

Digital Systems and Design Automation Group
dynamo.ethz.ch

Exercise: Specification and Verification Using Set Operations and BDDs

Jiahui Xu

Discrete Event Systems, HS 2026

Computer Engineering and Networks Laboratory

ETH zürich

About Myself

Jiahui Xu

- PhD student in the DYNAMO group since April 2022

Research topics that I am interested in:

- High-level synthesis (HLS): compile C/C++ code into digital circuits
- **Optimization and formal verification of digital circuits**

Email: jxu@ethz.ch Office: ETZ G75 Website: jiahui17.github.io

Schedule

We have five exercise sessions:

- 12.11.2026: set operations, characteristic functions, BDDs
- 19.11.2026: reachability analysis and temporal logic
- 26.11.2026: no lecture and no exercise
- 03.12.2026: to be announced
- 10.12.2026: Petri nets
- 17.12.2026: time Petri nets

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

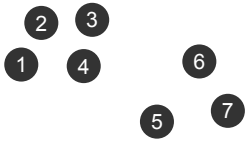
- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

Characteristic function: $\psi_N(\sigma(s))$ *evaluates to 1 if* $s \in N$*, and 0 otherwise.*

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

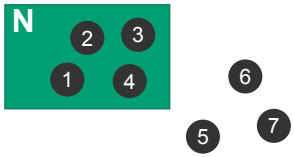


Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

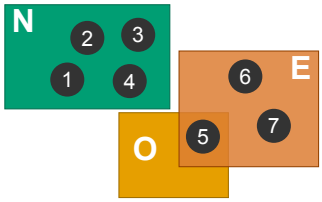


Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

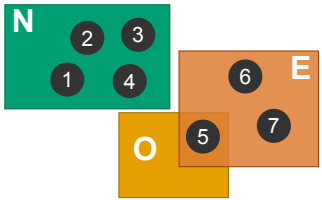


Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**



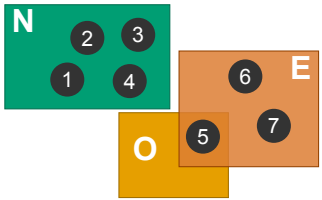
(a) What is the characteristic function ψ_X of the set of *all states* X?

Characteristic function: $\psi_N(\sigma(s))$ *evaluates to 1 if* $s \in N$*, and 0 otherwise.*

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**



(a) What is the characteristic function ψ_X of the set of *all states* X?

$$\psi_X(\sigma(s)) = 1$$

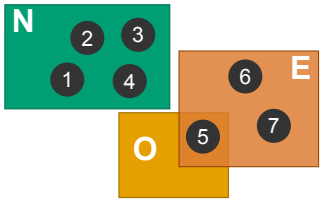
Because state s is always in the set of all states.

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**



(b) Describe Q_b, the set of states that are normal states, error states, or both.

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

(b) Describe Q_b, the set of states that are normal states, error states, or both.

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

(b) Describe Q_b, the set of states that are normal states, error states, or both.

States satisfying this property: $\{ s : s \in N \vee s \in E \}$

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

(b) Describe Q_b, the set of states that are normal states, error states, or both.

States satisfying this property: $\{ s : s \in N \vee s \in E \}$

Set notation: $Q_b = N \cup E$

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

(b) Describe Q_b, the set of states that are normal states, error states, or both.

States satisfying this property: $\{ s : s \in N \vee s \in E \}$

Set notation: $Q_b = N \cup E$ Characteristic function: $\psi_{Q_b} = \psi_N + \psi_E$

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

Describe the following sets using sets and characteristic functions:

(c) Q_c, the set of states s such that if s is an overflow state, then s is not a normal state.

(d) Q_d, the set of error states that are not overflow states.

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Convention: if A then B

Let’s agree on the following convention…

Convention: if A then B

Let’s agree on the following convention…

When we test a specification “if A then B”, we are actually executing the following program:

Convention: if A then B

Let's agree on the following convention…

When we test a specification "if A then B", we are actually executing the following program:

```
// if A is true, then
if (A) {
    // check if B is true
    assert(B);
}
// the execution is ok
return(0);
```

A	B	What happens to this code?

assert(B): the program crashes if B is false, and the program ignores the statement if B is true.

Convention: if A then B

Let's agree on the following convention…

When we test a specification "if A then B", we are actually executing the following program:

```
// if A is true, then
if (A) {
    // check if B is true
    assert(B);
}
// the execution is ok
return(0);
```

A	B	What happens to this code?
False	False	Good

assert(B): the program crashes if B is false, and the program ignores the statement if B is true.

Convention: if A then B

Let's agree on the following convention…

When we test a specification "if A then B", we are actually executing the following program:

```
// if A is true, then
if (A) {
    // check if B is true
    assert(B);
}
// the execution is ok
return(0);
```

A	B	What happens to this code?
False	False	Good
False	True	Good

assert(B): the program crashes if B is false, and the program ignores the statement if B is true.

Convention: if A then B

Let's agree on the following convention…

When we test a specification "if A then B", we are actually executing the following program:

```
// if A is true, then
if (A) {
    // check if B is true
    assert(B);
}
// the execution is ok
return(0);
```

A	B	What happens to this code?
False	False	Good
False	True	Good
True	True	Good

assert(B): the program crashes if B is false, and the program ignores the statement if B is true.

Let's agree on the following convention…

When we test a specification "if A then B", we are actually executing the following program:

```
// if A is true, then
if (A) {
    // check if B is true
    assert(B);
}
// the execution is ok
return(0);
```

A	B	What happens to this code?
False	False	Good
False	True	Good
True	False	Assertion Error
True	True	Good

assert(B): the program crashes if B is false, and the program ignores the statement if B is true.

Convention: if A then B

Let's agree on the following convention…

When we test a specification "if A then B", we are actually executing the following program:

```
// if A is true, then
if (A) {
    // check if B is true
    assert(B);
}
// the execution is ok
return(0);
```

A	B	What happens to this code?
False	False	Good
False	True	Good
True	False	Assertion Error
True	True	Good

Conclusion: "if A then B" is the same as "(not A) or B"

assert(B): the program crashes if B is false, and the program ignores the statement if B is true.

Convention: if A then B

Material implication (rule of inference)

2 languages

Article Talk | Read Edit View history Tools

From Wikipedia, the free encyclopedia

For other uses, see Material implication (disambiguation).
Not to be confused with Material inference.

