LTL$_f$ to Symbolic DFA

Shufang Zhu

Post-doc of WhiteMech group

LTL_f and DFA

$$\varphi ::= \mathit{true} \mid \mathit{false} \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid X\varphi \mid \varphi_1 U \varphi_2$$

- Every LTL_f formula has a corresponding DFA.
- LTL_f reasoning to DFA reasoning.
  - Computations on graph, intuitive and simpler.
- Explicit DFA: **2EXP** number of states.
  - $|\varphi| = 10$, $\#\text{states} = 2^{2^{10}}$.
- Main obstacle of LTL_f reasoning: explicit DFA is too large.

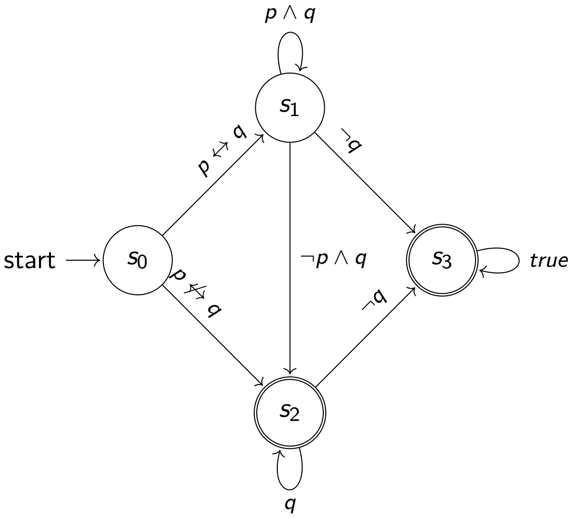
Why Symbolic Techniques?

- Graph, network, nodes and edges.
- Planning, RL, Games on graph etc.
- Symbolic encoding: same information, less resource.
- Scalability? Go symbolic.

Outline

- Explicit DFA.
- From explicit to symbolic.
  - Monolithic encoding.
  - Partitioned encoding.
- Symbolic representation BDD.

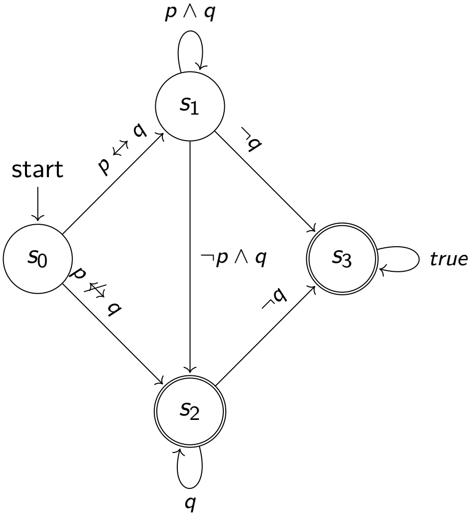
Explicit DFA



Informations Contained in DFA

- $\mathcal{D} = \{\mathcal{P}, \mathcal{S}, s_0, \delta, \mathcal{F}\}$
  - $\mathcal{P}$ a set of propositions.
  - $\mathcal{S}$ a set of states.
  - s_0 initial state.
  - $\delta : \mathcal{S} \times 2^{\mathcal{P}} \to \mathcal{S}$ transition function.
  - $\mathcal{F}$ a set of accepting states.

Explicit DFA Example



- $\mathcal{P} = \{p, q\}$.
- $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$.
- s_0 initial state.
- $\delta : \mathcal{S} \times 2^{\mathcal{P}} \to \mathcal{S}$.
  - $\delta(s_1, \neg p \wedge q) = s_2$.
- $\mathcal{F} = \{s_2, s_3\}$.

From Explicit DFA to Symbolic DFA

- $\mathcal{D} = \{\mathcal{P}, \mathcal{S}, s_0, \delta, \mathcal{F}\}$
- State space $\mathcal{S}$.
- Answer queries:
  - Which state is the initial state?
  - Is s an accepting states?
  - ...
- Perform computations:
  - Current state s, transition condition σ.
  - Return the successor state.

From Explicit DFA to Symbolic DFA

- Explicit DFA: $\mathcal{D} = \{\mathcal{P}, \mathcal{S}, s_0, \delta, \mathcal{F}\}$
- Symbolic DFA: Maintain the information as in the explicit DFA
  - State space $\mathcal{S}$.
  - Answer queries: initial state? accepting state?
  - Perform computations: get successor state.

State Space

- $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$.
- Binary state encoding $\mathcal{Z} = \{z_0, z_1\}$.

