBDD).
- OBDD's as Canonical Forms.
- Building OBDD's

Reducing the size of Ordered Decision Trees

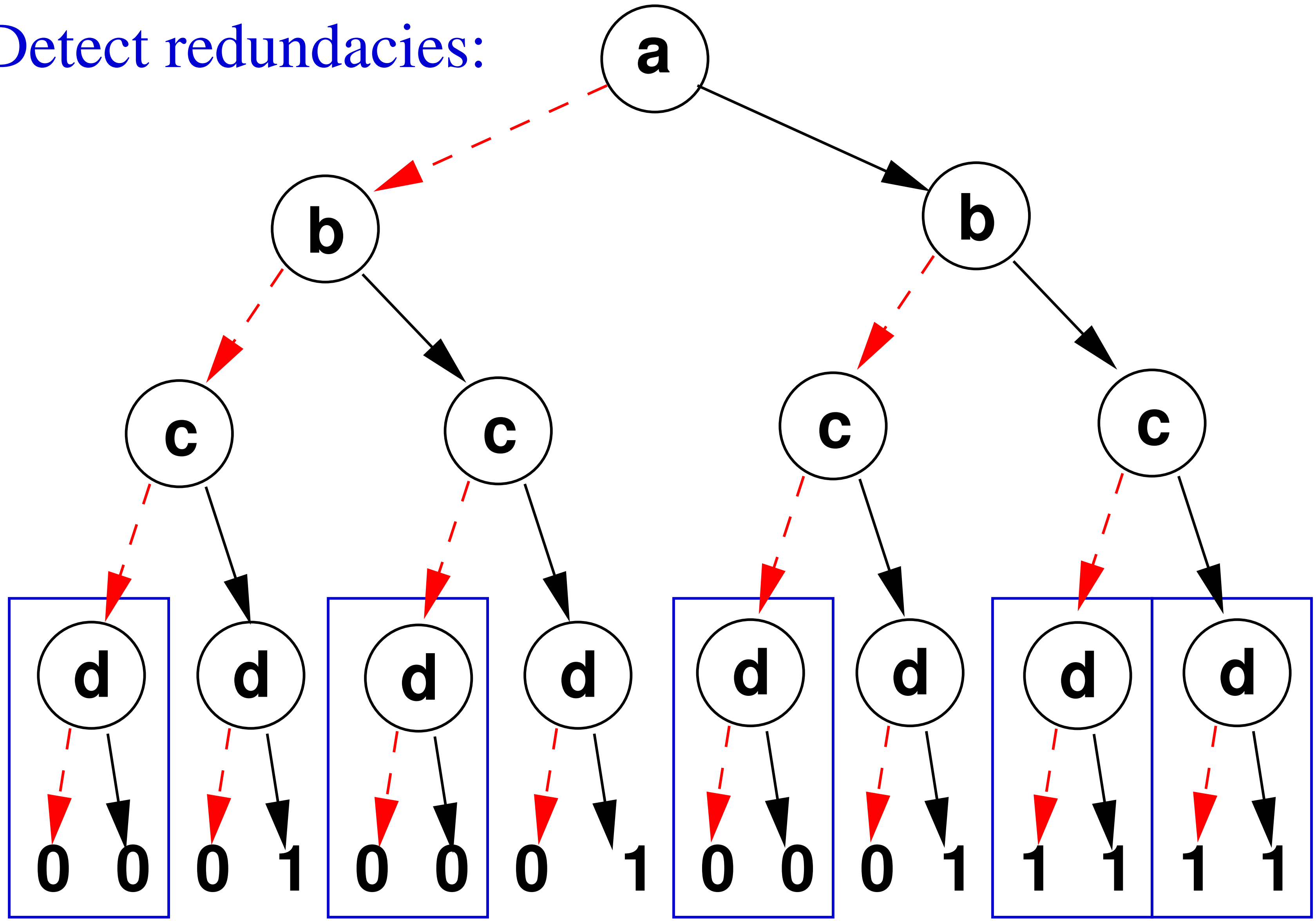
- OBDT's are still exponential in the number of variables: Given n variables the OBDT's will have $2^{n+1} - 1$ nodes!
- We can reduce the size of OBDT's by a recursive applications of the following reductions:
  - Remove Redundancies: Nodes with same left and right children can be eliminated;
  - Share Subnodes: Roots of structurally identical sub-trees can be collapsed.

Reduction: Example



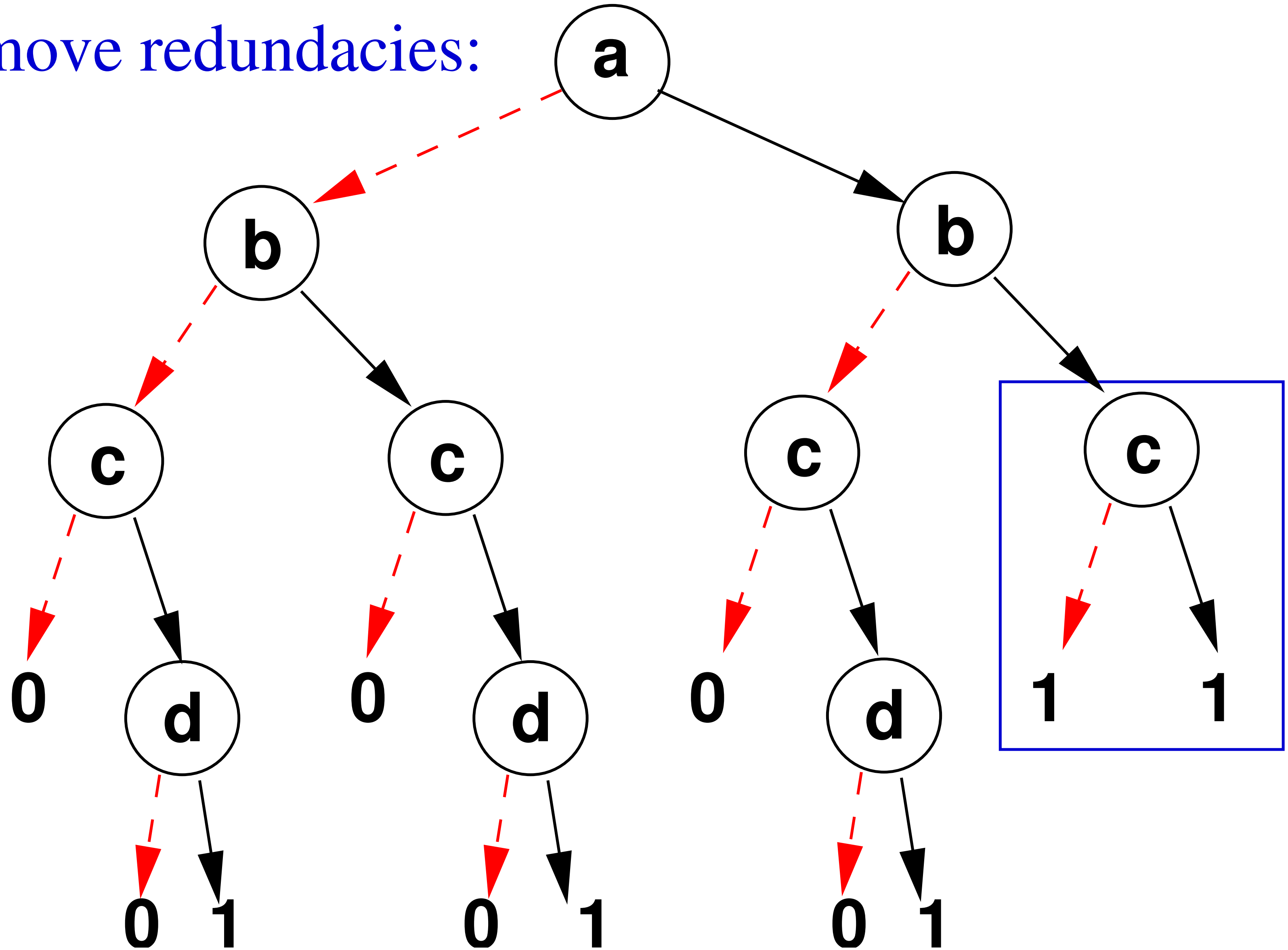
Reduction: Example (Cont.)

