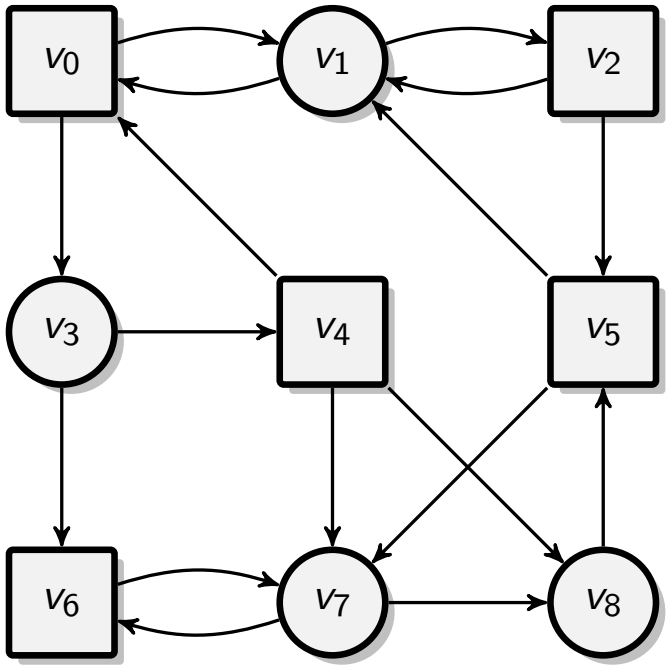
Reactive Synthesis

Lecture 3

Swen Jacobs and Martin Zimmermann
(Saarland University)

Recap

A *game* $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ consists of an arena $\mathcal{A} = (V, V_0, V_1, E)$ and a winning condition $\mathrm{Win} \subseteq V^\omega$.



$$\mathrm{Win} = \{\rho_0\rho_1\rho_2 \cdots \in V^\omega \mid \exists v \in V \text{ such that } \rho_n \neq v \text{ for all } n\}$$

Recap

Let $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ be a game with vertex set V.

- $\mathcal{G}$ is a safety game if there is a set $S \subseteq V$ such that

$$\mathrm{Win} = \textsc{Safety}(S) := \{\rho \in V^\omega \mid \mathrm{Occ}(\rho) \subseteq S\}.$$

Recap

Let $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ be a game with vertex set V.

- $\mathcal{G}$ is a safety game if there is a set $S \subseteq V$ such that

$$\mathrm{Win} = \textsc{Safety}(S) := \{\rho \in V^\omega \mid \mathrm{Occ}(\rho) \subseteq S\}.$$

- $\mathcal{G}$ is a reachability game if there is a set $R \subseteq V$ such that

$$\mathrm{Win} = \textsc{Reach}(R) := \{\rho \in V^\omega \mid \mathrm{Occ}(\rho) \cap R \neq \emptyset\}.$$

Recap

Let $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ be a game with vertex set V.

- $\mathcal{G}$ is a safety game if there is a set $S \subseteq V$ such that

$$\mathrm{Win} = \textsc{Safety}(S) := \{\rho \in V^\omega \mid \mathrm{Occ}(\rho) \subseteq S\}.$$

- $\mathcal{G}$ is a reachability game if there is a set $R \subseteq V$ such that

$$\mathrm{Win} = \textsc{Reach}(R) := \{\rho \in V^\omega \mid \mathrm{Occ}(\rho) \cap R \neq \emptyset\}.$$

Remark

Safety and reachability games and are dual:

	Player 0	Player 1
Safety	Avoid $V \setminus S$	Reach $V \setminus S$
Reachability	Reach R	Avoid R

Recap

Solving Reachability and Safety Games:

- these games are determined with positional strategies
- can be solved with attractor construction,
  using (for reachability)

$$\begin{aligned}\mathrm{CPre}_0(R) = \{v \in V_0 \mid v' \in R \text{ for some successor } v' \text{ of } v\} \cup \\ \{v \in V_1 \mid v' \in R \text{ for all successors } v' \text{ of } v\}.\end{aligned}$$

  or (for safety)

$$\begin{aligned}\mathrm{CPre}_1(R) = \{v \in V_1 \mid v' \in R \text{ for some successor } v' \text{ of } v\} \cup \\ \{v \in V_0 \mid v' \in R \text{ for all successors } v' \text{ of } v\}.\end{aligned}$$

- efficient algorithm with running time bounded by $|E|$

Plan for Today

- Basic Games
- Algorithms & Data Structures

- Advanced Games
- Temporal Logic Synthesis

Plan for Today

- Basic Games
- Algorithms & Data Structures
  - Symbolic Representations of Games
  - Binary Decision Diagrams (BDDs)
- Advanced Games
- Temporal Logic Synthesis

Scalability of Game Solving

We want to use reactive synthesis to solve really large games, where the arena is defined by the memory components (latches) of a circuit or the state variables of a program.

Scalability of Game Solving

We want to use reactive synthesis to solve really large games, where the arena is defined by the memory components (latches) of a circuit or the state variables of a program.

Problem: a circuit with 32 latches (or a program with 4 variables with each 8 bit) can contain up to 2^{32} states, and

$$2^{32} \cdot 2^{32} = 18.446.744.073.709.551.616$$

edges. If linear algorithm visits each edge once for 1 ns, traversal would need $>$ 500 years!

Scalability of Game Solving

We want to use reactive synthesis to solve really large games, where the arena is defined by the memory components (latches) of a circuit or the state variables of a program.

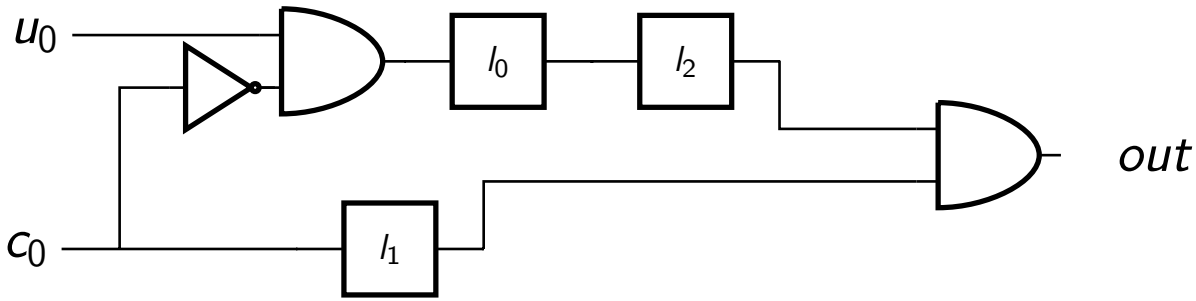
Problem: a circuit with 32 latches (or a program with 4 variables with each 8 bit) can contain up to 2^{32} states, and

$$2^{32} \cdot 2^{32} = 18.446.744.073.709.551.616$$

edges. If linear algorithm visits each edge once for 1 ns, traversal would need > 500 years!

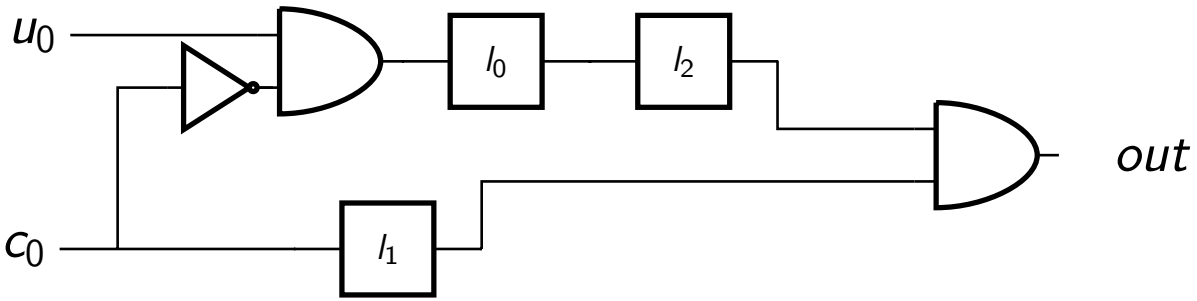
Solution: use *symbolic* representations of sets of states, such as formulas or certain graphs, which can be much smaller than the represented state space. However, this also requires algorithms that can work on these symbolic representations.

Games from Circuits



Goal: for uncontrollable u_0, control c_0 (based on values of u_0, l_0, l_1, l_2) such that the value of *out* will never be 1.

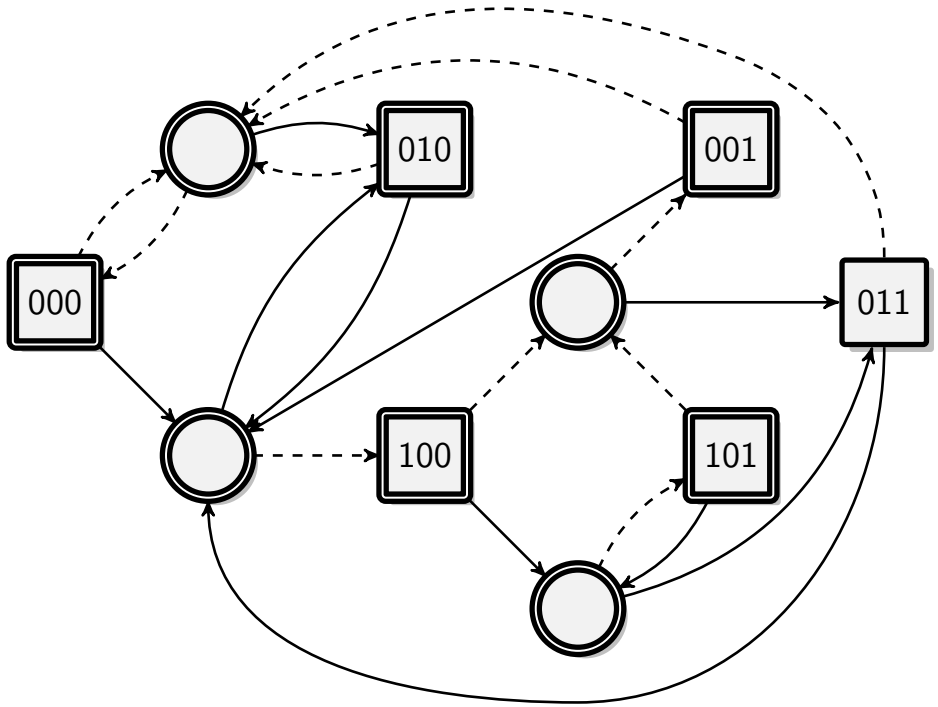
Games from Circuits



Goal: for uncontrollable u_0, control c_0 (based on values of u_0, l_0, l_1, l_2) such that the value of *out* will never be 1.