This article **may be too technical for most readers to understand**. Please help improve it to make it understandable to non-experts, without removing the technical details. *(December 2018)* *(template removal help)*

In propositional logic, **material implication**[1][2] is a valid rule of replacement that allows for a conditional statement to be replaced by a disjunction in which the antecedent is negated. The rule states that *P implies Q* is logically equivalent to *not-P or Q* and that either form can replace the other in logical proofs. In other words, if P is true, then Q must also be true, while if Q is *not* true, then P cannot be true either; additionally, when P is not true, Q may be either true or false.

$$P \to Q \Leftrightarrow \neg P \lor Q$$

Where "$\Leftrightarrow$" is a metalogical symbol representing "can be replaced in a proof with," *P* and *Q* are any given logical statements, and $\neg P \lor Q$ can be read as "(not *P*) or *Q*". To illustrate this, consider the following statements:

- P: Sam ate an orange for lunch
- Q: Sam ate a fruit for lunch

Then, to say, "Sam ate an orange for lunch" *implies* "Sam ate a fruit for lunch" ($P \to Q$). Logically, if Sam did not eat a fruit for lunch, then Sam also cannot have eaten an orange for lunch (by contraposition). However, merely saying that Sam did not eat an orange for lunch provides no information on whether or not Sam ate a fruit (of any kind) for lunch.

Material implication

Type	Rule of replacement
Field	Propositional calculus
Statement	*P implies Q* is logically equivalent to *not-P* or *Q*. Either form can replace the other in logical proofs.
Symbolic statement	$P \to Q \Leftrightarrow \neg P \lor Q$

Transformation rules

Propositional calculus

Rules of inference

Implication introduction / elimination (*modus ponens*)
Biconditional introduction / elimination
Conjunction introduction / elimination
Disjunction introduction / elimination
Disjunctive / hypothetical syllogism

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

Describe the following sets using sets and characteristic functions:

(c) Q_c, the set of states s such that if s is an overflow state, then s is not a normal state.

(d) Q_d, the set of error states that are not overflow states.

Your turn!

Rule of inference: "if A then B" is the same as "(not A) or B"

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

(c) Describe Q_c, the set of states s such that if s is an overflow state, then s is not a normal state.

Rule of inference: "if A then B" is the same as "(not A) or B"

Complement: X is the set of all states; the complement of a set A is $X \setminus A$.

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

(c) Describe Q_c, the set of states s such that if s is an overflow state, then s is not a normal state.

States satisfying this property: $\{ s : s \in O \rightarrow s \notin N \}$

Rule of inference: "if A then B" is the same as "(not A) or B"
Complement: X is the set of all states; the complement of a set A is $X \setminus A$.
Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

(c) Describe Q_c, the set of states s such that if s is an overflow state, then s is not a normal state.

States satisfying this property: $\{ s : s \in O \rightarrow s \notin N \} \xrightarrow{\text{simplified}} \{ s : s \notin O \vee s \notin N \}$

Rule of inference: "if A then B" is the same as "(not A) or B"

Complement: X is the set of all states; the complement of a set A is $X \setminus A$.

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

(c) Describe Q_c, the set of states s such that if s is an overflow state, then s is not a normal state.

States satisfying this property: $\{ s : s \in O \rightarrow s \notin N \} \xrightarrow{\text{simplified}} \{ s : s \notin O \vee s \notin N \}$

Set notation: $Q_c = X \setminus (O \cap N)$

Rule of inference: "if A then B" is the same as "(not A) or B"

Complement: X is the set of all states; the complement of a set A is $X \setminus A$.

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

(c) Describe Q_c, the set of states s such that if s is an overflow state, then s is not a normal state.

States satisfying this property: $\{ s : s \in O \rightarrow s \notin N \} \xrightarrow{\text{simplified}} \{ s : s \notin O \vee s \notin N \}$

Set notation: $Q_c = X \setminus (O \cap N)$ Characteristic function: $\psi_{Q_c} = \overline{\psi_O \cdot \psi_N}$

Rule of inference: "if A then B" is the same as "(not A) or B"

Complement: X is the set of all states; the complement of a set A is $X \setminus A$.

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

(d) Describe Q_d, the set of error states that are not overflow states.

Rule of inference: "if A then B" is the same as "(not A) or B"
Complement: X is the set of all states; the complement of a set A is $X \setminus A$.
Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

(d) Describe Q_d, the set of error states that are not overflow states.

States satisfying this property: $\{ s : s \notin O \wedge s \in E \}$

Rule of inference: "if A then B" is the same as "(not A) or B"

Complement: X is the set of all states; the complement of a set A is $X \setminus A$.

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

(d) Describe Q_d, the set of error states that are not overflow states.

States satisfying this property: $\{ s : s \notin O \wedge s \in E \}$

Set notation: $Q_d = (X \setminus O) \cap E$

Rule of inference: "if A then B" is the same as "(not A) or B"

Complement: X is the set of all states; the complement of a set A is $X \setminus A$.

Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.1: Set Operations and Characteristic Functions

Program states classified in sets

- **X: Set of all states**
- **N: Set of normal states**
- **E: Set of error states**
- **O: Set of states with memory overflow**

The union of two sets A and B is the set of elements which are in A, in B, or in both A and B: $A \cup B = \{x : x \in A \vee x \in B\}$.

The intersection of two sets A and B, denoted by $A \cap B$, is the set of all objects that are members of both A and B: $A \cap B = \{x : x \in A \wedge x \in B\}$.

(d) Describe Q_d, the set of error states that are not overflow states.

States satisfying this property: $\{ s : s \notin O \wedge s \in E \}$

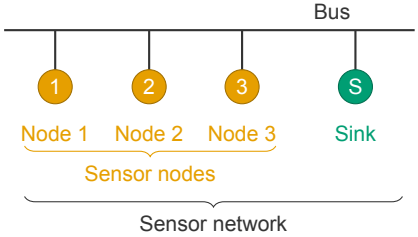
Set notation: $Q_d = (X \setminus O) \cap E$ Characteristic function: $\psi_{Q_d} = \overline{\psi_O} \cdot \psi_E$

Rule of inference: "if A then B" is the same as "(not A) or B"

Complement: X is the set of all states; the complement of a set A is $X \setminus A$.

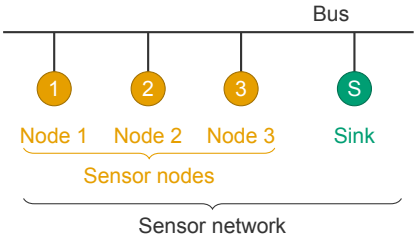
Characteristic function: $\psi_N(\sigma(s))$ evaluates to 1 if $s \in N$, and 0 otherwise.