Detect redundacies:



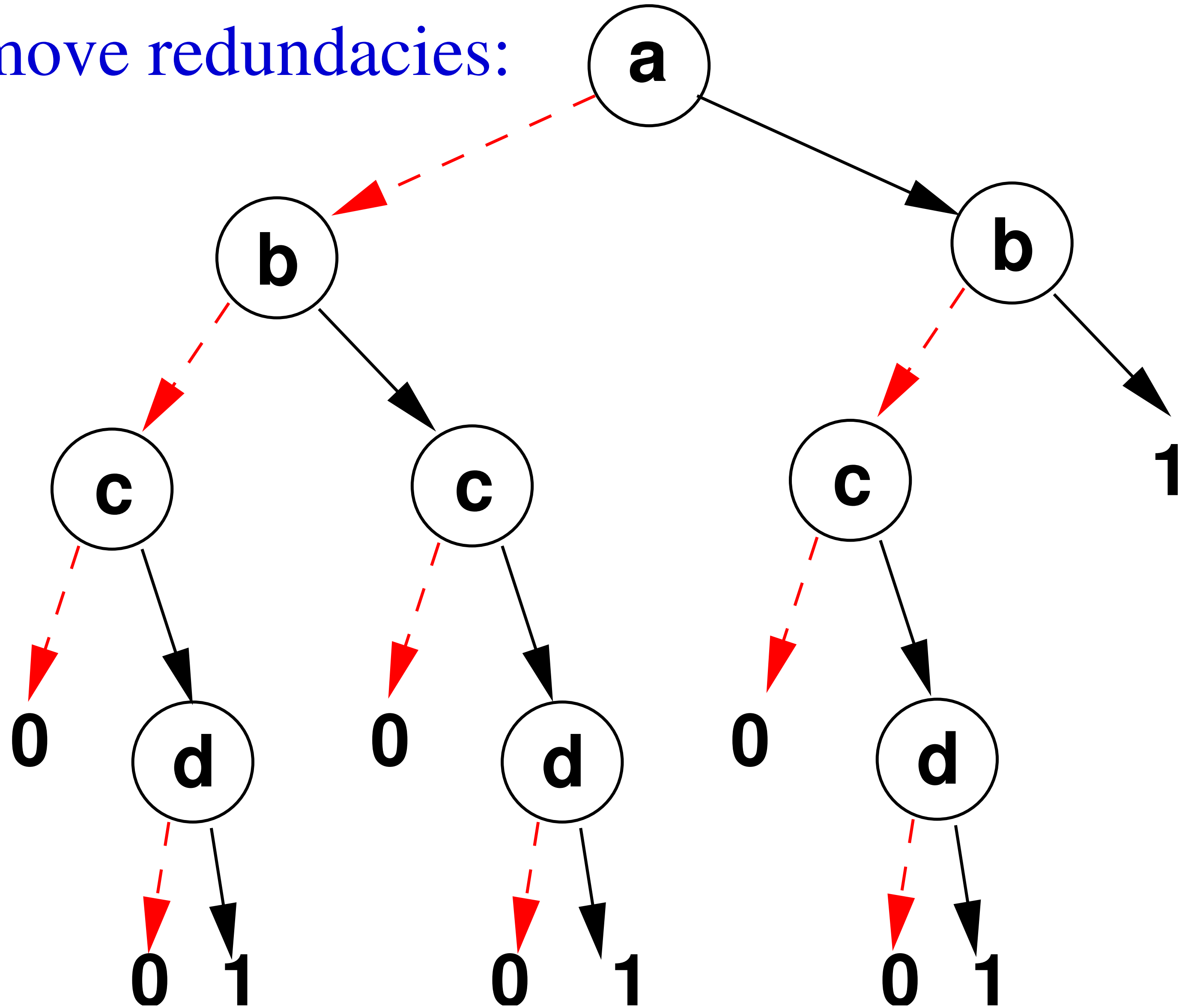
Reduction: Example (Cont.)

Remove redundacies:



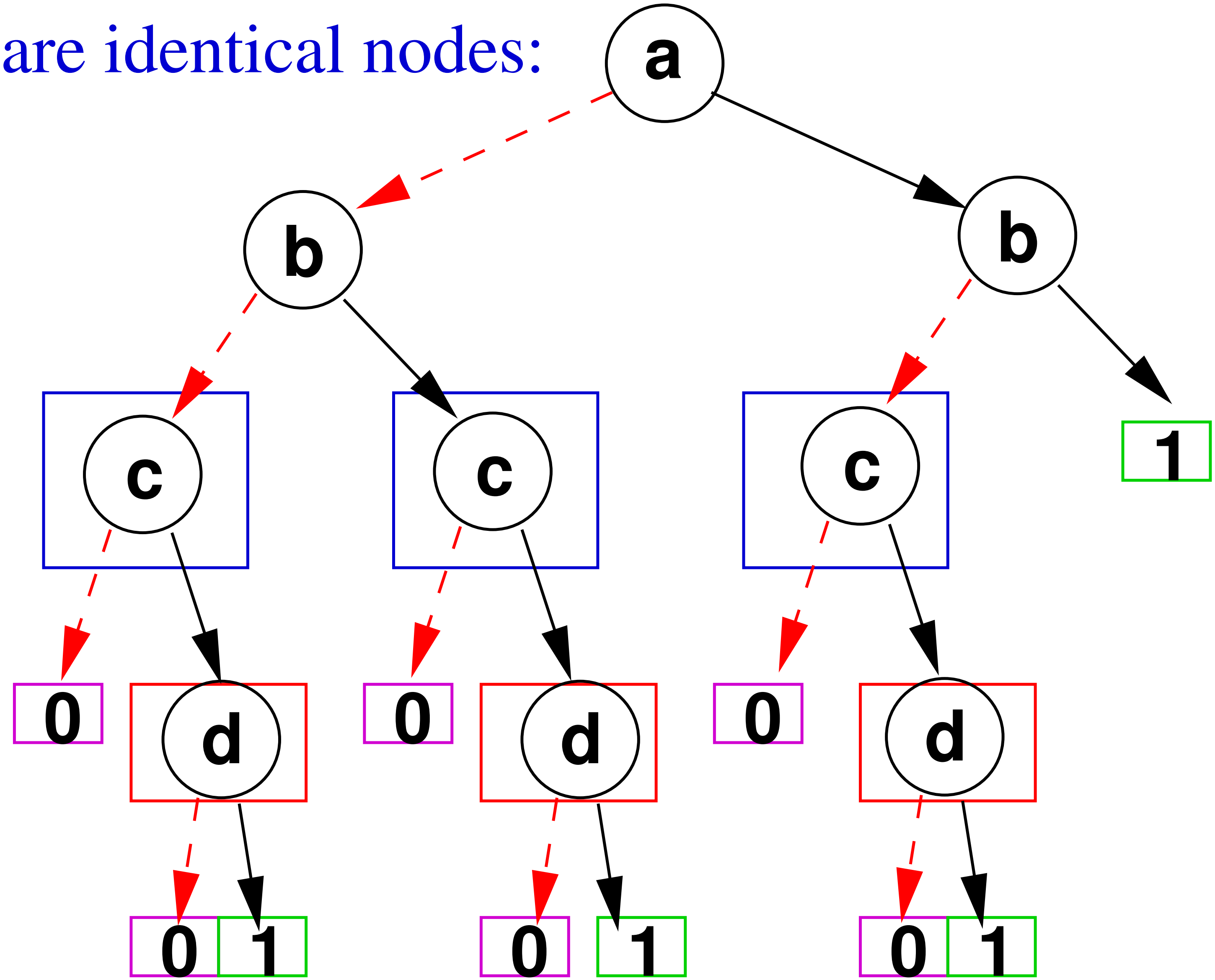
Reduction: Example (Cont.)

Remove redundacies:



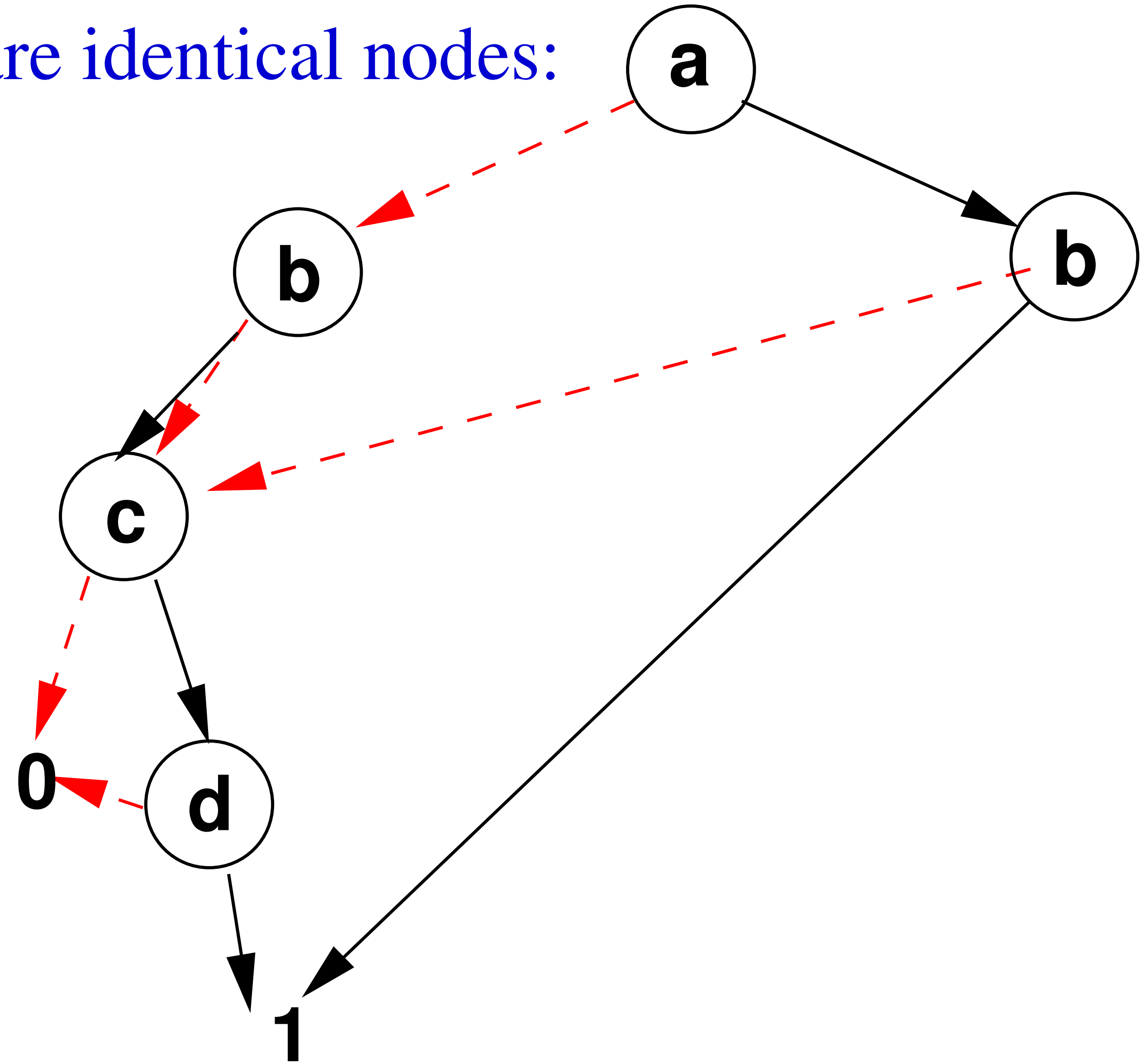
Reduction: Example (Cont.)