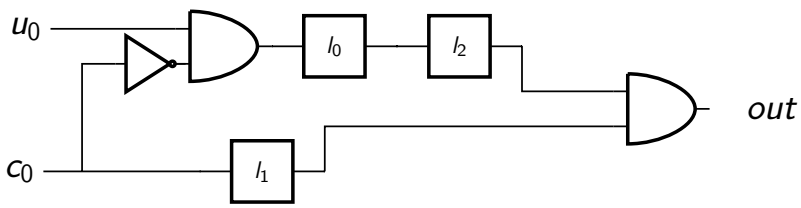
This corresponds to the game on the right:

- state 010 means $l_0 = 0, l_1 = 1, l_2 = 0$
- every player always has two choices, corresponding to setting their input to 0 or 1
- 011 is the only unsafe state



Symbolic Representation of Games: Example

The game that is induced by the example circuit can be represented **much more succinctly**:



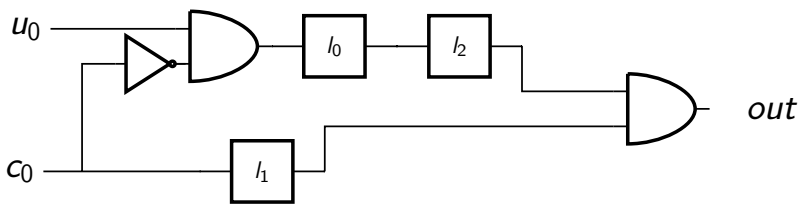
Set of state variables:

Transition relation:

Unsafe states:

Symbolic Representation of Games: Example

The game that is induced by the example circuit can be represented **much more succinctly**:



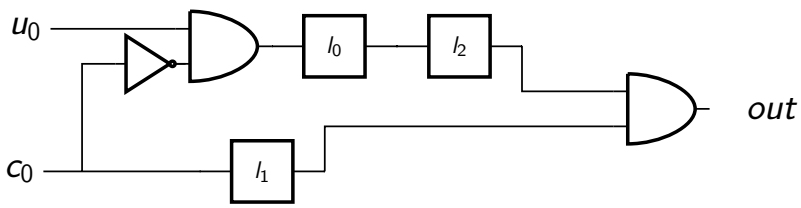
Set of state variables: $L = \{l_0, l_1, l_2\}$

Transition relation:

Unsafe states:

Symbolic Representation of Games: Example

The game that is induced by the example circuit can be represented **much more succinctly**:



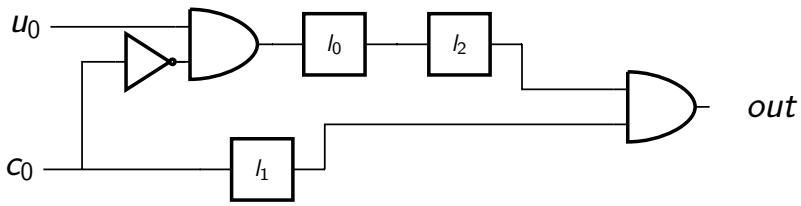
Set of state variables: $L = \{l_0, l_1, l_2\}$

Transition relation: $(l_0' \Leftrightarrow u_0 \wedge \bar{c_0}) \wedge (l_1' \Leftrightarrow c_0) \wedge (l_2' \Leftrightarrow l_0)$
(where x' for a variable x means its value in the next time step, and we contract the coices of both players into one transition)

Unsafe states:

Symbolic Representation of Games: Example

The game that is induced by the example circuit can be represented **much more succinctly**:



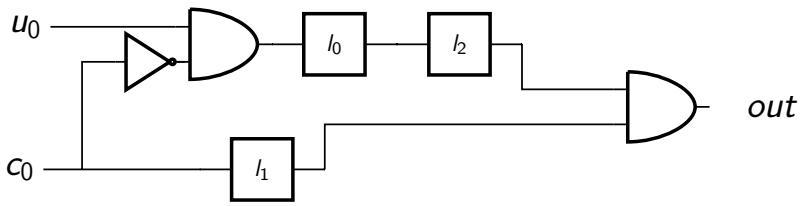
Set of state variables: $L = \{l_0, l_1, l_2\}$

Transition relation: $(l_0' \Leftrightarrow u_0 \wedge \bar{c_0}) \wedge (l_1' \Leftrightarrow c_0) \wedge (l_2' \Leftrightarrow l_0)$
(where x' for a variable x means its value in the next time step, and we contract the coices of both players into one transition)

Unsafe states: $l_1 \wedge l_2$

Symbolic Representation of Games: Example

The game that is induced by the example circuit can be represented **much more succinctly**:



Set of state variables: $L = \{l_0, l_1, l_2\}$

Transition relation: $(l_0' \Leftrightarrow u_0 \wedge \bar{c_0}) \wedge (l_1' \Leftrightarrow c_0) \wedge (l_2' \Leftrightarrow l_0)$
(where x' for a variable x means its value in the next time step, and we contract the coices of both players into one transition)

Unsafe states: $l_1 \wedge l_2$

Remark

Note that each formula F represents a set of states with arbitrary size. A formula over variables $x_1, \ldots, x_n$ also represents a **boolean function** *$f_F : \mathbb{B}^n \to \mathbb{B}$, the* **characteristic function** *of the set it represents.*

Symbolic Representation of Games: Definition

Definition

A symbolic representation for a safety game is given as a tuple $(L, X_u, X_c, T(L, X_u, X_c, L'), \mathit{Unsafe}(L))$, where

- L is a set of (boolean) state variables
- X_u is a set of (boolean) uncontrollable input variables
- X_c is a set of (boolean) controllable input variables
- $T(L, X_u, X_c, L')$ is the transition relation, given as a quantifier-free boolean formula over variables from L, X_u, X_c and L', which represents the values of variables from L after the transition
- $\mathit{Unsafe}(L)$ defines the unsafe states, as a quantifier-free boolean formula over state variables L

Remark

We sometimes emphasize the variables X occurring in a formula F by writing $F(X)$.

Quantified Boolean Formulas:

For a boolean formula F and variable x:

$$\forall x.F \Leftrightarrow (F[x := 1] \wedge F[x := 0])$$

$$\exists x.F \Leftrightarrow (F[x := 1] \vee F[x := 0])$$

For $X = \{x_1, \ldots, x_n\}$, we also write $\exists X.F$ instead of $\exists x_1 \ldots \exists x_n.F$.

Solving Games on Symbolic Representations

Quantified Boolean Formulas:

For a boolean formula F and variable x:

$$\forall x.F \Leftrightarrow (F[x := 1] \wedge F[x := 0])$$

$$\exists x.F \Leftrightarrow (F[x := 1] \vee F[x := 0])$$

For $X = \{x_1, \ldots, x_n\}$, we also write $\exists X.F$ instead of $\exists x_1 \ldots \exists x_n.F$.

Example

For a transition relation $T(L, X_u, X_c, L')$, the formula $\exists L, X_u, X_c.T(L, X_u, X_c, L')$ represents the set of all values of L' such that there exists a transition from some state (value of L) with some inputs (values of X_u and X_c).

Solving Games on Symbolic Representations

To solve a safety game in symbolic representation, we can compute the (Player 1) attractor of the unsafe states.

The controllable predecessor of Player 1 becomes

$$\mathrm{CPre}_1(F) = \exists X_u \forall X_c \exists L'.\ F(L') \wedge T(L, X_u, X_c, L')$$

Solving Games on Symbolic Representations

To solve a safety game in symbolic representation, we can compute the (Player 1) attractor of the unsafe states.

The controllable predecessor of Player 1 becomes

$$\mathrm{CPre}_1(F) = \exists X_u \forall X_c \exists L'.\ F(L') \wedge T(L, X_u, X_c, L')$$

and the attractor is defined as (with $F = \mathit{Unsafe}$)

- $\mathrm{Attr}_1^0(F) = F$,
- $\mathrm{Attr}_1^{n+1}(F) = \mathrm{Attr}_1^n(F) \vee \mathrm{CPre}_1(\mathrm{Attr}_1^n(F))$, and
- $\mathrm{Attr}_1(F) = \bigvee_{n \in \mathbb{N}} \mathrm{Attr}_1^n(F)$.

Solving Games on Symbolic Representations

To solve a safety game in symbolic representation, we can compute the (Player 1) attractor of the unsafe states.

The controllable predecessor of Player 1 becomes

$$\mathrm{CPre}_1(F) = \exists X_u \forall X_c \exists L'.\ F(L') \wedge T(L, X_u, X_c, L')$$

and the attractor is defined as (with $F = \mathit{Unsafe}$)

- $\mathrm{Attr}_1^0(F) = F$,
- $\mathrm{Attr}_1^{n+1}(F) = \mathrm{Attr}_1^n(F) \vee \mathrm{CPre}_1(\mathrm{Attr}_1^n(F))$, and
- $\mathrm{Attr}_1(F) = \bigvee_{n \in \mathbb{N}} \mathrm{Attr}_1^n(F)$.

Problem: how can we efficiently compute this?
Need algorithms and data structures that allow:

- efficient combination of existing formulas with of $\exists, \forall, \wedge, \vee$
- efficient equivalence check of formulas (to determine attractor computation has finished),
- compact representation during these computations.

Representations for Boolean Formulas

How hard is it to apply operations and obtain an equivalent formula in the given representation? How hard is equivalence checking? Is this representation usually compact?

Representation	$\forall, \wedge$	$\exists, \vee$	equivalence	compact?
arbitrary formula	easy	easy	**hard**	often

Representations for Boolean Formulas

How hard is it to apply operations and obtain an equivalent formula in the given representation? How hard is equivalence checking? Is this representation usually compact?

Representation	$\forall, \wedge$	$\exists, \vee$	equivalence	compact?
arbitrary formula	easy	easy	**hard**	often
DNF	**hard**	easy	easy	sometimes

Disjunctive Normal Form (DNF): a disjunction of conjunctions of literals, i.e., $\bigvee(L_1 \wedge \ldots \wedge L_n)$

Representations for Boolean Formulas

How hard is it to apply operations and obtain an equivalent formula in the given representation? How hard is equivalence checking? Is this representation usually compact?

Representation	$\forall, \wedge$	$\exists, \vee$	equivalence	compact?
arbitrary formula	easy	easy	**hard**	often
DNF	**hard**	easy	easy	sometimes
CNF	easy	medium	medium	sometimes

Disjunctive Normal Form (DNF): a disjunction of conjunctions of literals, i.e., $\bigvee(L_1 \wedge \ldots \wedge L_n)$

Conjunctive Normal Form (CNF): a conjunction of disjunctions of literals, i.e., $\bigwedge(L_1 \vee \ldots \vee L_n)$

Representations for Boolean Formulas

How hard is it to apply operations and obtain an equivalent formula in the given representation? How hard is equivalence checking? Is this representation usually compact?

Representation	$\forall, \wedge$	$\exists, \vee$	equivalence	compact?
arbitrary formula	easy	easy	**hard**	often
DNF	**hard**	easy	easy	sometimes
CNF	easy	medium	medium	sometimes
BDD	medium	medium	easy	often

Disjunctive Normal Form (DNF): a disjunction of conjunctions of literals, i.e., $\bigvee(L_1 \wedge \ldots \wedge L_n)$

Conjunctive Normal Form (CNF): a conjunction of disjunctions of literals, i.e., $\bigwedge(L_1 \vee \ldots \vee L_n)$

BDD (Binary Decision Diagram): a graph-based representation of quantifier-free boolean formulas

Binary Decision Trees

Definition (BDT)

A **binary decision tree (BDT)** for a set of variables X with a total order $<$ is a tree where

- every non-leaf node v is labeled with a variable $var(v) \in X$,
- every node has exactly two successors, $high(v)$ and $low(v)$,
- on every path through the tree, all variables appear and are ordered according to $<$, and
- every leaf node is labeled with a value $val(v) \in \mathbb{B}$.