Q1.2: Specifications in Boolean Encoding



Rule of inference: "if A then B" is the same as "(not A) or B"

Q1.2: Specifications in Boolean Encoding

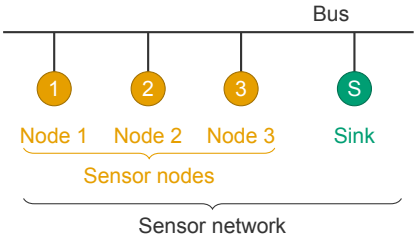


We can use binary variables to indicate the state of the system:

- $x_1 = 1$: node 1 is using the bus
- $x_2 = 1$: node 2 is using the bus
- $x_3 = 1$: node 3 is using the bus
- $x_s = 1$: the sink is awake
- $x_b = 1$: the system is bootstrapping

Rule of inference: "if A then B" is the same as "(not A) or B"

Q1.2: Specifications in Boolean Encoding



We can use binary variables to indicate the state of the system:

- $x_1 = 1$: node 1 is using the bus
- $x_2 = 1$: node 2 is using the bus
- $x_3 = 1$: node 3 is using the bus
- $x_s = 1$: the sink is awake
- $x_b = 1$: the system is bootstrapping

Describe the 3 constraints as characteristic functions ψ_{C_1}, ψ_{C_2}, and ψ_{C_3}:

(C1) When one or more nodes are using the bus, the sink must be awake to receive data.

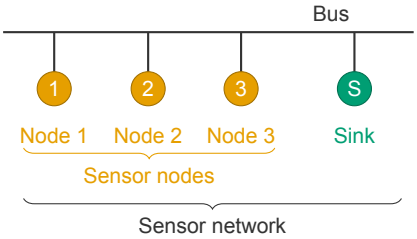
(C2) No more than one node can use the bus at the same time.

(C3) When the network is in bootstrapping mode, then the sink must be awake, and the nodes cannot use the bus.

Your turn!

Rule of inference: "if A then B" is the same as "(not A) or B"

Q1.2: Specifications in Boolean Encoding



We can use binary variables to indicate the state of the system:

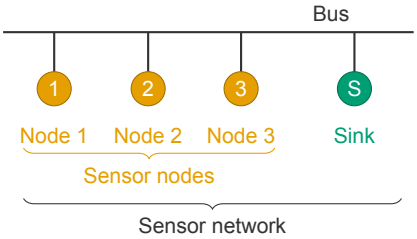
- $x_1 = 1$: node 1 is using the bus
- $x_2 = 1$: node 2 is using the bus
- $x_3 = 1$: node 3 is using the bus
- $x_s = 1$: the sink is awake
- $x_b = 1$: the system is bootstrapping

(C1) When one or more nodes are using the bus, the sink must be awake to receive data.

Not (Not (If "**one or more nodes are using the bus**" then "**the sink is awake**"))

Rule of inference: "if A then B" is the same as "(not A) or B"

Q1.2: Specifications in Boolean Encoding



We can use binary variables to indicate the state of the system:

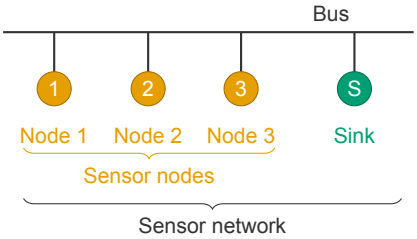
- $x_1 = 1$: node 1 is using the bus
- $x_2 = 1$: node 2 is using the bus
- $x_3 = 1$: node 3 is using the bus
- $x_s = 1$: the sink is awake
- $x_b = 1$: the system is bootstrapping

(C1) When one or more nodes are using the bus, the sink must be awake to receive data.

Not (Not (If "**one or more nodes are using the bus**" then "**the sink is awake**"))
Not ("**not (no node is using the bus)**" and "**the sink is not awake**")

Rule of inference: "if A then B" is the same as "(not A) or B"

Q1.2: Specifications in Boolean Encoding



We can use binary variables to indicate the state of the system:

- $x_1 = 1$: node 1 is using the bus
- $x_2 = 1$: node 2 is using the bus
- $x_3 = 1$: node 3 is using the bus
- $x_s = 1$: the sink is awake
- $x_b = 1$: the system is bootstrapping

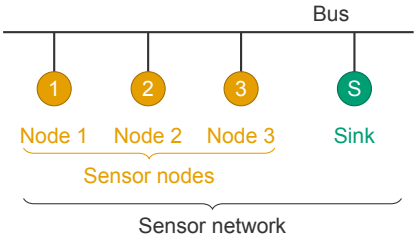
(C1) When one or more nodes are using the bus, the sink must be awake to receive data.

Not (Not (If "**one or more nodes are using the bus**" then "**the sink is awake**"))
Not ("**not (no node is using the bus)**" and "**the sink is not awake**")

$$\psi_{C_1} = \overline{\overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3} \cdot \overline{x_s}}$$

Rule of inference: "if A then B" is the same as "(not A) or B"

Q1.2: Specifications in Boolean Encoding



We can use binary variables to indicate the state of the system:

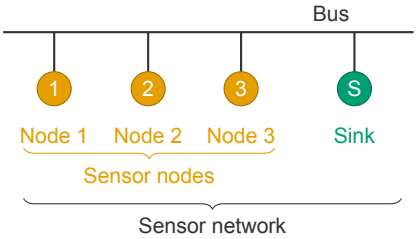
- $x_1 = 1$: node 1 is using the bus
- $x_2 = 1$: node 2 is using the bus
- $x_3 = 1$: node 3 is using the bus
- $x_s = 1$: the sink is awake
- $x_b = 1$: the system is bootstrapping

(C2) No more than one node can use the bus at the same time.

"**exactly one node is using the bus**" or "**no node is using the bus**"

Rule of inference: "if A then B" is the same as "(not A) or B"

Q1.2: Specifications in Boolean Encoding



We can use binary variables to indicate the state of the system:

- $x_1 = 1$: node 1 is using the bus
- $x_2 = 1$: node 2 is using the bus
- $x_3 = 1$: node 3 is using the bus
- $x_s = 1$: the sink is awake
- $x_b = 1$: the system is bootstrapping

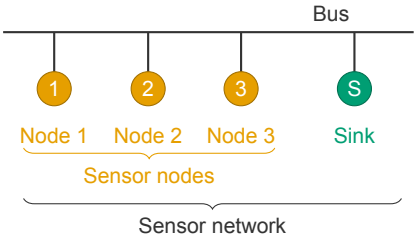
(C2) No more than one node can use the bus at the same time.