Share identical nodes:



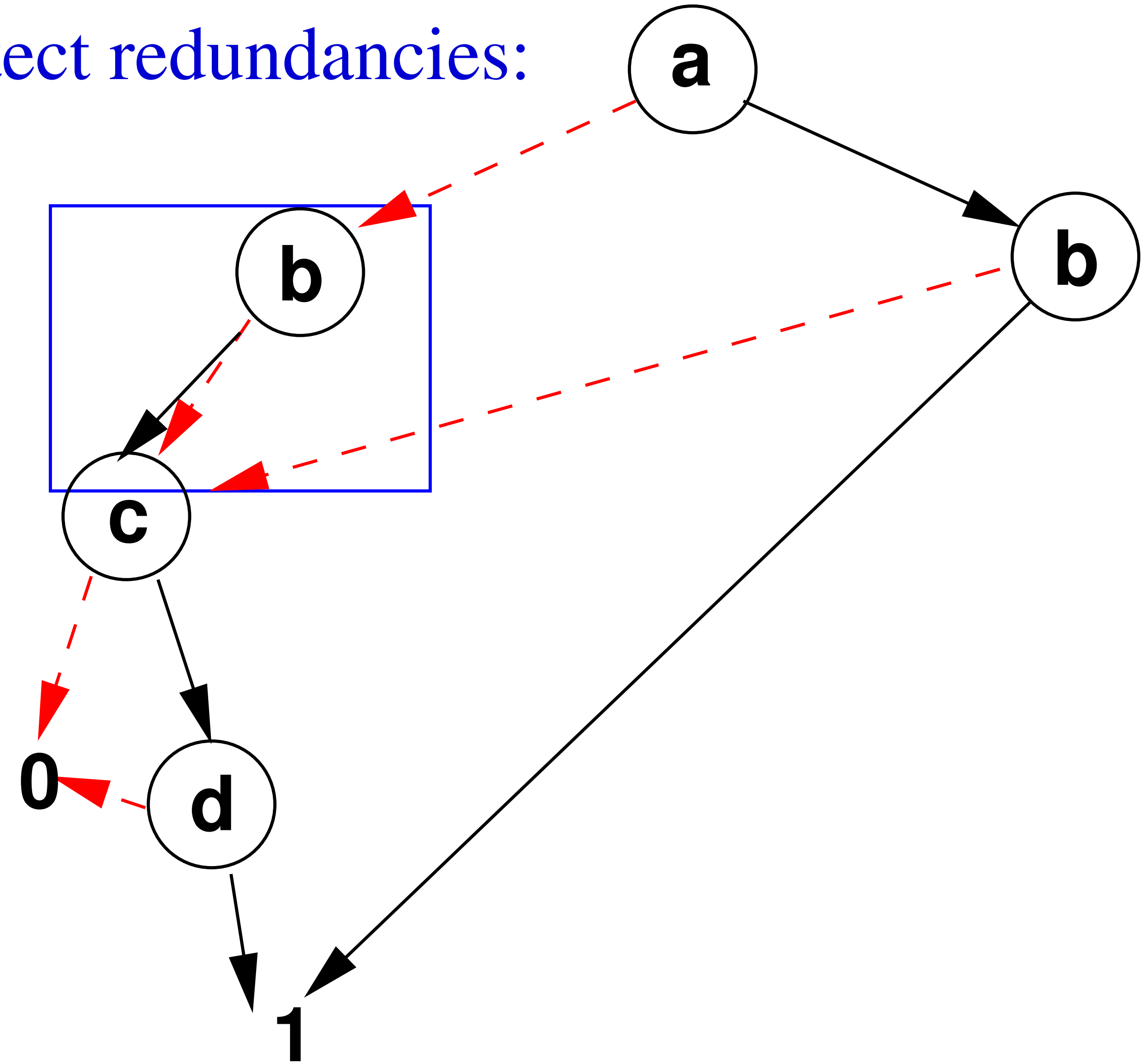
Reduction: Example (Cont.)

Share identical nodes:



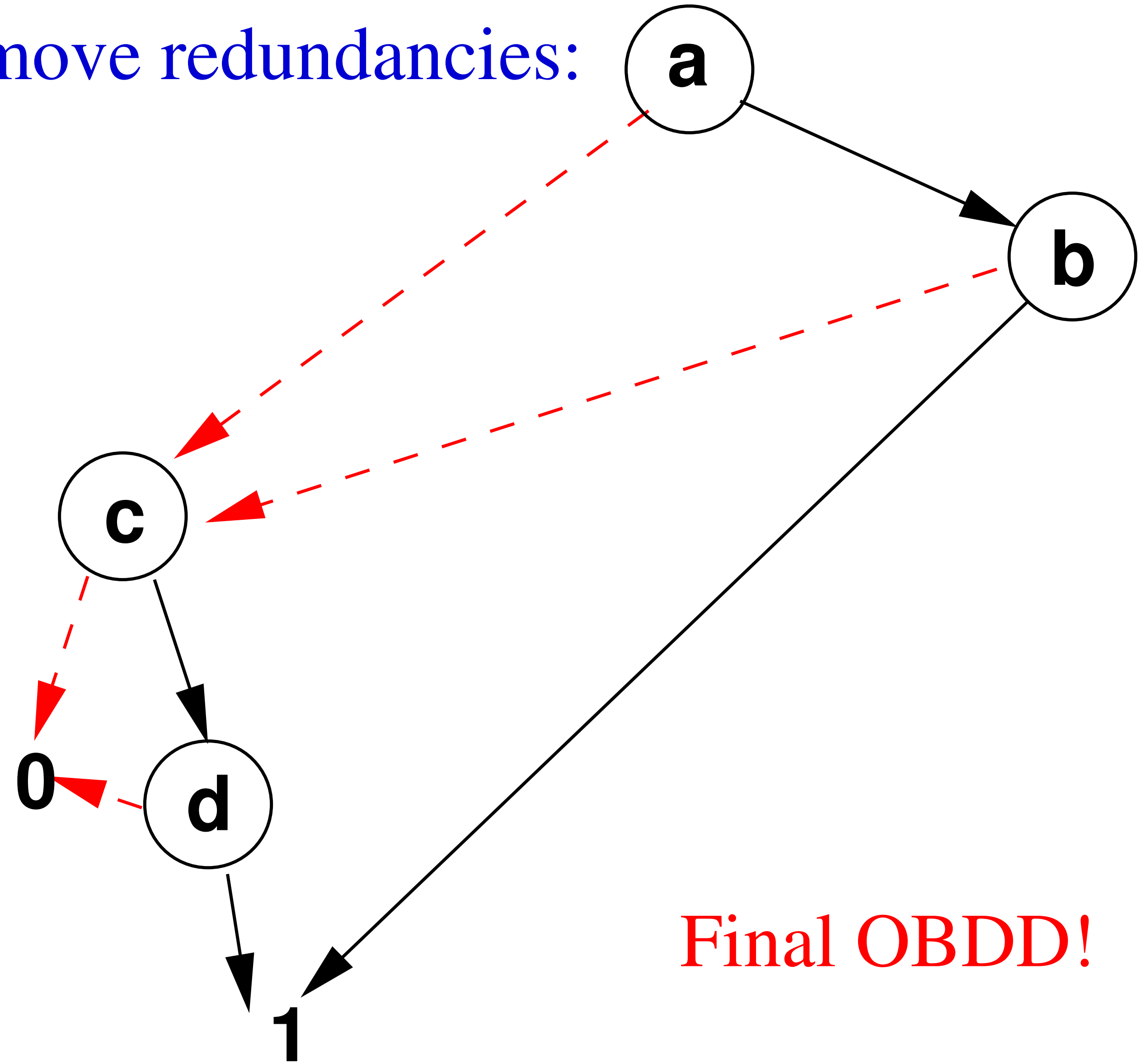
Reduction: Example (Cont.)