Binary Decision Trees

Definition (BDT)

A **binary decision tree (BDT)** for a set of variables X with a total order $<$ is a tree where

- every non-leaf node v is labeled with a variable $var(v) \in X$,
- every node has exactly two successors, $high(v)$ and $low(v)$,
- on every path through the tree, all variables appear and are ordered according to $<$, and
- every leaf node is labeled with a value $val(v) \in \mathbb{B}$.

Definition (Semantics of BDTs)

Every node v in a BDT represents a boolean formula $F(v)$:

- if v is a leaf node, then $F(v)$ is $val(v)$, and
- if v is not a leaf node, then $F(v)$ is $(var(v) \wedge F(high(v))) \vee (\neg var(v) \wedge F(low(v)))$.

Obtaining a BDT from a boolean formula

Let F be a boolean formula over variables X, and $<$ a total order on X. Let $max(X)$ denote the maximal variable in X wrt. $<$.

Then $BDT(F, X, <)$ is defined recursively in the following way:

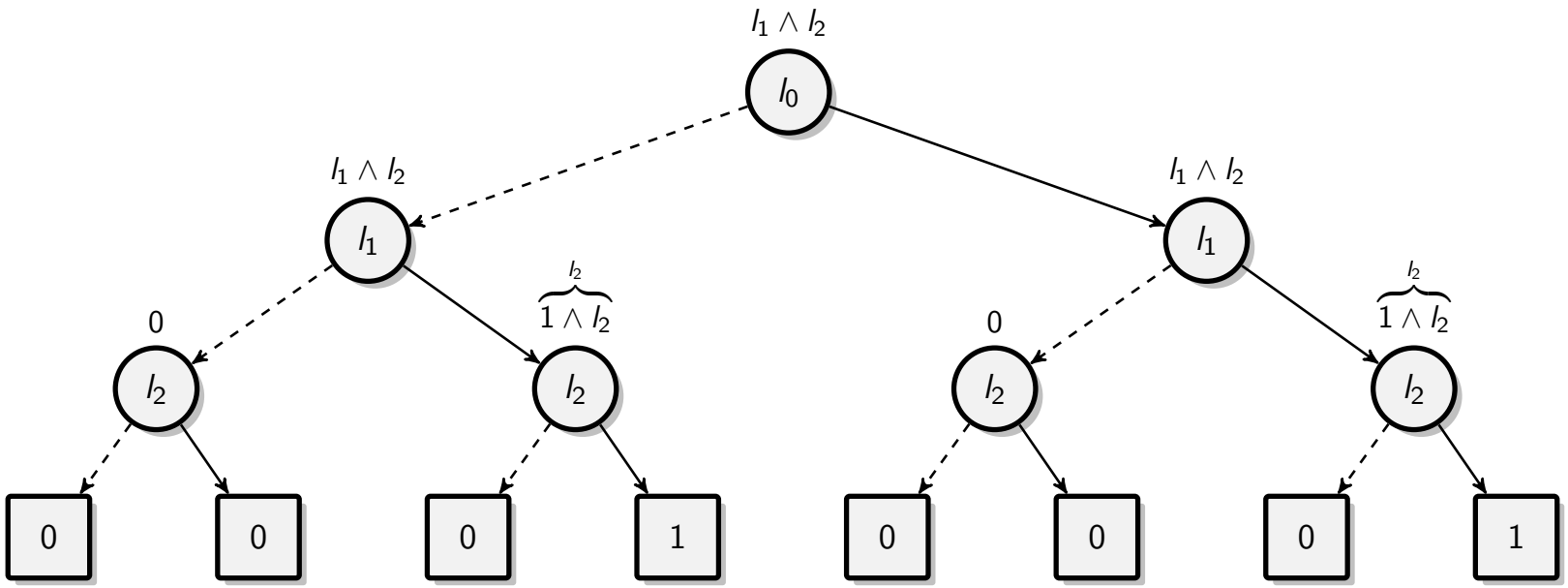
1. if X is not empty, create a node v with $var(v) = max(X)$, let $low(v) = BDT(F[var(v) := 0], X \setminus var(v), <)$ and $high(v) = BDT(F[var(v) := 1], X \setminus var(v), <)$
2. if X is empty, create a leaf node with the value of F.

Example: BDT from Formula

Example: BDT for $(l_1 \wedge l_2, \{l_0, l_1, l_2\}, l_0 > l_1 > l_2\})$

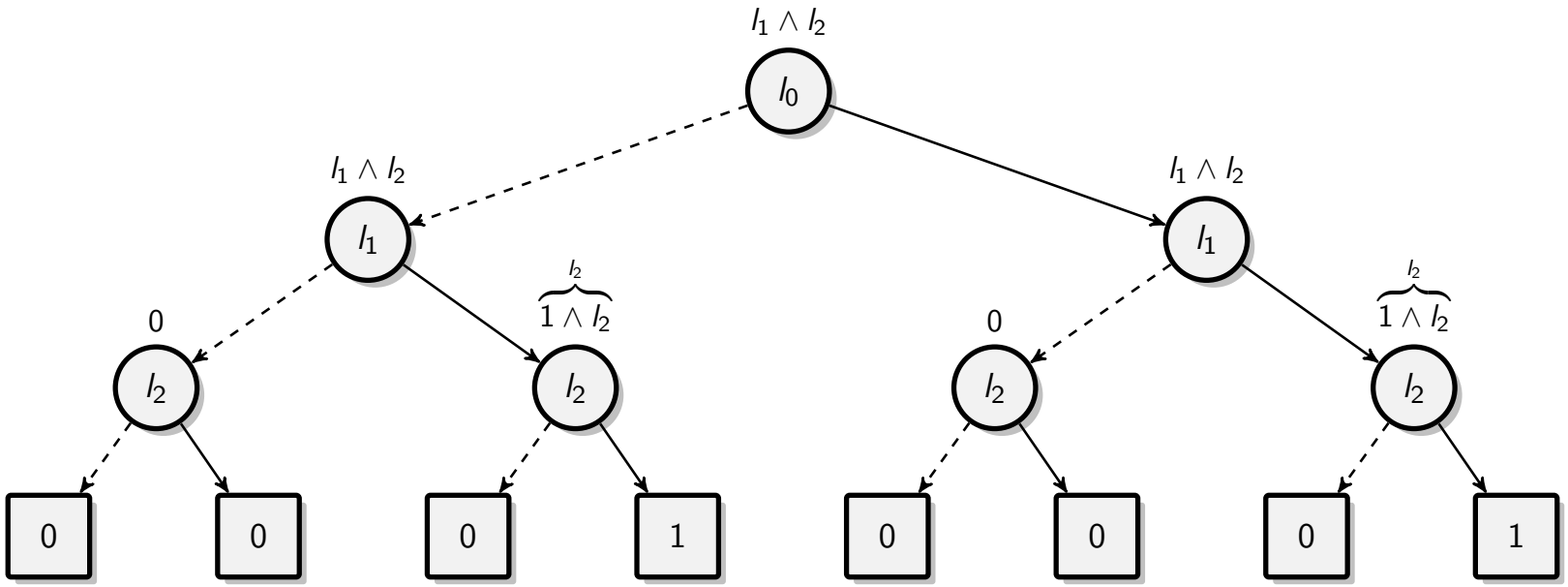
Example: BDT from Formula

Example: BDT for $(l_1 \wedge l_2, \{l_0, l_1, l_2\}, l_0 > l_1 > l_2\})$



Example: BDT from Formula

Example: BDT for $(l_1 \wedge l_2, \{l_0, l_1, l_2\}, l_0 > l_1 > l_2\})$



Observation: BDTs contain redundant information, such as

- multiple leaves with same value
- isomorphic subtrees.

Binary Decision Diagrams

Idea: allow more general graphs than trees, and remove redundant information from BDTs.

Binary Decision Diagrams

Idea: allow more general graphs than trees, and remove redundant information from BDTs.

Definition

A **Binary Decision Diagram (BDD)** over a set of variables X is a directed, acyclic graph $G(V, E)$ with exactly one root $v_r \in V$ and the following properties:

- every node in V is either terminal or non-terminal
- each terminal node is labeled with a value from $\{0, 1\}$
- each non-terminal node is labeled with a variable $x \in X$ and has exactly two outgoing edges, denoted by $high(v) \in V$ and $low(v) \in V$, respectively.

Binary Decision Diagrams

Idea: allow more general graphs than trees, and remove redundant information from BDTs.

Definition

A **Binary Decision Diagram (BDD)** over a set of variables X is a directed, acyclic graph $G(V, E)$ with exactly one root $v_r \in V$ and the following properties:

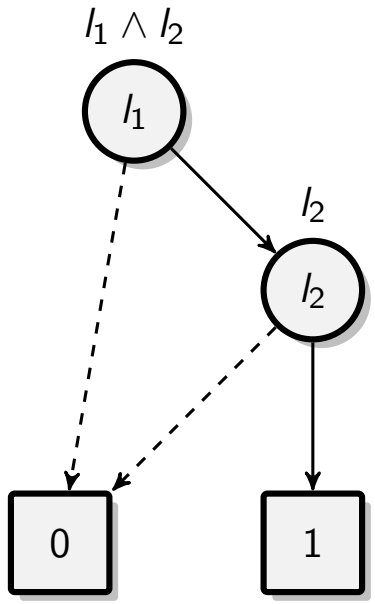
- every node in V is either terminal or non-terminal
- each terminal node is labeled with a value from $\{0, 1\}$
- each non-terminal node is labeled with a variable $x \in X$ and has exactly two outgoing edges, denoted by $\mathit{high}(v) \in V$ and $\mathit{low}(v) \in V$, respectively.

Remark

1. *The semantics of BDDs is the same as for BDTs.*
2. *Every BDT is a BDD.*

Example: BDD

A BDD for $(l_1 \wedge l_2, \{l_0, l_1, l_2\})$



Observation: much less redundant information than the BDT.

Drawbacks of BDDs

"General" BDDs do not provide a **canonical representation**, i.e., two different BDDs can represent the same formula.

Example

The BDD from the slide before, and the BDT (which is a BDD) for the same formula.

Drawbacks of BDDs

"General" BDDs do not provide a **canonical representation**, i.e., two different BDDs can represent the same formula.

Example

The BDD from the slide before, and the BDT (which is a BDD) for the same formula.

To enable fast equivalence checking, limitations regarding the BDD structure are necessary. Commonly used BDDs are almost always **Reduced Ordered Binary Decision Diagrams (ROBDDs)**.