"**exactly one node is using the bus**" or "**no node is using the bus**"

$$\psi_{C_2} = x_1 \cdot \overline{x_2} \cdot \overline{x_3} + \overline{x_1} \cdot x_2 \cdot \overline{x_3} + \overline{x_1} \cdot \overline{x_2} \cdot x_3 + \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_3}$$

Rule of inference: "if A then B" is the same as "(not A) or B"

Q1.2: Specifications in Boolean Encoding



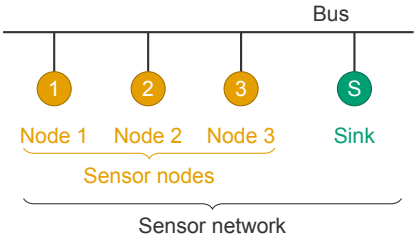
We can use binary variables to indicate the state of the system:

- $x_1 = 1$: node 1 is using the bus
- $x_2 = 1$: node 2 is using the bus
- $x_3 = 1$: node 3 is using the bus
- $x_s = 1$: the sink is awake
- $x_b = 1$: the system is bootstrapping

(C3) When the network is in bootstrapping mode, then the sink must be awake, and the nodes cannot use the bus.

Rule of inference: "if A then B" is the same as "(not A) or B"

Q1.2: Specifications in Boolean Encoding



We can use binary variables to indicate the state of the system:

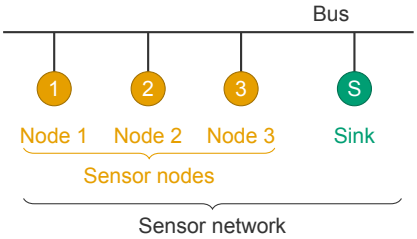
- $x_1 = 1$: node 1 is using the bus
- $x_2 = 1$: node 2 is using the bus
- $x_3 = 1$: node 3 is using the bus
- $x_s = 1$: the sink is awake
- $x_b = 1$: the system is bootstrapping

(C3) When the network is in bootstrapping mode, then the sink must be awake, and the nodes cannot use the bus.

If "**the system is bootstrapping**" then ("**the sink is awake**" and "**no node is using the bus**")

Rule of inference: "if A then B" is the same as "(not A) or B"

Q1.2: Specifications in Boolean Encoding



We can use binary variables to indicate the state of the system:

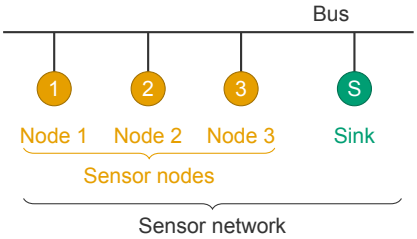
- $x_1 = 1$: node 1 is using the bus
- $x_2 = 1$: node 2 is using the bus
- $x_3 = 1$: node 3 is using the bus
- $x_s = 1$: the sink is awake
- $x_b = 1$: the system is bootstrapping

(C3) When the network is in bootstrapping mode, then the sink must be awake, and the nodes cannot use the bus.

If "**the system is bootstrapping**" then ("**the sink is awake**" and "**no node is using the bus**")
Not "**the system is bootstrapping**" or ("**the sink is awake**" and "**no node is using the bus**")

Rule of inference: "if A then B" is the same as "(not A) or B"

Q1.2: Specifications in Boolean Encoding



We can use binary variables to indicate the state of the system:

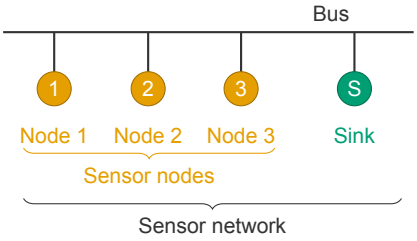
- $x_1 = 1$: node 1 is using the bus
- $x_2 = 1$: node 2 is using the bus
- $x_3 = 1$: node 3 is using the bus
- $x_s = 1$: the sink is awake
- $x_b = 1$: the system is bootstrapping

(C3) When the network is in bootstrapping mode, then the sink must be awake, and the nodes cannot use the bus.

If "**the system is bootstrapping**" then ("**the sink is awake**" and "**no node is using the bus**")
Not "**the system is bootstrapping**" or ("**the sink is awake**" and "**no node is using the bus**")
Not ("**the system is bootstrapping**" and ("**the sink is not awake**" or "**one or more nodes are using the bus**"))

Rule of inference: "if A then B" is the same as "(not A) or B"

Q1.2: Specifications in Boolean Encoding



We can use binary variables to indicate the state of the system:

- $x_1 = 1$: node 1 is using the bus
- $x_2 = 1$: node 2 is using the bus
- $x_3 = 1$: node 3 is using the bus
- $x_s = 1$: the sink is awake
- $x_b = 1$: the system is bootstrapping

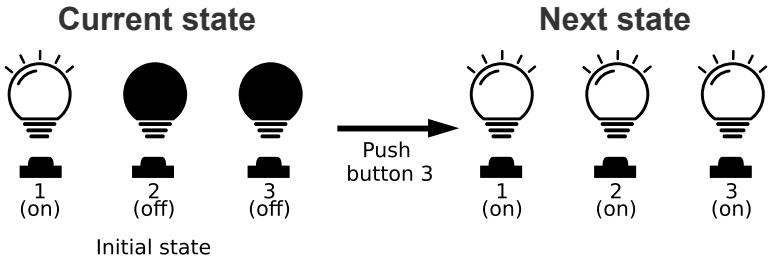
(C3) When the network is in bootstrapping mode, then the sink must be awake, and the nodes cannot use the bus.

If "**the system is bootstrapping**" then ("**the sink is awake**" and "**no node is using the bus**")
Not "**the system is bootstrapping**" or ("**the sink is awake**" and "**no node is using the bus**")
Not ("**the system is bootstrapping**" and ("**the sink is not awake**" or "**one or more nodes are using the bus**"))

$$\psi_{C_3} = \overline{x_b \cdot (\overline{x_s} + x_1 + x_2 + x_3)}$$

Rule of inference: "if A then B" is the same as "(not A) or B"

Q2: Set of States and Transitions (Exam Question in HS2023)



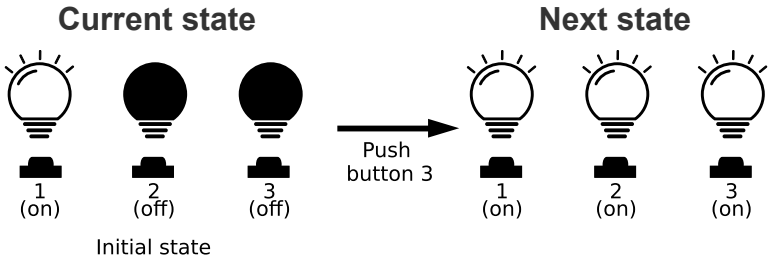
Current state:
$(x, y, z) = (1, 0, 0)$

Next state:
$(x', y', z') = (1, 1, 1)$

Transition:
$(x, y, z, x', y', z') = (1, 0, 0, 1, 1, 1)$

Pushing a button: toggle the light bulb above it and the adjacent light bulbs

Q2: Set of States and Transitions (Exam Question in HS2023)



Current state: $(x, y, z) = (1, 0, 0)$

Next state: $(x', y', z') = (1, 1, 1)$

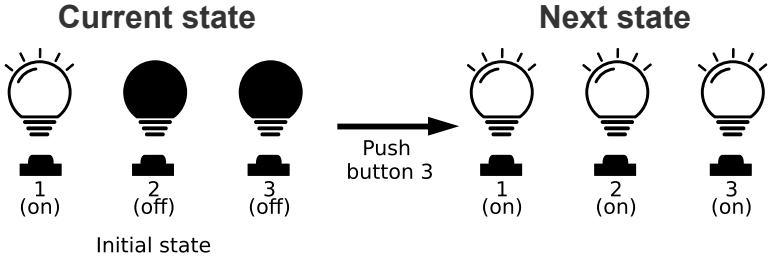
Transition: $(x, y, z, x', y', z') = (1, 0, 0, 1, 1, 1)$

Describe the following property of the current states as a characteristic function:
“if light bulb 1 is in the on-state, then one and exactly one of light bulb 2 or 3 is also in the on-state”

Your turn!