Detect redundancies:



Reduction: Example (Cont.)

Remove redundancies:



Final OBDD!

OBDD's as Canonical Forms

- **Definition.** Given two OBDD's, B_φ, B_ψ, they have a compatible variable ordering if there are no variables x, y such that $x < y$ in B_φ while $y < x$ in B_ψ.
- **Theorem.** A Reduced OBDD is a Canonical Form of a boolean formula: Once a variable ordering is established (i.e., OBDD's have compatible variable ordering), equivalent formulas are represented by the same OBDD:

$$\varphi_1 \Leftrightarrow \varphi_2 \text{ iff } OBDD(\varphi_1) \equiv OBDD(\varphi_2)$$

Importance of OBDD's

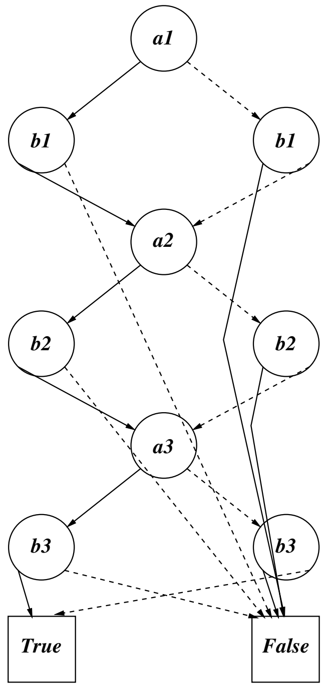
Canonical forms for OBDD's allow us to perform in an efficient way the following tests:

- Equivalence check is simple:
  We test whether the reduced and order compatible OBDD's have identical structure.
  Validity check requires constant time!
  $\varphi \Leftrightarrow \top$ iff the reduced OBDD $B_\varphi \equiv B_\top$
  (un)satisfiability check requires constant time!
  $\varphi \Leftrightarrow \bot$ iff the reduced OBDD $B_\varphi \equiv B_\bot$
- The set of the paths from the root to 1 represent all the models of the formula;
- The set of the paths from the root to 0 represent all the counter-models of the formula.

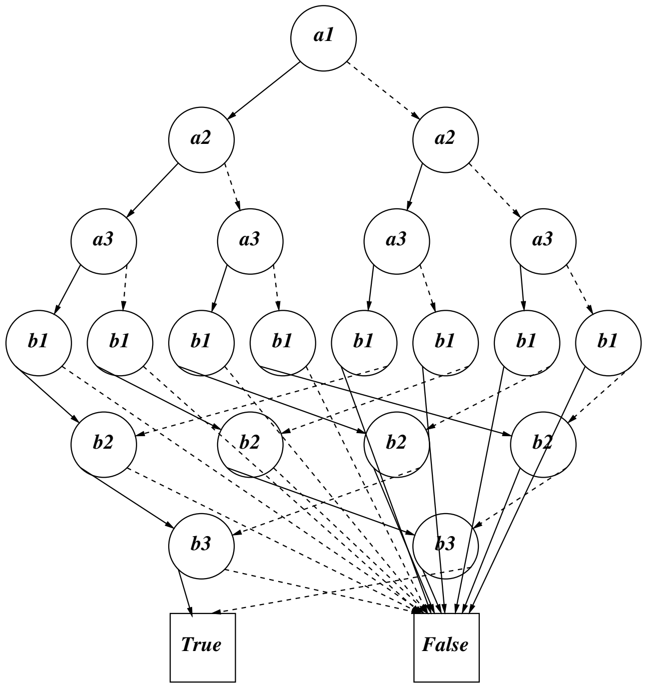
Importance of Variable Ordering

Changing the ordering of variables may increase the size of OBDD's. Example, two OBDD's for the formula:

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



Linear size



Exponential size

Summary

- Motivations.
- Ordered Binary Decision Diagrams (OBDD).
- OBDD’s as Canonical Forms.
- Building OBDD’s.

The REDUCE Algorithm

Notation. Given a non-terminal node, n, then $\texttt{lo}(\texttt{n})$ denotes the node pointed via the dashed line, while $\texttt{hi}(\texttt{n})$ denotes the node pointed by the solid line.

Given an OBDD, REDUCE proceeds bottom-up assigning an integer label, $\texttt{id}(\texttt{n})$, to each node:

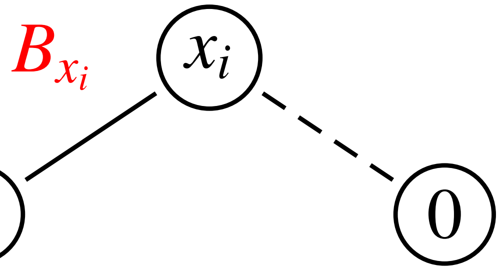
1. Assign label 0 to all 0-terminals and label 1 to all 1-terminals. Given now a non-terminal node for x_i, say n, then:
2. If $\texttt{id}(\texttt{lo}(\texttt{n})) = \texttt{id}(\texttt{hi}(\texttt{n}))$, then $\texttt{id}(\texttt{n}) = \texttt{id}(\texttt{lo}(\texttt{n}))$;
3. If there is another node for x_i, say m, such that $\texttt{id}(\texttt{lo}(\texttt{n})) = \texttt{id}(\texttt{lo}(\texttt{m}))$ and $\texttt{id}(\texttt{hi}(\texttt{n})) = \texttt{id}(\texttt{hi}(\texttt{m}))$, then, $\texttt{id}(\texttt{n}) = \texttt{id}(\texttt{m})$;
4. Otherwise we set $\texttt{id}(\texttt{n})$ to the next unused integer.