Free BDDs

Definition (Free BDD)

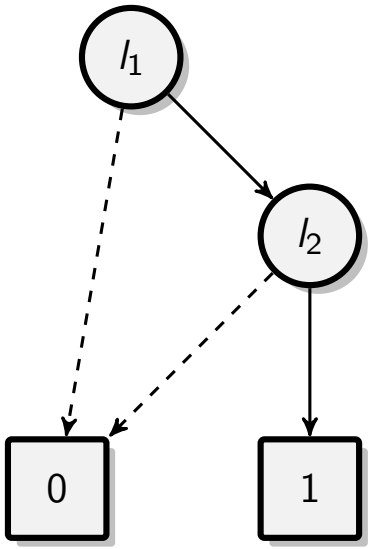
A BDD G is **free** if each variable occurs at most once along every path from the root to a terminal node.

Free BDDs

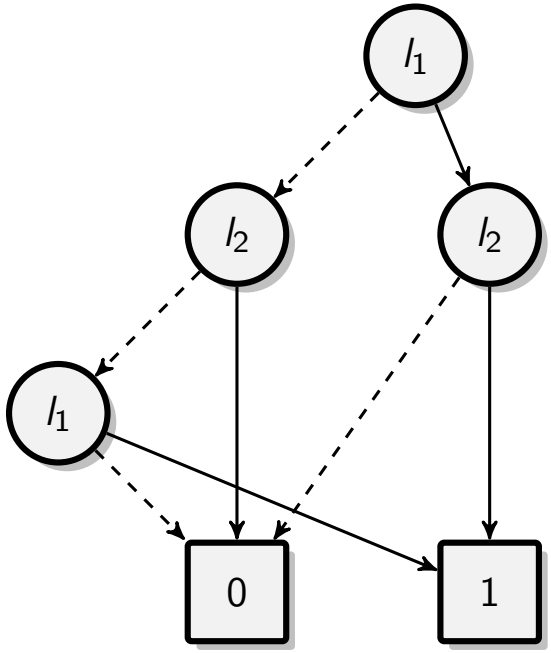
Definition (Free BDD)

A BDD G is **free** if each variable occurs at most once along every path from the root to a terminal node.

Example



free



not free

Ordered BDDs

Definition (Ordered BDD (OBDD))

A BDD G over the set of variables X is **ordered** if it is free and the variables on every path from the root to a terminal node occur in the same order.

This sequence is called **variable order**.

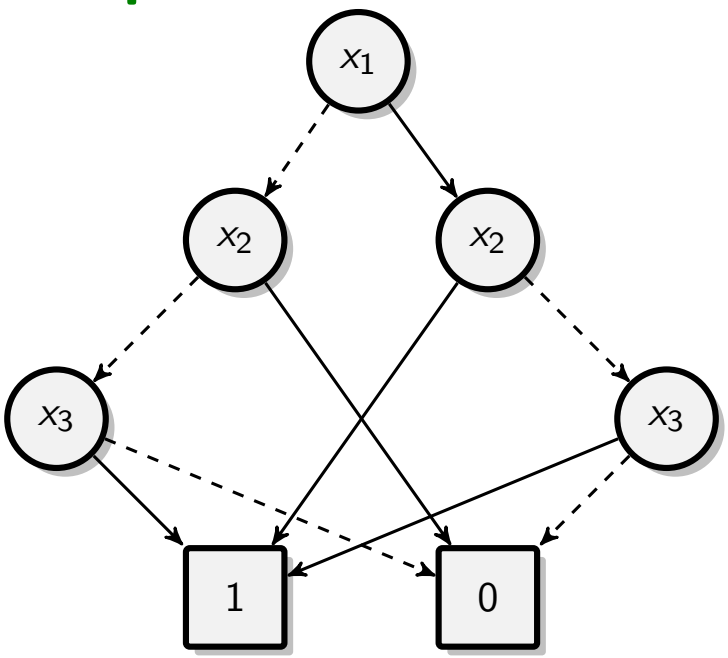
Ordered BDDs

Definition (Ordered BDD (OBDD))

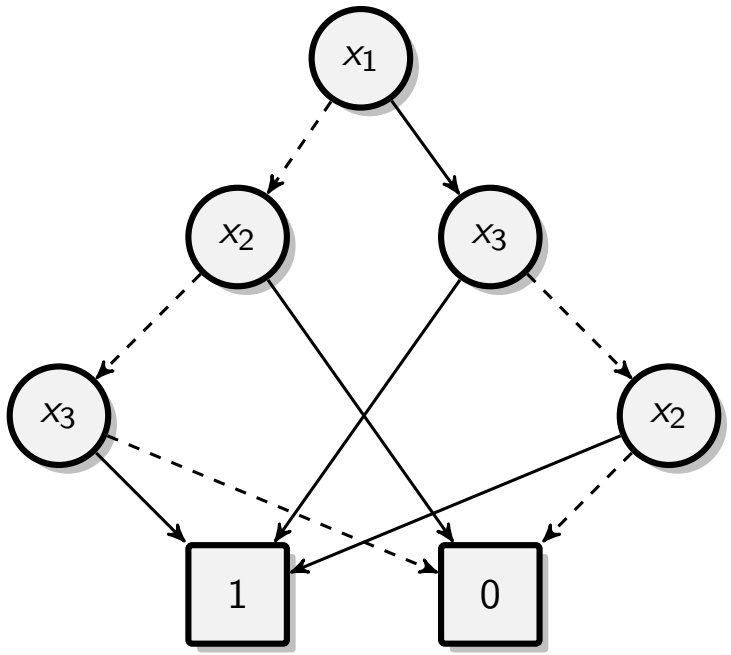
A BDD G over the set of variables X is **ordered** if it is free and the variables on every path from the root to a terminal node occur in the same order.

This sequence is called **variable order**.

Example



ordered

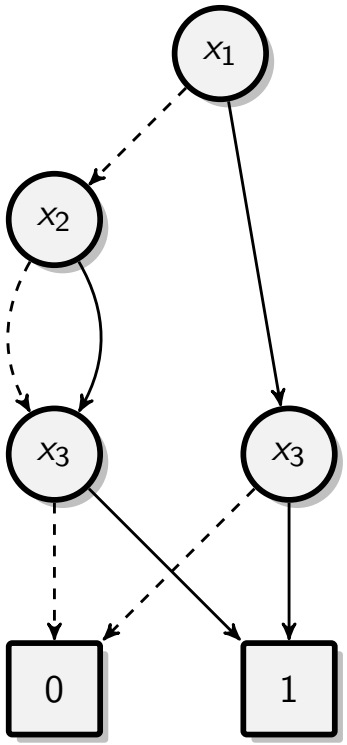


not ordered

Redundancy in OBDDs

Even ordered BDDs are not canonical, as we can still have redundancy

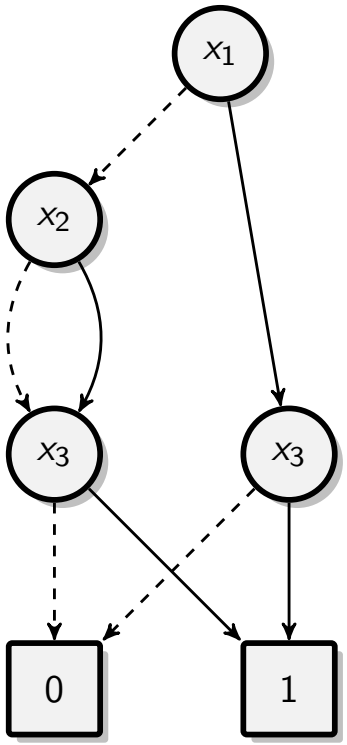
Example



Redundancy in OBDDs

Even ordered BDDs are not canonical, as we can still have redundancy

Example



How to remove redundancy?

Removing Redundancy: Isomorphism

Definition (Isomorphism Reduction)

Let $G = (V, E)$ be a BDD. Let $v \neq w \in V$ be either

- two non-terminal nodes with $var(v) = var(w)$, $low(v) = low(w)$ and $high(v) = high(w)$, or
- two terminal nodes with $val(v) = val(w)$.

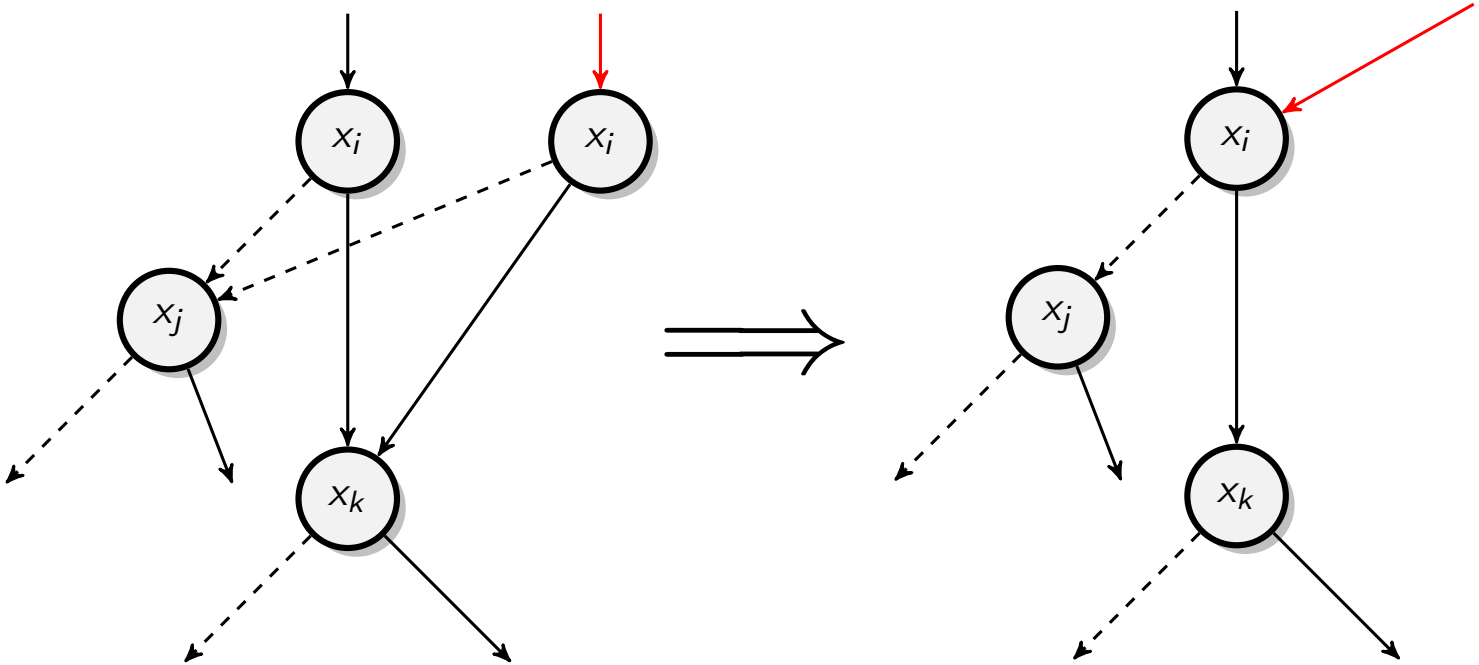
Then: redirect all edges that end in w to end in v, and remove w (and all outgoing edges, if any).

Remark

The result is a BDD that represents the same formula.

Example: Isomorphism Reduction

Example



Isomorphism reduction

Removing Redundancy: Shannon Reduction

Definition (Shannon Reduction)

Let $G = (V, E)$ be a BDD. Let $v \in V$ be a non-terminal node with $\mathit{low}(v) = \mathit{high}(v)$.