Pushing a button: toggle the light bulb above it and the adjacent light bulbs

Q2: Set of States and Transitions (Exam Question in HS2023)



Current state: $(x, y, z) = (1, 0, 0)$

Next state: $(x', y', z') = (1, 1, 1)$

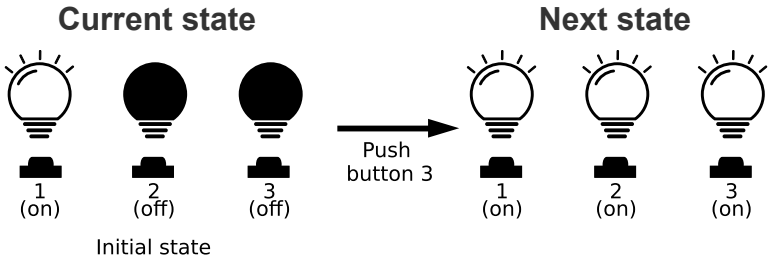
Transition: $(x, y, z, x', y', z') = (1, 0, 0, 1, 1, 1)$

Describe the following property of the current states as a characteristic function:
"if light bulb 1 is in the on-state, then one and exactly one of light bulb 2 or 3 is also in the on-state"

$$\psi_p := x \to (y \cdot \overline{z} + \overline{y} \cdot z)$$

Pushing a button: toggle the light bulb above it and the adjacent light bulbs

Q2: Set of States and Transitions (Exam Question in HS2023)



Current state: $(x, y, z) = (1, 0, 0)$

Next state: $(x', y', z') = (1, 1, 1)$

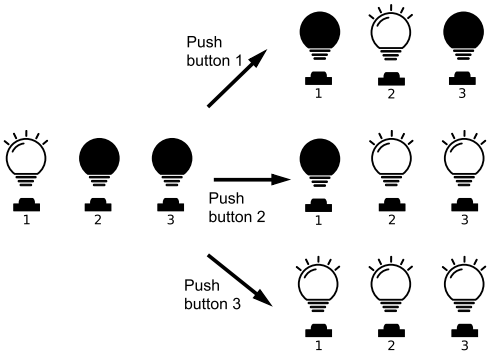
Transition: $(x, y, z, x', y', z') = (1, 0, 0, 1, 1, 1)$

Characteristic function of **the set of transitions** starting from the initial state.

Your turn!

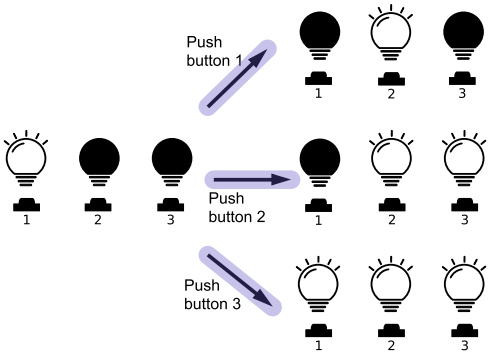
Pushing a button: toggle the light bulb above it and the adjacent light bulbs

Q2: Set of States and Transitions (Exam Question in HS2023)



Pushing a button: toggle the light bulb above it and the adjacent light bulbs

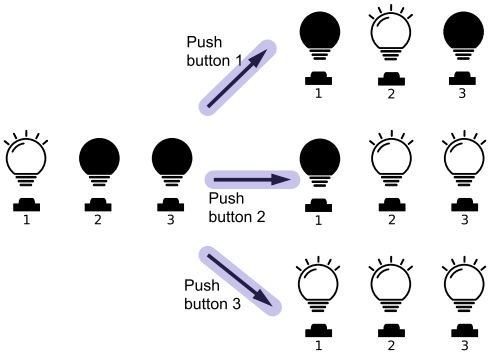
Q2: Set of States and Transitions (Exam Question in HS2023)



We describe the set of transitions: the three arrows starting from the initial state.

Pushing a button: toggle the light bulb above it and the adjacent light bulbs

Q2: Set of States and Transitions (Exam Question in HS2023)



We describe the set of transitions: the three arrows starting from the initial state.

3 transitions

$(x, y, z, x', y', z') =$
$(1, 0, 0, 1, 1, 1)$ or $(1, 0, 0, 0, 1, 0)$ or
$(1, 0, 0, 0, 1, 1)$

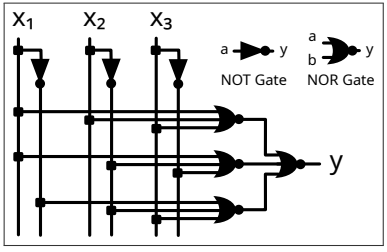
$$x \cdot \overline{y} \cdot \overline{z} \cdot (\overline{x'} \cdot y' \cdot \overline{z'} + \overline{x'} \cdot y' \cdot z' + x' \cdot y' \cdot z').$$

Pushing a button: toggle the light bulb above it and the adjacent light bulbs

Q3.1: Combinational Equivalence Checking (CEC) Using BDDs

We want to check that the circuit implements the functionality of F_1.

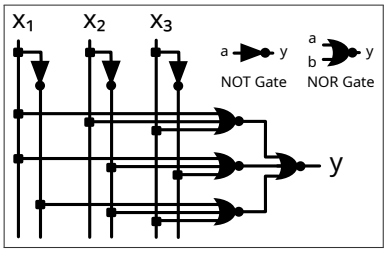
$$F_1 := x_1\overline{x_2} + x_1x_3 + \overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}.$$



Q3.1: Combinational Equivalence Checking (CEC) Using BDDs

We want to check that the circuit implements the functionality of F_1.

$$F_1 := x_1\overline{x_2} + x_1x_3 + \overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}.$$

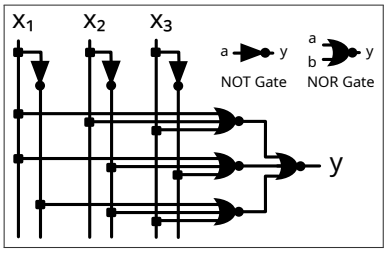


Can we just test the two circuits with all possible inputs?
Not scalable for large designs!