The Reduce Algorithm (Cont)

- REDUCE **Final Step:** Collapsing nodes with the same label and redirecting edges accordingly with the node collapsing.
- **Example:** See Figure 6.14 from the book.

Recursive structure of OBDD's

- Given a formula φ and a variable ordering $X = \{x_1, x_2, \ldots, x_n\}$, the algorithm to build OBDD's from formulas, $\mathtt{OBDD}(\varphi, \mathtt{X})$, operates recursively:
  1. If $\varphi = \top$, then, $\mathtt{OBDD}(\top, \mathtt{X}) = B_\top = 1$;
  2. If $\varphi = \bot$, then, $\mathtt{OBDD}(\bot, \mathtt{X}) = B_\bot = 0$;
  3. If $\varphi = x_i$, then, $\mathtt{OBDD}(\mathtt{x_i}, \mathtt{X}) =$



  4. If $\varphi = \neg\varphi_1$, then, $\mathtt{OBDD}(\neg\varphi_1, \mathtt{X})$ is obtained by negating the terminal nodes of $\mathtt{OBDD}(\varphi_1, \mathtt{X})$;
  5. If $\varphi = \varphi_1 \ \mathtt{op} \ \varphi_2$ ($\mathtt{op}$ a binary boolean operator), then, $\mathtt{OBDD}(\varphi_1 \ \mathtt{op} \ \varphi_2, \mathtt{X}) = \mathtt{apply}(\mathtt{op}, \mathtt{OBDD}(\varphi_1, \mathtt{X}), \mathtt{OBDD}(\varphi_2, \mathtt{X}))$.

The algorithm Apply

- Given two OBDD's, B_φ, B_ψ, the call $\mathtt{apply}(\mathtt{op}, \mathtt{B}_\varphi, \mathtt{B}_\psi)$ computes the reduced OBDD of the formula $\varphi \ \mathtt{op}\ \psi$.
- The algorithms operates recursively on the structure of the two OBDD's:
  1. Let x be the variable highest in the ordering which occurs in B_φ or B_ψ, then
  2. Split the problem in two sub-problems: one for x being true and the other for x being false and solve recursively;
  3. At the leaves, apply the boolean operation directly.

The algorithm APPLY (Cont.)

Definition. Let φ be a formula and x a variable. We denote by $\varphi[0/x]$ ($\varphi[1/x]$) the formula obtained by replacing all occurrences of x in φ by 0 (1).

This allow us to split boolean formulas in simpler ones.

Lemma [Shannon Expansion]. Let φ be a formula and x a variable, then:

$$\varphi \equiv (x \land \varphi[1/x]) \lor (\neg x \land \varphi[0/x])$$

The function APPLY is based on the Shannon Expansion:

$$\varphi \,\mathrm{op}\, \psi \equiv (x \land (\varphi[1/x] \,\mathrm{op}\, \psi[1/x])) \lor (\neg x \land (\varphi[0/x] \,\mathrm{op}\, \psi[0/x]))$$

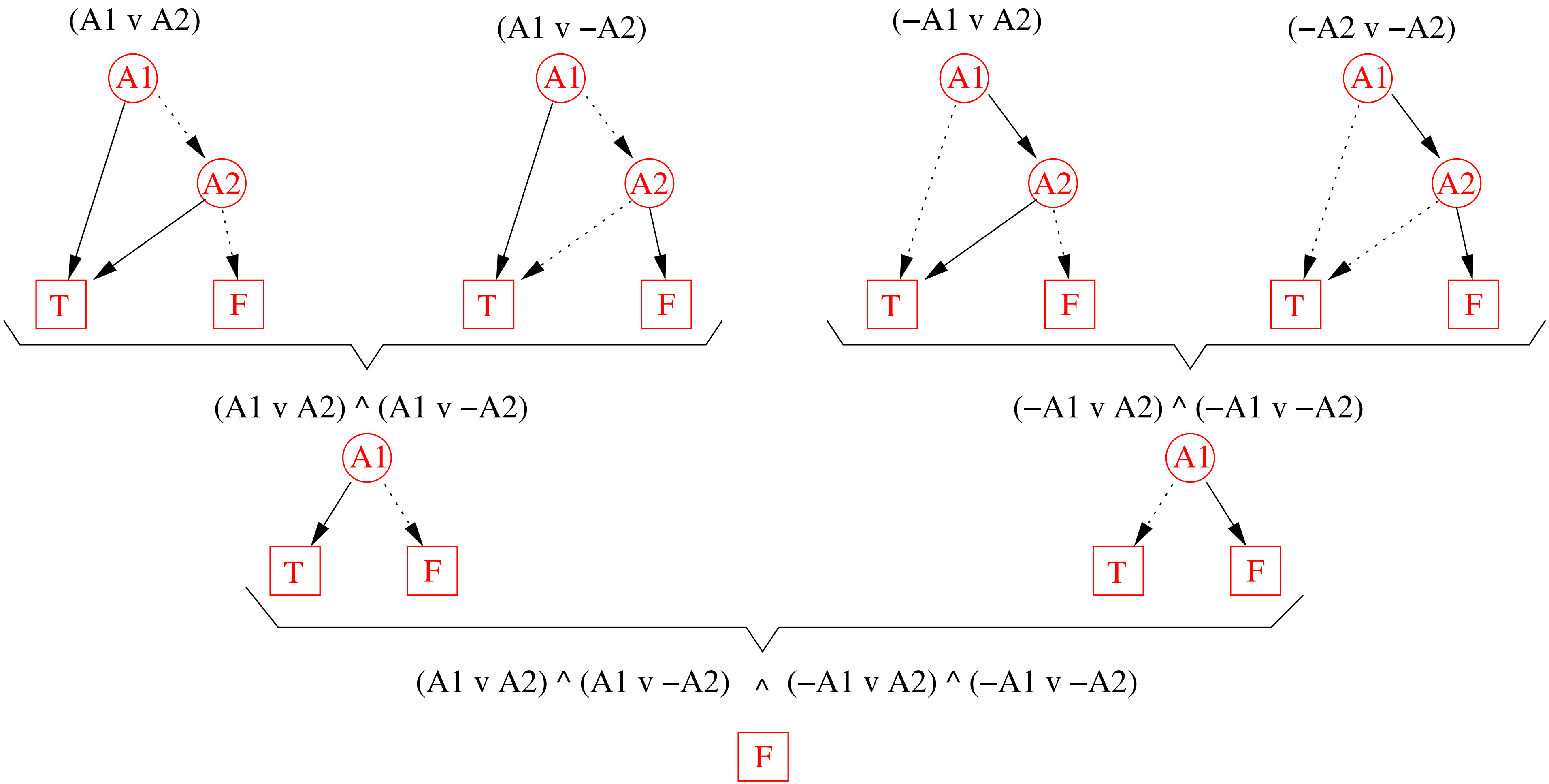
The algorithm APPLY (Cont.)