Then: redirect all edges that end in v to end in $\mathit{low}(v)$, and remove v and its outgoing edges.

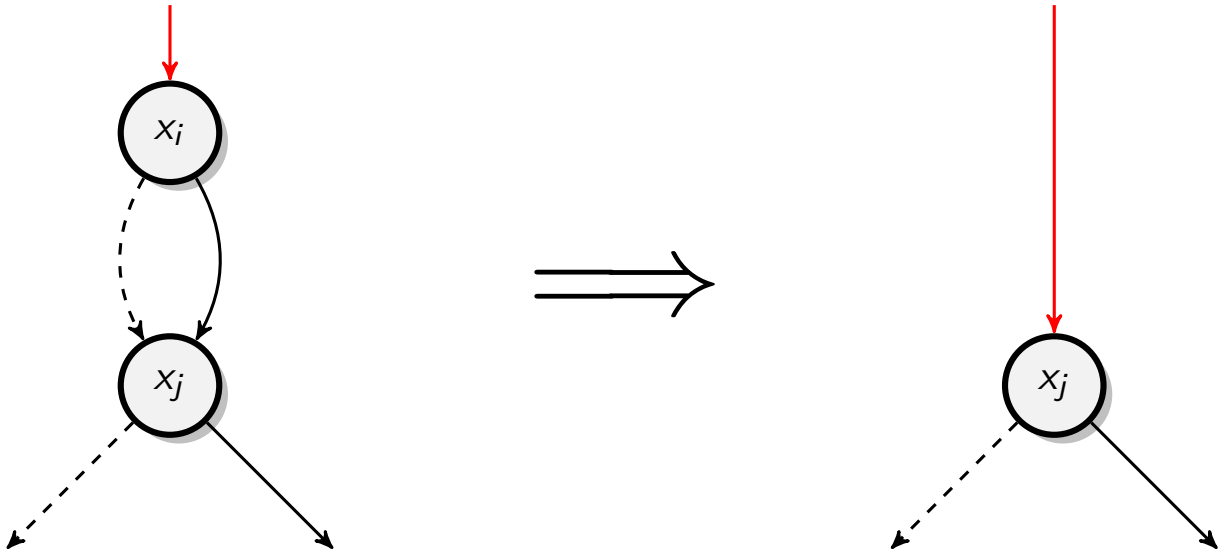
Removing Redundancy: Shannon Reduction

Definition (Shannon Reduction)

Let $G = (V, E)$ be a BDD. Let $v \in V$ be a non-terminal node with $low(v) = high(v)$.

Then: redirect all edges that end in v to end in $low(v)$, and remove v and its outgoing edges.

Example



Shannon reduction

Reduced Ordered Binary Decision Diagrams

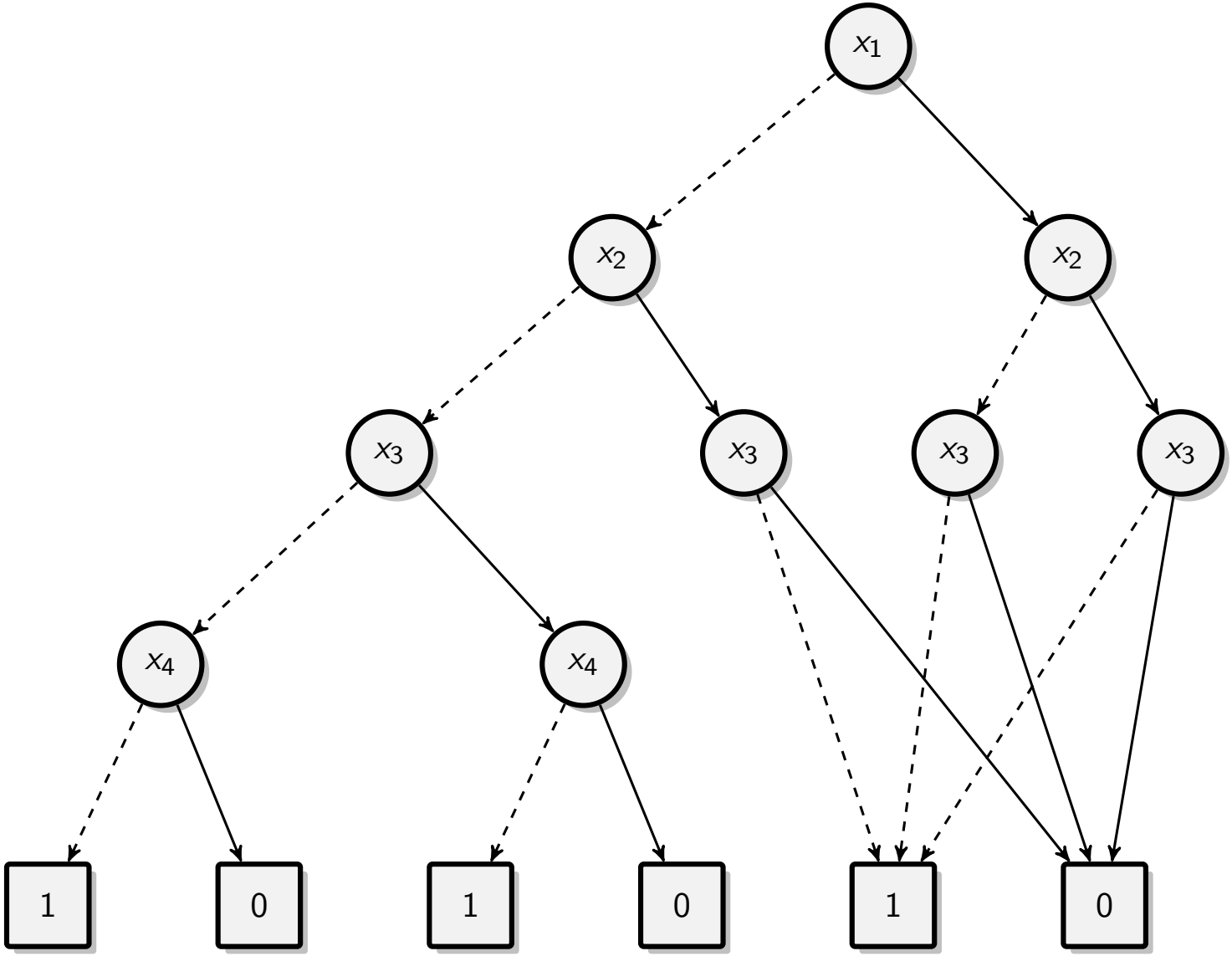
Definition

A BDD G is a **Reduced Ordered Binary Decision Diagram (ROBDD)** if it is ordered and reduced, i.e., neither the isomorphism nor the Shannon reduction can be applied.

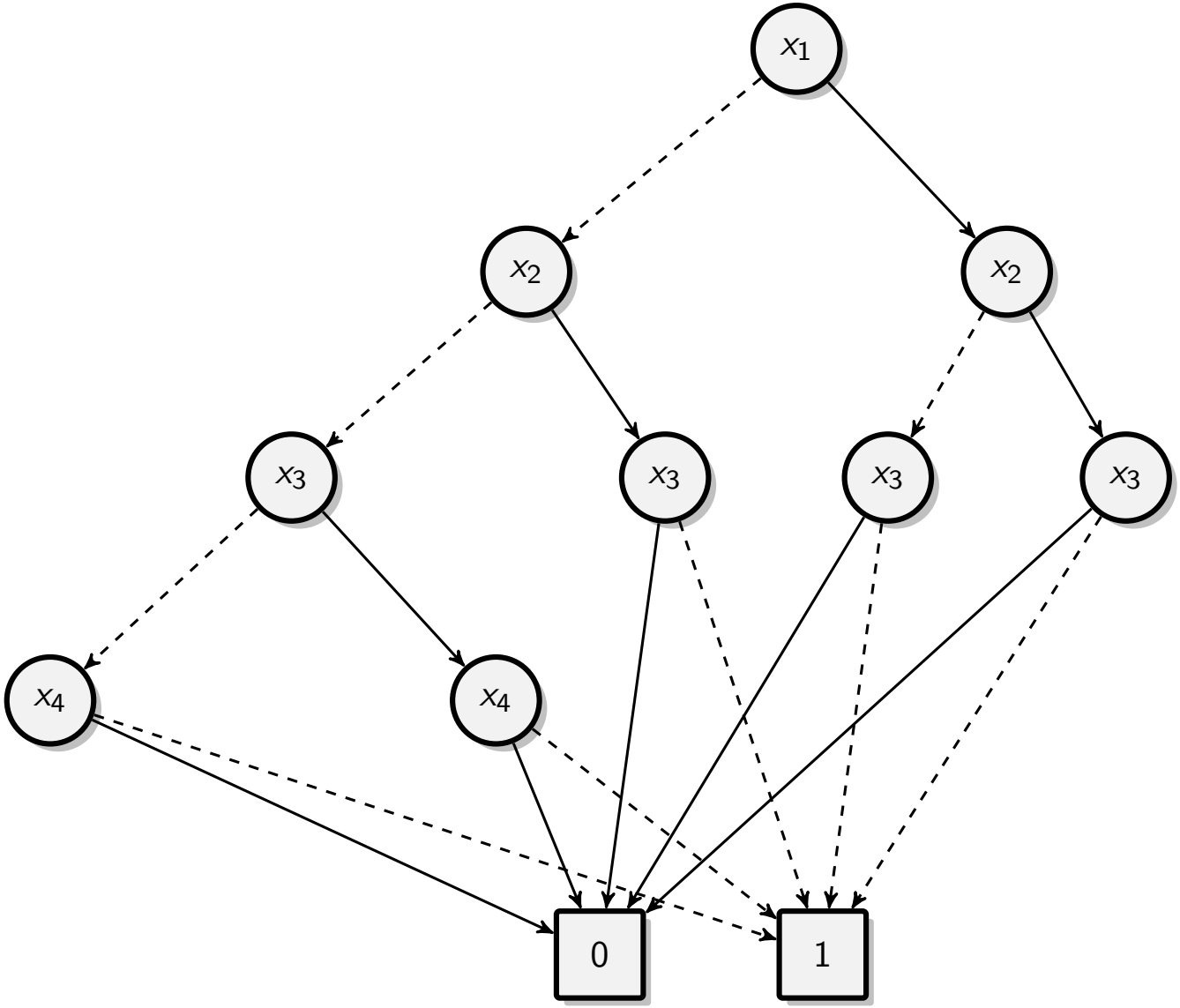
Remark

- *OBDDs can be reduced to ROBDDs in layers, starting from the terminal nodes.*
- *Current implementations of BDD-Packages do not use these reduction operations as defined. Rather, they provide BDD "managers" that may govern several BDDs (which may share nodes), and guarantee that at no time there exist two nodes that represent the same formula.*