Q3.1: Combinational Equivalence Checking (CEC) Using BDDs

We want to check that the circuit implements the functionality of F_1.

$$F_1 := x_1\overline{x_2} + x_1x_3 + \overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}.$$



Can we just test the two circuits with all possible inputs?
Not scalable for large designs!

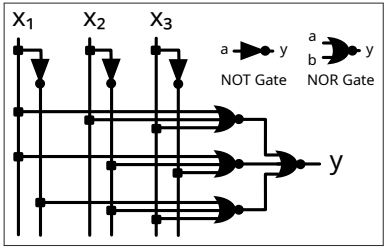
For a 64-bit input circuit, it takes about **290 years to verify the design**, assuming we have a 2 GHz CPU that checks one input in every clock cycle.

The ROBDD-based method has **linear complexity** (in the best case) with respect to the number of variables.

Q3.1: Combinational Equivalence Checking (CEC) Using BDDs

We want to check that the circuit implements the functionality of F_1.

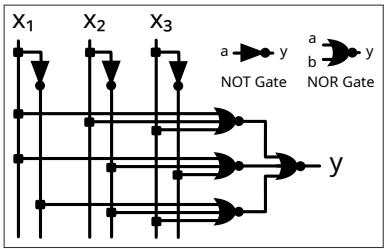
$$F_1 := x_1\overline{x_2} + x_1x_3 + \overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}.$$



Q3.1: Combinational Equivalence Checking (CEC) Using BDDs

We want to check that the circuit implements the functionality of F_1.

$$F_1 := x_1\overline{x_2} + x_1x_3 + \overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}.$$



(a) Derive the Boolean function of the circuit diagram:

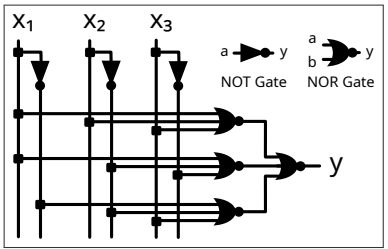
$$\overline{\overline{x_1 + x_2 + x_3} + \overline{x_1 + \overline{x_2} + \overline{x_3}} + \overline{\overline{x_1} + \overline{x_2} + x_3}}$$

Your turn!

Q3.1: Combinational Equivalence Checking (CEC) Using BDDs

We want to check that the circuit implements the functionality of F_1.

$$F_1 := x_1\overline{x_2} + x_1x_3 + \overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}.$$



(a) Derive the Boolean function of the circuit diagram:

$$\overline{\overline{x_1 + x_2 + x_3} + \overline{x_1 + \overline{x_2} + \overline{x_3}} + \overline{\overline{x_1} + \overline{x_2} + x_3}}$$

(b) Draw and compare the ROBDDs of F_1 and F_2 (evaluation order $x_1 < x_2 < x_3$). First: the ROBDD of F_1.

Your turn!

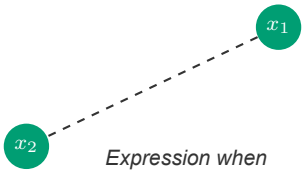
Q3.1: Combinational Equivalence Checking (CEC) Using BDDs



Root node: same as F_1

$$x_1\overline{x_2} + x_1x_3 + \overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}$$

Q3.1: Combinational Equivalence Checking (CEC) Using BDDs



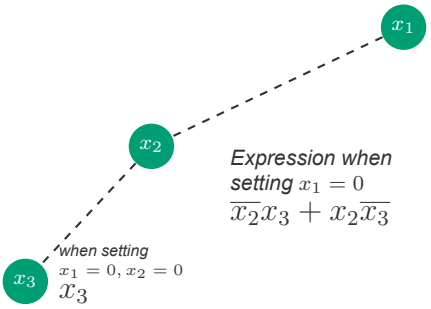
Root node: same as F_1

$$x_1\overline{x_2} + x_1x_3 + \overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}$$

Expression when setting $x_1 = 0$

$$\overline{x_2}x_3 + x_2\overline{x_3}$$

Q3.1: Combinational Equivalence Checking (CEC) Using BDDs



Root node: same as F_1

$$x_1\overline{x_2} + x_1x_3 + \overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}$$

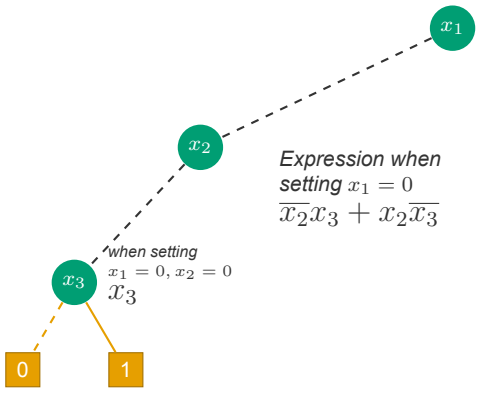
Expression when setting $x_1 = 0$

$$\overline{x_2}x_3 + x_2\overline{x_3}$$

when setting $x_1 = 0, x_2 = 0$

$$x_3$$

Q3.1: Combinational Equivalence Checking (CEC) Using BDDs



Root node: same as F_1

$$x_1\overline{x_2} + x_1x_3 + \overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}$$

Expression when setting $x_1 = 0$

$$\overline{x_2}x_3 + x_2\overline{x_3}$$

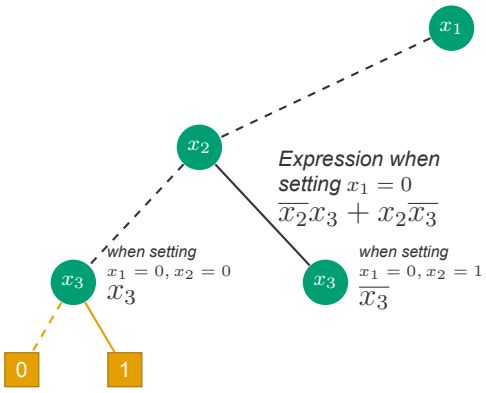
when setting $x_1 = 0, x_2 = 0$

$$x_3$$

Q3.1: Combinational Equivalence Checking (CEC) Using BDDs

Root node: same as F_1

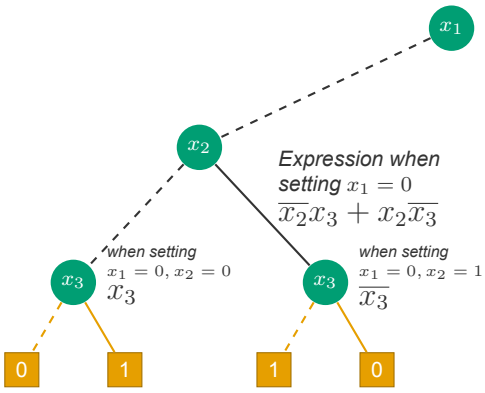
$$x_1\overline{x_2} + x_1x_3 + \overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}$$