$\texttt{Apply}(\texttt{op}, \texttt{B}_\varphi, \texttt{B}_\psi)$ proceeds from the roots downward. Let r_φ, r_ψ the roots of B_φ, B_ψ respectively:

1. If both r_φ, r_ψ are terminal nodes, then, $\texttt{Apply}(\texttt{op}, \texttt{B}_\varphi, \texttt{B}_\psi) = \texttt{B}_{(\texttt{r}_\varphi\ \texttt{op}\ \texttt{r}_\psi)}$;
2. If both roots are x_i-nodes, then create an x_i-node with a dashed line to $\texttt{Apply}(\texttt{op}, \texttt{B}_{\texttt{lo}(\texttt{r}_\varphi)}, \texttt{B}_{\texttt{lo}(\texttt{r}_\psi)})$ and a solid line to $\texttt{Apply}(\texttt{op}, \texttt{B}_{\texttt{hi}(\texttt{r}_\varphi)}, \texttt{B}_{\texttt{hi}(\texttt{r}_\psi)})$;
3. If r_φ is an x_i-node, but r_ψ is a terminal node or an x_j-node with $j > i$ (i.e., $\psi[0/x_i] \equiv \psi[1/x_i] \equiv \psi$), then create an x_i-node with dashed line to $\texttt{Apply}(\texttt{op}, \texttt{B}_{\texttt{lo}(\texttt{r}_\varphi)}, \texttt{B}_\psi)$ and solid line to $\texttt{Apply}(\texttt{op}, \texttt{B}_{\texttt{hi}(\texttt{r}_\varphi)}, \texttt{B}_\psi)$;
4. If r_ψ is an x_i-node, but r_φ is a terminal node or an x_j-node with $j > i$, is handled as above.

OBBD Incremental Building: An Example

$$\varphi = (A_1 \vee A_2) \wedge (A_1 \vee \neg A_2) \wedge (\neg A_1 \vee A_2) \wedge (\neg A_1 \vee \neg A_2)$$



Boolean Quantification

- Quantifying over boolean variables is a crucial operation to compute Preimages (i.e., the next-time operator).
- If x is a boolean variable, then

$$\begin{aligned} \exists x.\varphi &\equiv \varphi[0/x] \lor \varphi[1/x] \\ \forall x.\varphi &\equiv \varphi[0/x] \land \varphi[1/x] \end{aligned}$$

- Let $W = \{w_1, \ldots, w_n)$. Multi-variable quantification:

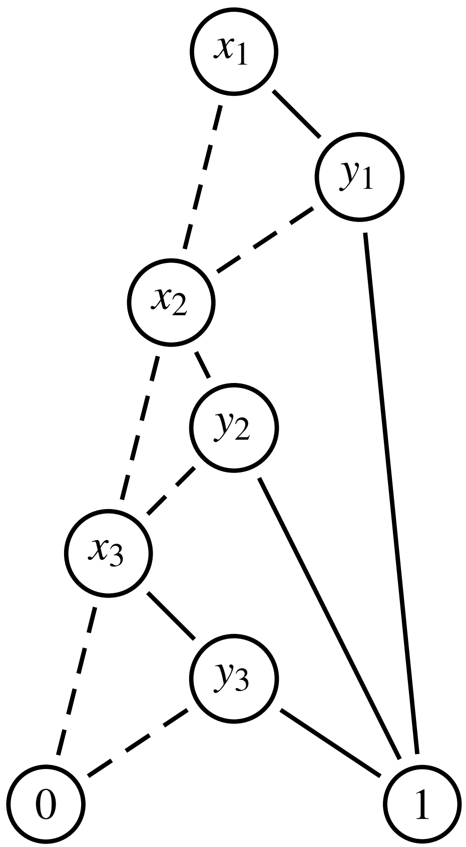
$$\exists W.\varphi \equiv \exists (w_1, \ldots, w_n).\varphi \equiv \exists w_1 \ldots \exists w_n.\varphi$$

The RESTRICT Algorithm

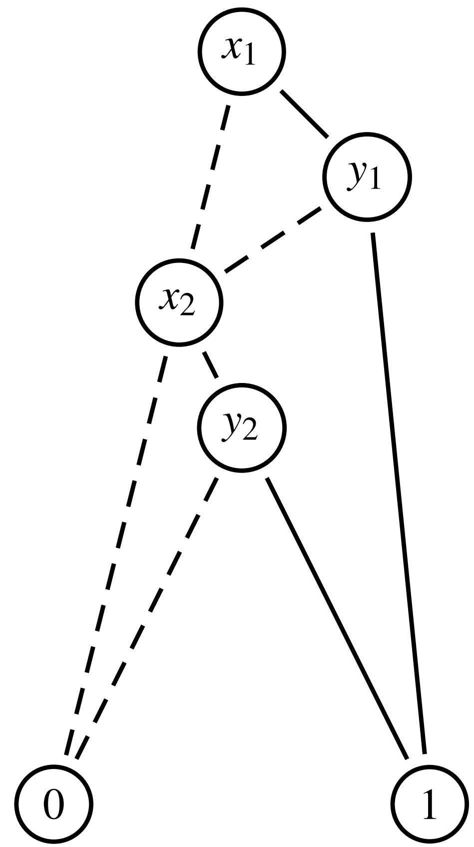
- To compute the OBDD for $\exists x.\varphi$ we need to compute the OBDD for both $\varphi[0/x]$ and $\varphi[1/x]$.
- $B_{\varphi[0/x]} = \text{RESTRICT}(0, x, B_\varphi)$.
  For each node n labeled with x, then:
  1. Incoming edges are redirected to $lo(n)$;
  2. n is removed.
- $B_{\varphi[1/x]} = \text{RESTRICT}(1, x, B_\varphi)$.
  As above, only redirect incoming edges to $hi(n)$.

Boolean Quantification (Cont.)