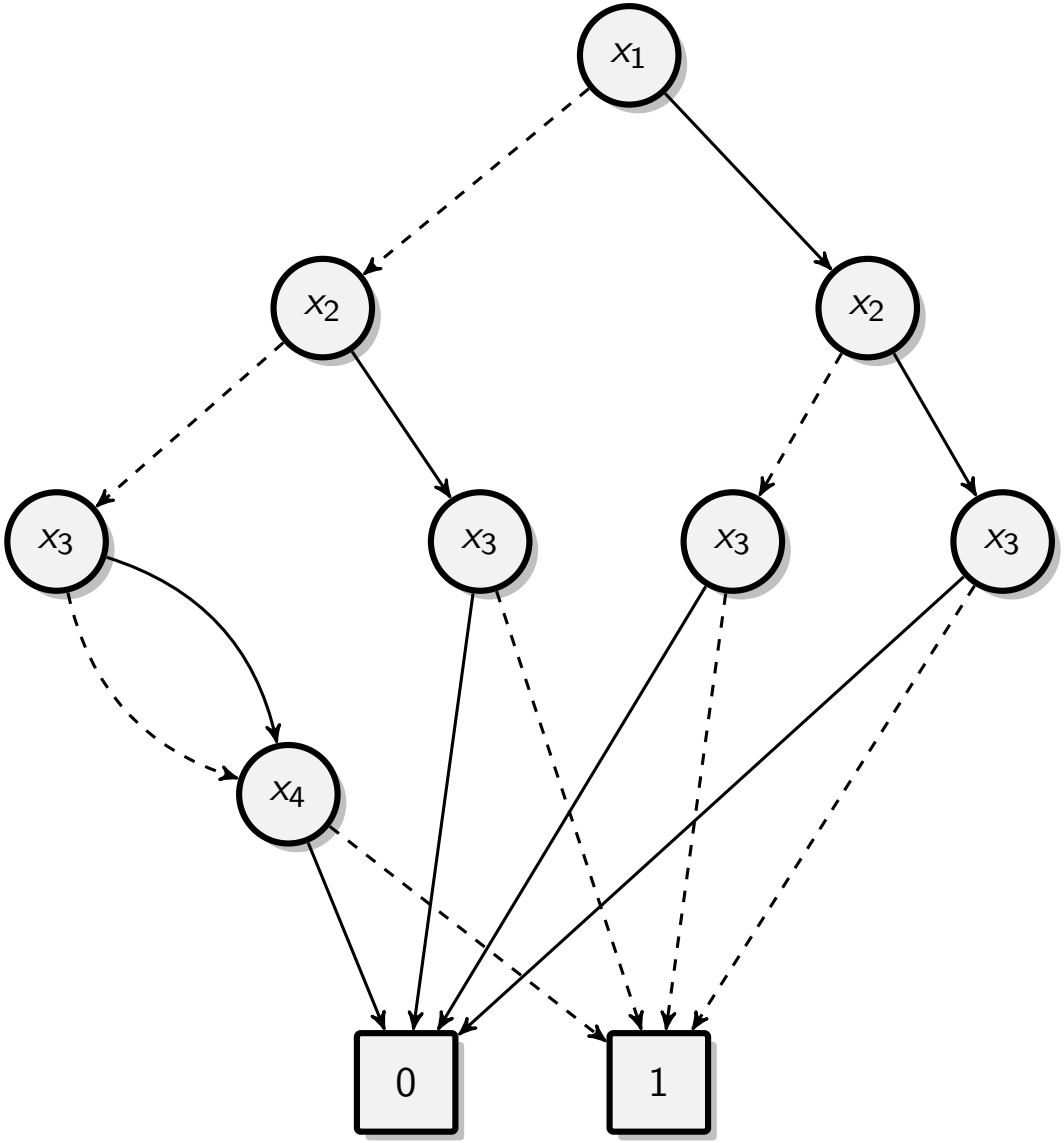
Example: Reducing an OBDD



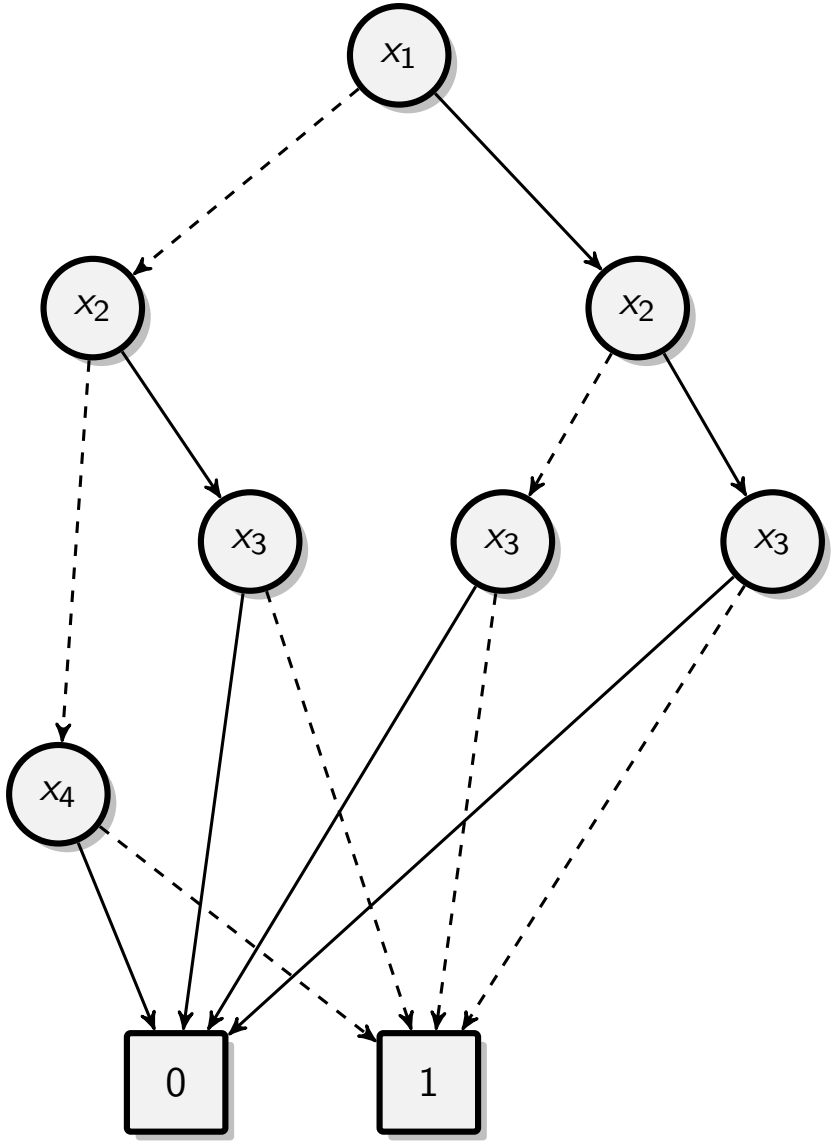
Example: Reducing an OBDD



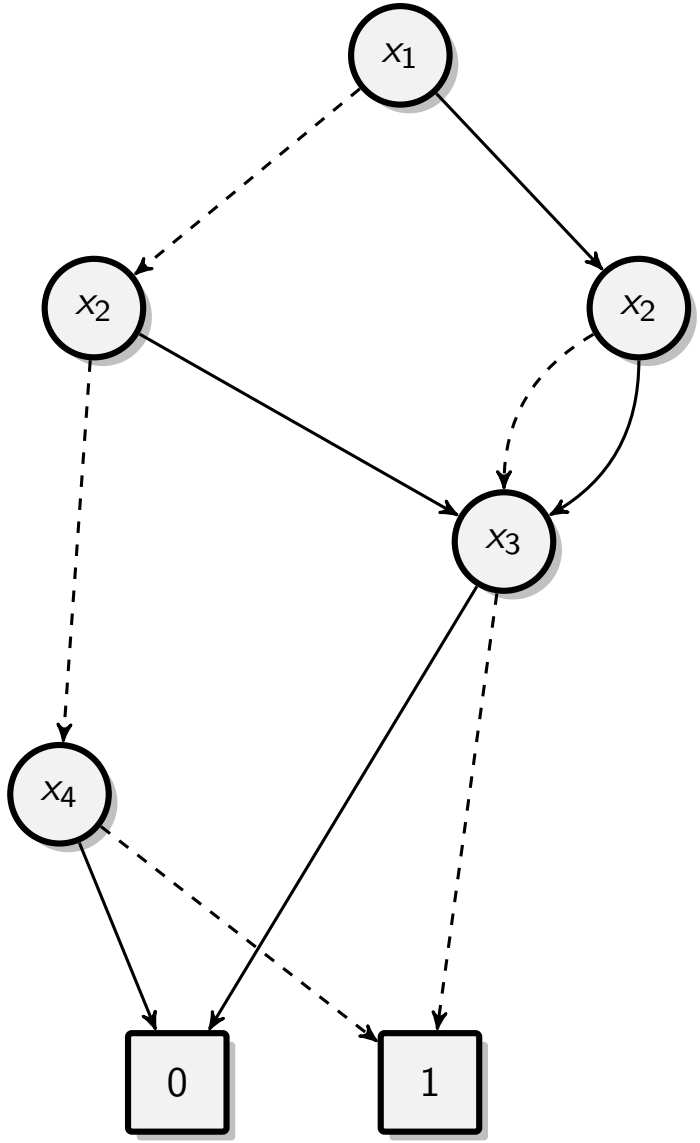
Example: Reducing an OBDD



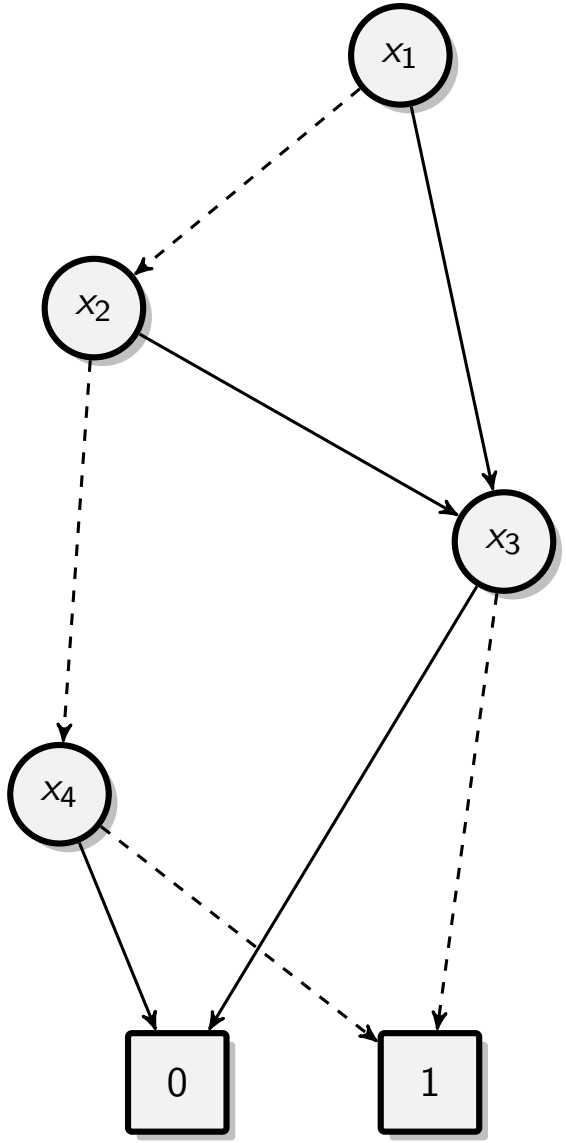
Example: Reducing an OBDD



Example: Reducing an OBDD



Example: Reducing an OBDD



Canonicity of ROBDDs: Isomorphism

Definition (Isomorphism)

Two BDDs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ over the set of variables X are **isomorphic** if there exists a bijective mapping $\alpha : V_1 \to V_2$ with the following properties:

- $\mathit{low}(\alpha(v)) = \alpha(\mathit{low}(v))$ for each $v \in V_1$
- $\mathit{high}(\alpha(v)) = \alpha(\mathit{high}(v))$ for each $v \in V_1$
- $\mathit{var}(v) = \mathit{var}(\alpha(v))$ for non-terminal nodes $v \in V_1$
- $\mathit{val}(v) = \mathit{val}(\alpha(v))$ for terminal nodes $v \in V_1$

Canonicity and Equivalence of ROBDDs

Theorem (Canonicity of ROBDDs)

Let $<$ be a fixed variable order on X. Then for each boolean formula F there exists (up to isomorphism) exactly one ROBDD over X with $<$.