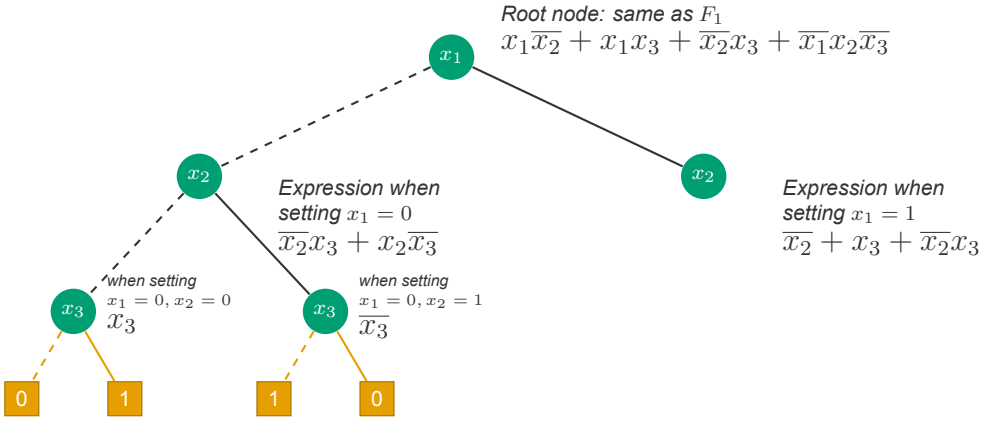
Q3.1: Combinational Equivalence Checking (CEC) Using BDDs

Root node: same as F_1

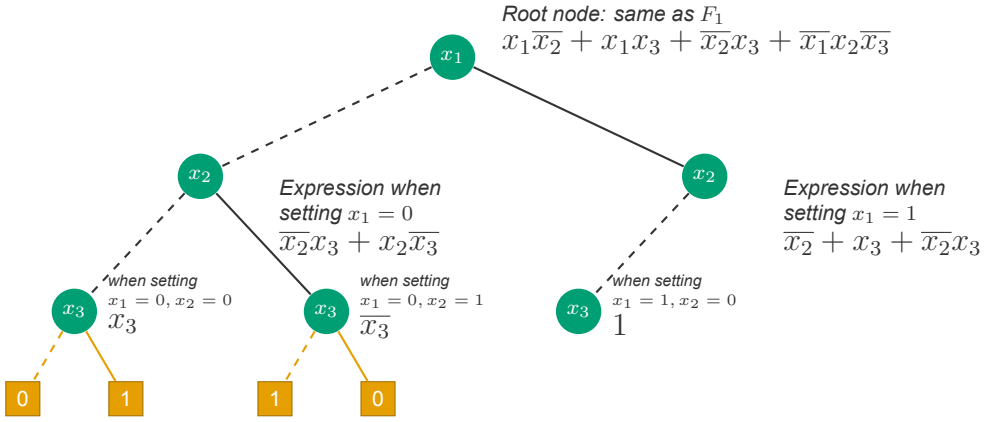
$$x_1\overline{x_2} + x_1x_3 + \overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}$$



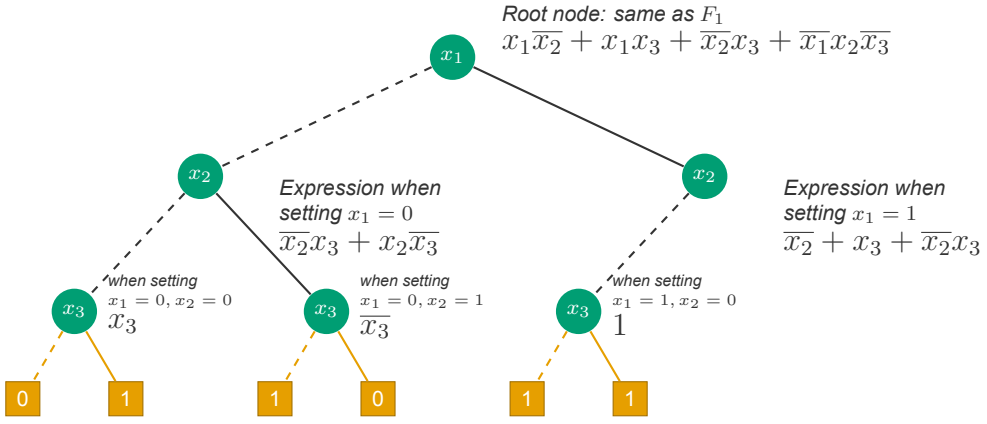
Q3.1: Combinational Equivalence Checking (CEC) Using BDDs



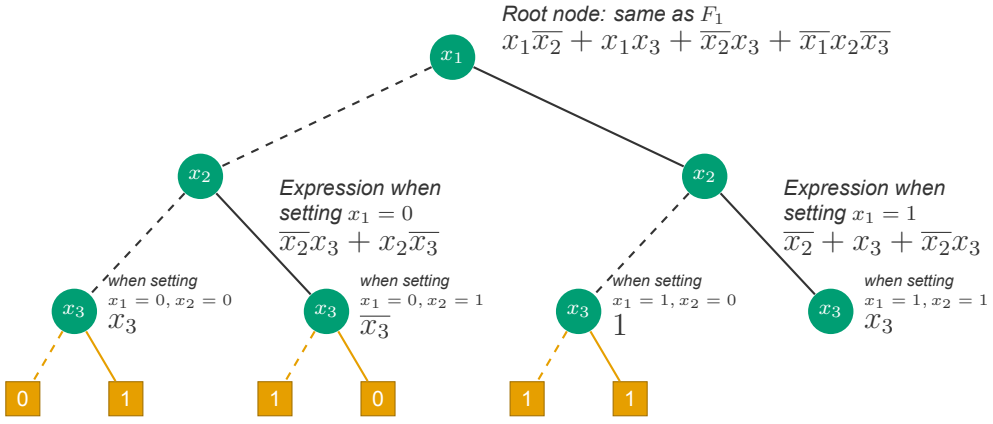
Q3.1: Combinational Equivalence Checking (CEC) Using BDDs