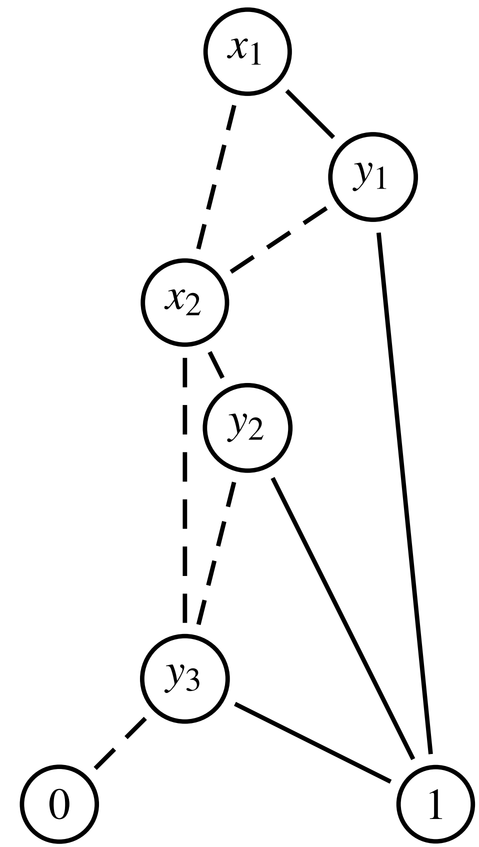
$$B_{\exists x.\varphi} = \text{APPLY}(\vee, \text{RESTRICT}(0, x, B_\varphi), \text{RESTRICT}(1, x, B_\varphi))$$



B_φ



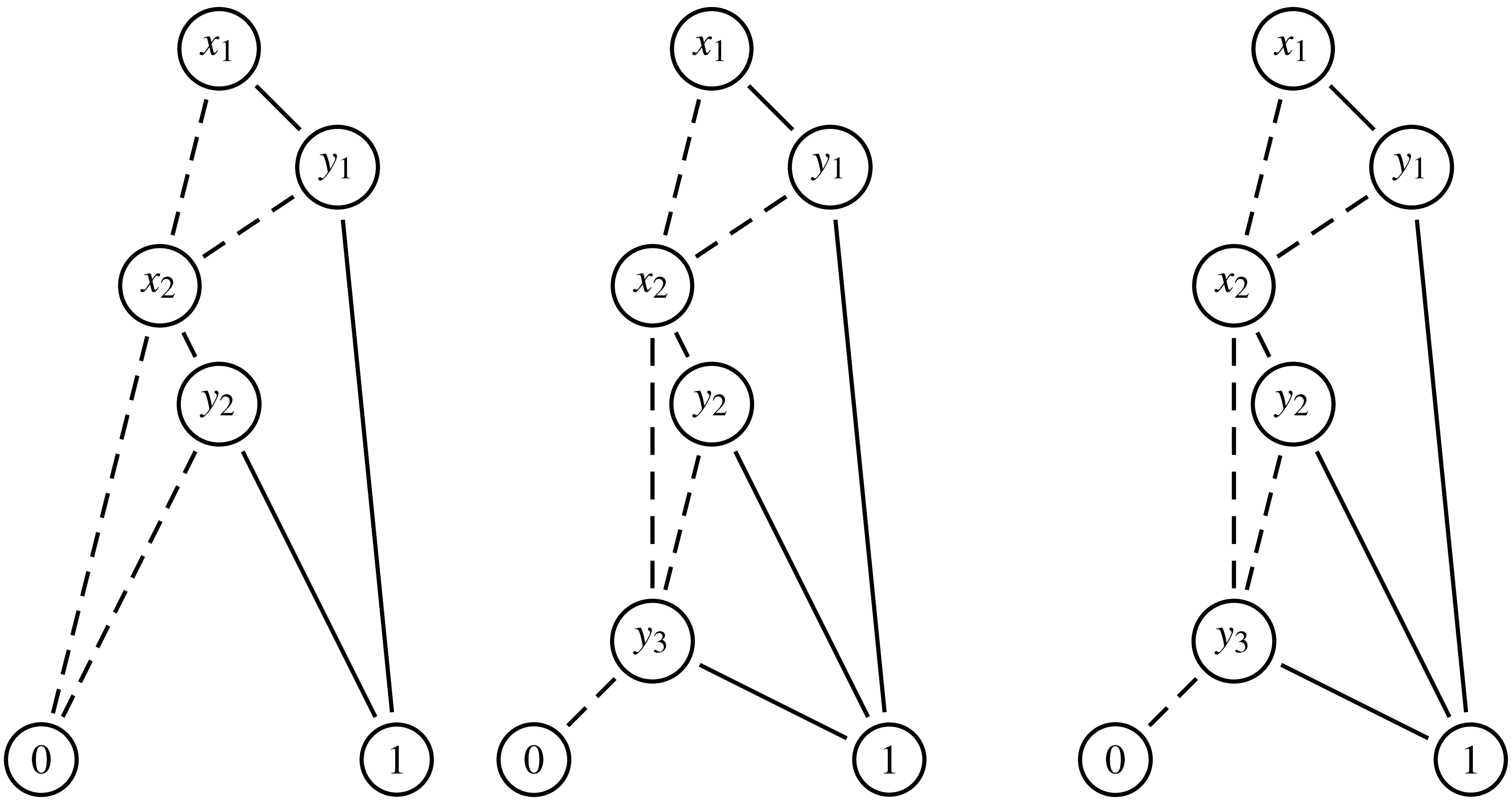
$\text{RESTRICT}(0, x_3, B_\varphi)$



$\text{RESTRICT}(1, x_3, B_\varphi)$

Boolean Quantification (Cont.)

$$B_{\exists x.\varphi} = \text{APPLY}(\vee, \text{RESTRICT}(0, x, B_\varphi), \text{RESTRICT}(1, x, B_\varphi))$$



$\text{RESTRICT}(0, x_3, B_\varphi)$ $\text{RESTRICT}(1, x_3, B_\varphi)$ $\exists x_3 . B_\varphi$

Time Complexity

Algorithm	Time-Complexity
REDUCE(B)	$O(\|B\| \times log\ \|B\|)$
APPLY($\mathrm{op}, \mathrm{B}_\varphi, \mathrm{B}_\psi$)	$O(\|B_\varphi\| \times \|B_\psi\|)$

N.B. The above complexity results depend from the size of the input OBDD's:

- The size of OBDD's may grow exponentially wrt. the number of variables in worst-case.
- Example: there exist no polynomial-size OBDD representing the electronic circuit of a bitwise multiplier.

OBDD – Summary

- Require setting a variable ordering a priori (critical!)
- Normal representation of a boolean formula.
- Once built, logical operations (satisfiability, validity, equivalence) immediate.
- Represents all models and counter-models of the formula.
- Require exponential space in worst-case.

Summary of Lecture VI

- Motivations.
- Ordered Binary Decision Diagrams (OBDD).
- OBDD's as Canonical Forms.
- Building OBDD's.