Remark

This means that two ROBDDs G_1, G_2 represent the same boolean formula if and only if they are isomorphic. Isomorphism, and thus equivalence, can be checked (by synchronous depth-first search) in time $O(|V_{G_1}| + |V_{G_2}|)$.

Obtaining an ROBDD from a Formula

1. Given a formula F over variables $X = \{x_1, \ldots, x_n\}$ and a variable order $x_1 > x_2 > \ldots > x_n$, compute recursively for F and all resulting formulas:
   - $F_0 = F[x_1 := 0]$
   - $F_1 = F[x_1 := 1]$
   - for $\vec{a} \in \mathbb{B}^i$, $i < n$, and $a \in \mathbb{B}$, let $F_{\vec{a}a} = F_{\vec{a}}[x_{i+1} := a]$

Obtaining an ROBDD from a Formula

1. Given a formula F over variables $X = \{x_1, \ldots, x_n\}$ and a variable order $x_1 > x_2 > \ldots > x_n$, compute recursively for F and all resulting formulas:
   - $F_0 = F[x_1 := 0]$
   - $F_1 = F[x_1 := 1]$
   - for $\vec{a} \in \mathbb{B}^i$, $i < n$, and $a \in \mathbb{B}$, let $F_{\vec{a}a} = F_{\vec{a}}[x_{i+1} := a]$
2. Partition the resulting set of formulas into sets S_i s.t.
   - $S_1 = \{F\}$
   - for $i > 1$, S_i contains all formulas $F_{\vec{a}}$ s.t. x_i is the largest variable that $F_{\vec{a}}$ depends on
   - $S_{n+1} = \mathbb{B}$.

Obtaining an ROBDD from a Formula

1. Given a formula F over variables $X = \{x_1, \ldots, x_n\}$ and a variable order $x_1 > x_2 > \ldots > x_n$, compute recursively for F and all resulting formulas:
   - $F_0 = F[x_1 := 0]$
   - $F_1 = F[x_1 := 1]$
   - for $\vec{a} \in \mathbb{B}^i$, $i < n$, and $a \in \mathbb{B}$, let $F_{\vec{a}a} = F_{\vec{a}}[x_{i+1} := a]$
2. Partition the resulting set of formulas into sets S_i s.t.
   - $S_1 = \{F\}$
   - for $i > 1$, S_i contains all formulas $F_{\vec{a}}$ s.t. x_i is the largest variable that $F_{\vec{a}}$ depends on
   - $S_{n+1} = \mathbb{B}$.
3. Define $V = \bigcup_{i=1}^{n+1} S_i$, and for $i \in \{1, \ldots, n\}$ and $F_{\vec{a}} \in S_i$ let $var(F_{\vec{a}}) = x_i$, $low(F_{\vec{a}}) = F_{\vec{a}}[x_i := 0]$ and $high(F_{\vec{a}}) = F_{\vec{a}}[x_i := 1]$. The root of G is F.

Example: ROBDD from Formula

$F = \bar{x_1}\bar{x_2}x_3\bar{x_4} \vee \bar{x_1}x_2x_4 \vee x_1\bar{x_2}x_3$

$S_1 = \{F\}$

$S_2 =$

$S_3 =$

$S_4 =$

$S_5 = \mathbb{B}$

Example: ROBDD from Formula

$F = \bar{x_1}\bar{x_2}x_3\bar{x_4} \vee \bar{x_1}x_2x_4 \vee x_1\bar{x_2}x_3$

$S_1 = \{F\}$

$S_2 = \{\underbrace{\bar{x_2}x_3\bar{x_4} \vee x_2x_4}_{F_0}, \underbrace{\bar{x_2}x_3}_{F_1}\}$

$S_3 =$

$S_4 =$

$S_5 = \mathbb{B}$

Example: ROBDD from Formula

$$F = \bar{x_1}\bar{x_2}x_3\bar{x_4} \vee \bar{x_1}x_2x_4 \vee x_1\bar{x_2}x_3$$

$$S_1 = \{F\}$$

$$S_2 = \{\underbrace{\bar{x_2}x_3\bar{x_4} \vee x_2x_4}_{F_0}, \underbrace{\bar{x_2}x_3}_{F_1}\}$$

$$S_3 = \{\underbrace{x_3\bar{x_4}}_{F_{00}}, \underbrace{x_3}_{F_{10}}\}$$

$$S_4 = \{\underbrace{x_4}_{F_{01}}, \quad\}$$

$$S_5 = \mathbb{B}$$

Example: ROBDD from Formula

$$F = \bar{x_1}\bar{x_2}x_3\bar{x_4} \vee \bar{x_1}x_2x_4 \vee x_1\bar{x_2}x_3$$

$$S_1 = \{F\}$$

$$S_2 = \{\underbrace{\bar{x_2}x_3\bar{x_4} \vee x_2x_4}_{F_0}, \underbrace{\bar{x_2}x_3}_{F_1}\}$$

$$S_3 = \{\underbrace{x_3\bar{x_4}}_{F_{00}}, \underbrace{x_3}_{F_{10}}\}$$

$$S_4 = \{\underbrace{x_4}_{F_{01}}, \underbrace{\bar{x_4}}_{F_{001}}\}$$

$$S_5 = \mathbb{B}$$

Example: ROBDD from Formula

$$F = \bar{x_1}\bar{x_2}x_3\bar{x_4} \vee \bar{x_1}x_2x_4 \vee x_1\bar{x_2}x_3$$

$$S_1 = \{F\}$$

$$S_2 = \{\underbrace{\bar{x_2}x_3\bar{x_4} \vee x_2x_4}_{F_0}, \underbrace{\bar{x_2}x_3}_{F_1}\}$$

$$S_3 = \{\underbrace{x_3\bar{x_4}}_{F_{00}}, \underbrace{x_3}_{F_{10}}\}$$

$$S_4 = \{\underbrace{x_4}_{F_{01}}, \underbrace{\bar{x_4}}_{F_{001}}\}$$

$$S_5 = \mathbb{B}$$

(F_{11}, F_{000}, F_{100} evaluate to 0, F_{101} evaluates to 1)

Example: ROBDD from Formula

$$F = \bar{x}_1\bar{x}_2x_3\bar{x}_4 \vee \bar{x}_1x_2x_4 \vee x_1\bar{x}_2x_3$$

$$S_1 = \{F\}$$

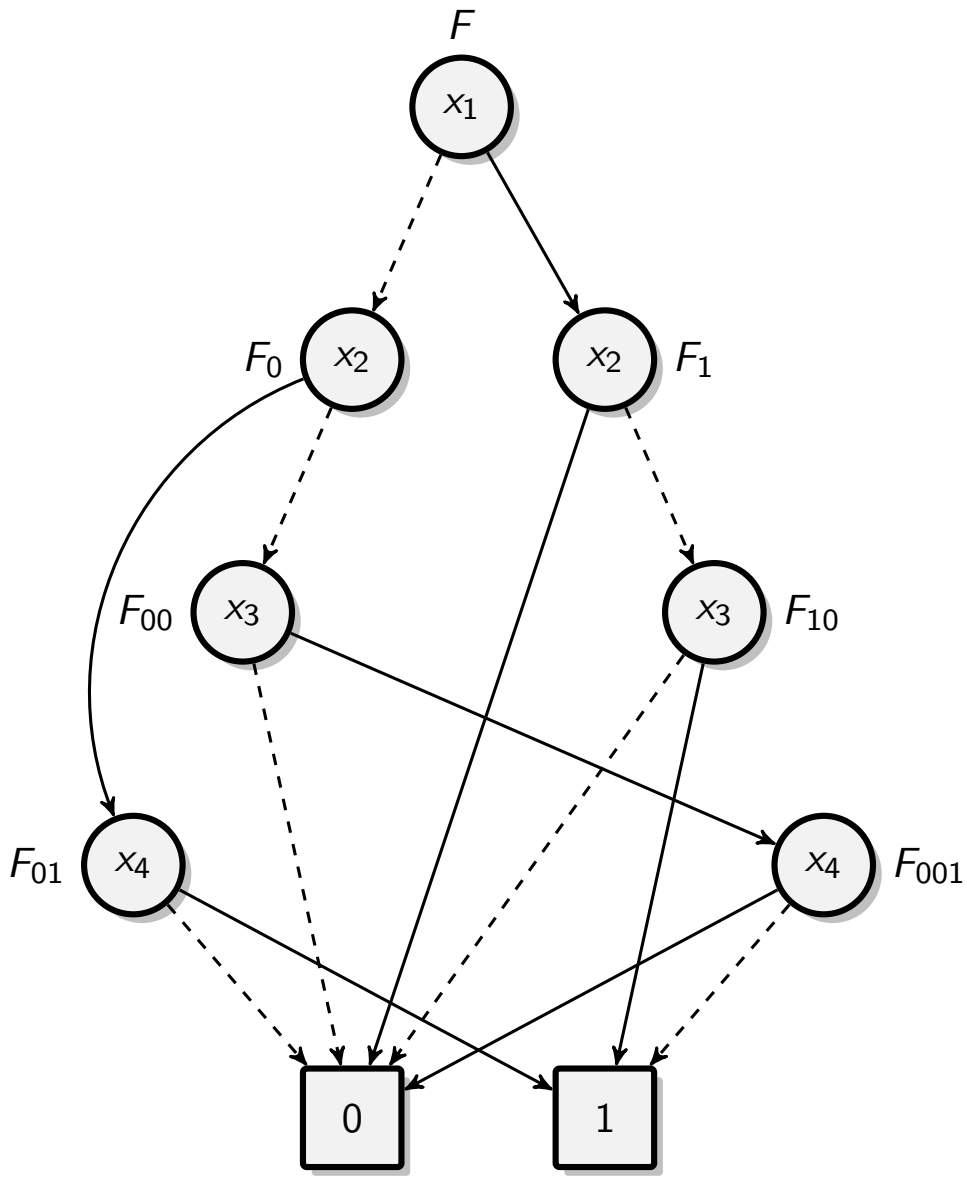
$$S_2 = \{\underbrace{\bar{x}_2x_3\bar{x}_4 \vee x_2x_4}_{F_0}, \underbrace{\bar{x}_2x_3}_{F_1}\}$$

$$S_3 = \{\underbrace{x_3\bar{x}_4}_{F_{00}}, \underbrace{x_3}_{F_{10}}\}$$

$$S_4 = \{\underbrace{x_4}_{F_{01}}, \underbrace{\bar{x}_4}_{F_{001}}\}$$

$$S_5 = \mathbb{B}$$

(F_{11}, F_{000}, F_{100} evaluate to 0, F_{101} evaluates to 1)