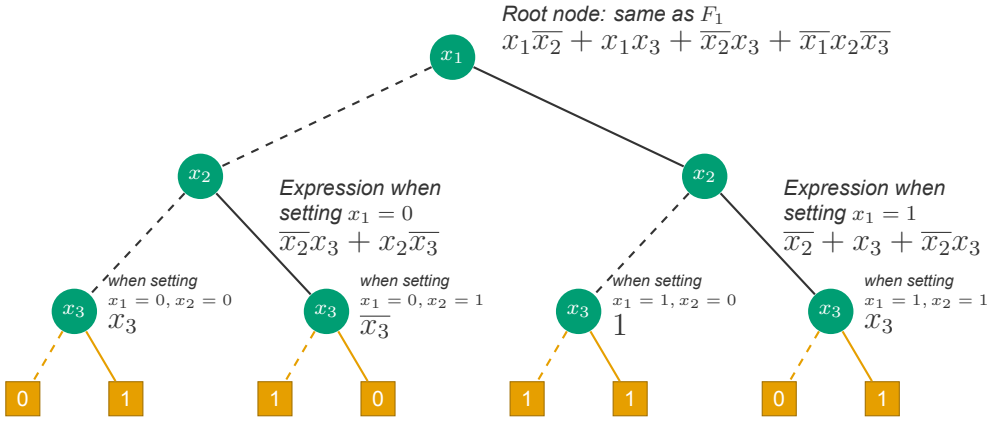
Q3.1: Combinational Equivalence Checking (CEC) Using BDDs



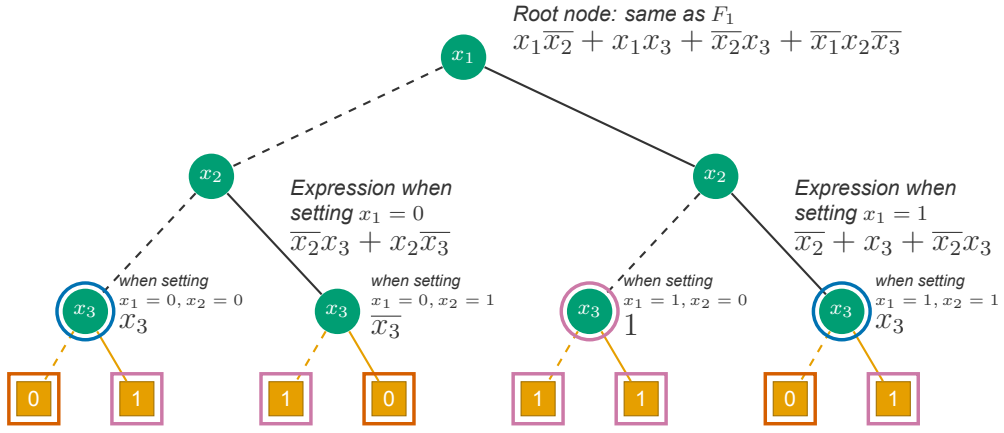
Q3.1: Combinational Equivalence Checking (CEC) Using BDDs



Q3.1: Combinational Equivalence Checking (CEC) Using BDDs

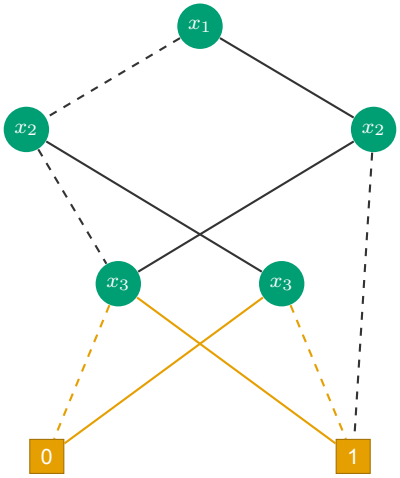


Q3.1: Combinational Equivalence Checking (CEC) Using BDDs



- Same expression x_3: merge the two nodes
- Expression 1 (both edges to 1): same as leaf 1; all 1-leaves merge into one
- All 0-leaves merge into one

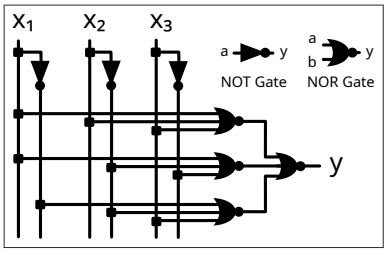
Q3.1: Combinational Equivalence Checking (CEC) Using BDDs



Q3.1: Combinational Equivalence Checking (CEC) Using BDDs

We want to check that the circuit implements the functionality of F_1.

$$F_1 := x_1\overline{x_2} + x_1x_3 + \overline{x_2}x_3 + \overline{x_1}x_2\overline{x_3}.$$



(a) Derive the Boolean function of the circuit diagram:

$$\overline{\overline{x_1 + x_2 + x_3} + \overline{x_1 + \overline{x_2} + \overline{x_3}} + \overline{\overline{x_1} + \overline{x_2} + x_3}}$$

(b) Draw and compare the ROBDDs of F_1 and F_2 (evaluation order $x_1 < x_2 < x_3$). Next: the ROBDD of F_2. Are the two functions identical?

Q3.2: ROBDDs with Different Orderings

$$G(x_1, x_2, y_1, y_2) := (x_1 \leftrightarrow y_1) \cdot (x_2 \leftrightarrow y_2)$$

(a) Compute the Boole-Shannon expansion of G, with respect to the following ordering:

$$x_1 < x_2 < y_1 < y_2$$

(b) Draw the ROBDD of G with the same ordering $x_1 < x_2 < y_1 < y_2$.

(c) Draw the ROBDD of G with the new ordering $x_1 < y_1 < x_2 < y_2$.
Which of the two orderings gives fewer decision nodes?

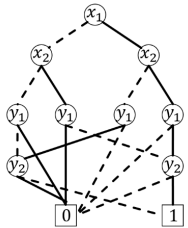
Q3.2: ROBDDs with Different Orderings

$$G(x_1, x_2, y_1, y_2) := (x_1 \leftrightarrow y_1) \cdot (x_2 \leftrightarrow y_2)$$

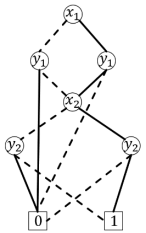
(a) Boole-Shannon expansion of G:

$$G = x_1\Big\{x_2\big[y_1(y_2) + \overline{y_1}(0)\big] + \overline{x_2}\big[y_1(\overline{y_2}) + \overline{y_1}(0)\big]\Big\}$$
$$+ \overline{x_1}\Big\{x_2\big[y_1(0) + \overline{y_1}(y_2)\big] + \overline{x_2}\big[y_1(0) + \overline{y_1}(\overline{y_2})\big]\Big\}$$

(b), (c) ROBDDs:



(b) $x_1 < x_2 < y_1 < y_2$



(c) $x_1 < y_1 < x_2 < y_2$