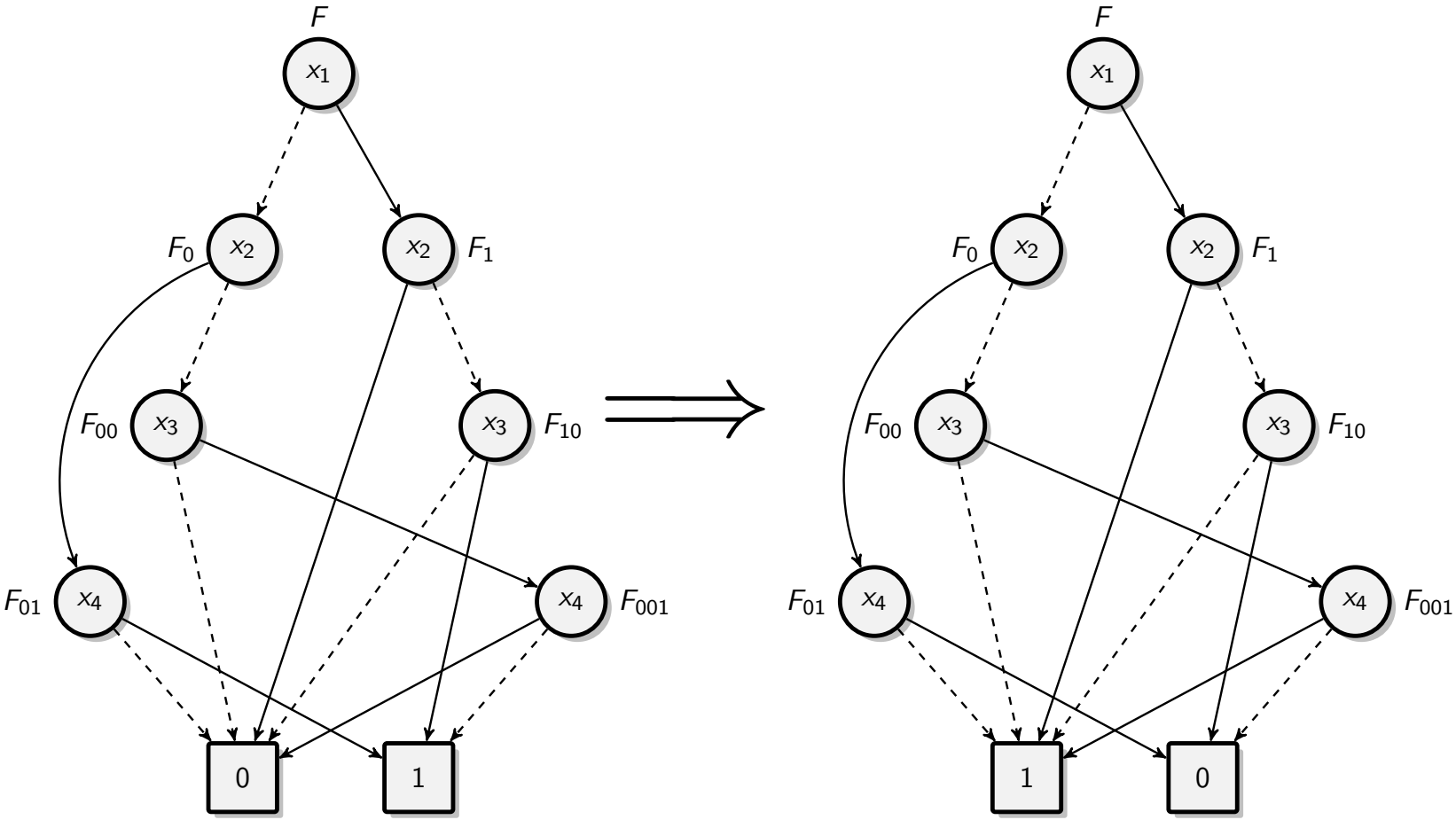
Operations on OBDDs: Negation

Let G be an OBDD for formula F. Then an OBDD for formula $\bar{F}$ is obtained by swapping the terminal nodes.

Operations on OBDDs: Negation

Let G be an OBDD for formula F. Then an OBDD for formula $\bar{F}$ is obtained by swapping the terminal nodes.

Example



Operations on OBDDs: Binary Operators

Let $\circ$ be a binary boolean operator (e.g., $\wedge, \vee, \rightarrow$) and G, H two OBDDs for formulas D, F, respectively. We want to compute an OBDD for $D \circ F$ (wrt. the same variable ordering).

The algorithm is based on the following observation, for a given variable x_i:

$$\begin{aligned} D \circ F &= x_i((D \circ F)[x_i := 1]) \vee \bar{x}_i((D \circ F)[x_i := 0]) \\ &= x_i(D[x_i := 1] \circ F[x_i := 1]) \vee \bar{x}_i(D[x_i := 0] \circ F[x_i := 0]) \end{aligned}$$

Operations on OBDDs: Binary Operators

Let $\circ$ be a binary boolean operator (e.g., $\wedge, \vee, \rightarrow$) and G, H two OBDDs for formulas D, F, respectively. We want to compute an OBDD for $D \circ F$ (wrt. the same variable ordering).

The algorithm is based on the following observation, for a given variable x_i:

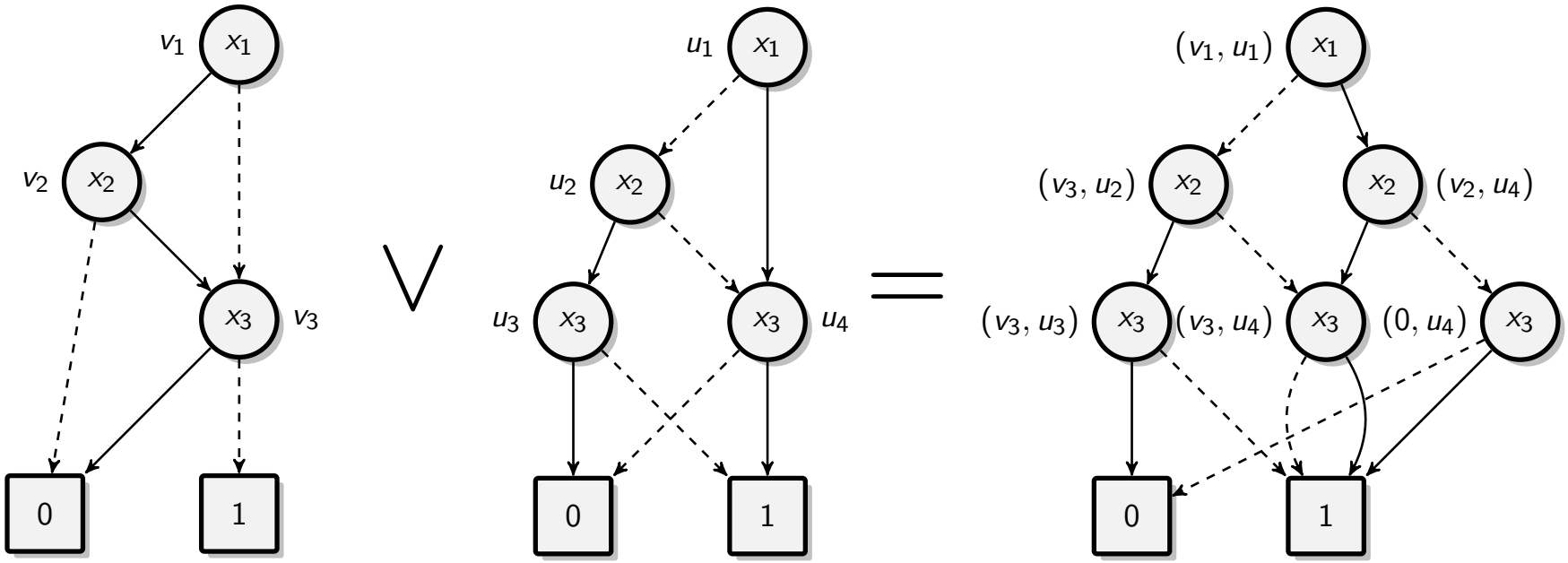
$$\begin{aligned} D \circ F &= x_i((D \circ F)[x_i := 1]) \vee \bar{x}_i((D \circ F)[x_i := 0]) \\ &= x_i(D[x_i := 1] \circ F[x_i := 1]) \vee \bar{x}_i(D[x_i := 0] \circ F[x_i := 0]) \end{aligned}$$

Notation: for BDD $G = (V, E)$ and $v \in V$, let G_v be the sub-BDD of G with root v.

Example

If BDD G represents formula F and is rooted in v with $var(v) = x$, then the formula $F[x := 1]$ is represented by the sub-BDD $G_{high(v)}$, and $F[x := 0]$ is represented by $G_{low(v)}$.

Example: Computing Binary Operators



Operations on OBDDs: Algorithm APPLY

Input: OBDDs G for F, H for D, binary operator $\circ$
Output: OBDD for $F \circ D$
Apply$(G, H, \circ)$

1. if (G and H are terminal nodes) then return $G \circ H$
2. if $(G, H) \in \mathit{HashTable}$ then return $\mathit{Hashtable}(G, H)$
3. let x be the maximal variable that G or H depend on
4. for $B \in \{G, H\}$ do: if $\mathit{var}(\mathit{root}(B)) = x$ then let $(B_0, B_1) = (B_{\mathit{low}(\mathit{root}(B))}, B_{\mathit{high}(\mathit{root}(B))})$ else let $B_0 = B_1 = B$
5. construct G_{new} with new node v_{new} as root and
   $\mathit{var}(v_{new}) = x$
   $\mathit{low}(v_{new}) = \textsc{Apply}(G_0, H_0, \circ)$
   $\mathit{high}(v_{new}) = \textsc{Apply}(G_1, H_1, \circ)$
6. add (G, H, G_{new}) to $\mathit{HashTable}$
7. return G_{new}

Correctness and Complexity of Apply

Theorem

If G, H are OBDDs for formulas F, D wrt. the same variable order and $\circ$ is a binary boolean operator, then:

- APPLY *computes in time $O(|G| \cdot |H|)$ an OBDD for $F \circ D$*
- *the size of the resulting OBDD is bounded by $|G| \cdot |H|$.*

Proof Idea. Correctness can be shown by induction on the OBDD structure.
Bounds on complexity and size follow from the use of *HashTable*: every pair of nodes causes at most two recursive calls, and everything except the recursive calls has constant complexity.