State	Binary Code.
s_0	00
s_1	01
s_2	10
s_3	11

- **EXP** less number of variables

Initial and Accepting States

- $s_0 = \{s_0, s_1, s_2, s_3\}$.
- $s_0(00)$ initial state.
- $\mathcal{F} = \{s_2(10), s_3(11)\}$.
- $\mathcal{F}$ is an explicit set, not succinct enough.

Symbolic Encoding of A Set of States

- Queries related to the set of accepting states.
  - $\mathcal{F}$: Is s an accepting state? Answers: *Yes*, *No*.
  - Boolean Function f: Is assignment Z a model of f? Answers: *True*, *False*.
  - Encode $\mathcal{F}$ as a Boolean function f, more succinct than an explicit set.

Symbolic Encoding of A Set of States

- Every state $s \in \mathcal{S}$ as a conjunction on the values of each bit.
  - This conjunction refers only to the specific state.

State	Binary Code	Conjunction
s_0	00	$\neg z_0 \wedge \neg z_1$
s_1	01	$\neg z_0 \wedge z_1$
s_2	10	$z_0 \wedge \neg z_1$
s_3	11	$z_0 \wedge z_1$

Symbolic Encoding of A Set of States

- A set of states is a disjunction on the conjunctions.
  - This disjunction refers only to the specific set of states.
- Initial state $I = \underbrace{\neg z_0 \wedge \neg z_1}_{s_0(00)}$.
- Accepting states $f = \underbrace{(\neg z_0 \wedge z_1)}_{s_1(01)} \vee \underbrace{(z_0 \wedge z_1)}_{s_3(11)}$.

Symbolic Transition Function

- State variables $\mathcal{Z} = \{z_0, z_1\}$.
- Transition function rewritten as $\underbrace{\eta = 2^{\mathcal{Z}} \times 2^{\mathcal{P}} \to 2^{\mathcal{Z}}}_{\delta = \mathcal{S} \times 2^{\mathcal{P}} \to \mathcal{S}}$.
- Boolean function only returns *True* or *False*.
- How to use Boolean function to encode transition function?
  - Monolithic encoding.
  - Partitioned encoding.

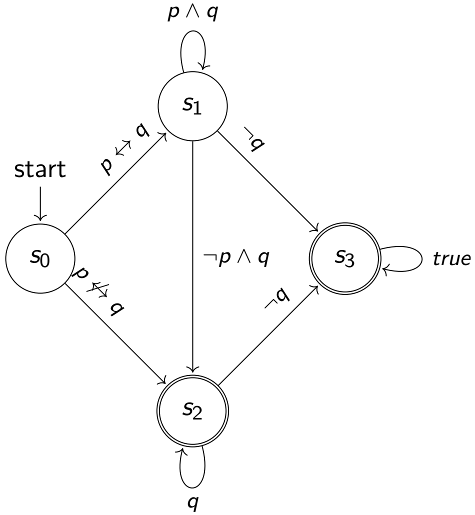
Monolithic Transition Function Encoding

- What does a transition function do?
  - **Given:** Current state Z, transition condition σ.
    Return: Successor state Z'.
- How about the following?
  - **Given:** Current state Z, transition condition σ, successor state Z'.
    Return: Is $(Z, \sigma) \rightarrow Z'$ a correct transition? *Yes*, *No*.

Monolithic Transition Function Encoding

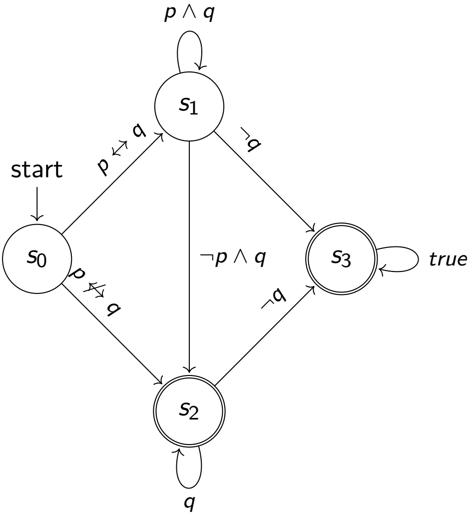
- **Given:** Current state Z, transition condition σ, successor state Z'.
  Return: Is $(Z, \sigma) \to Z'$ a correct transit? *Yes*, *No*.
- Introduce a so-called prime state variables $\mathcal{Z}' = \{z' \mid z \in \mathcal{Z}\}$
- Rewritten as $\eta = 2^{\mathcal{Z}} \times 2^{\mathcal{P}} \times 2^{\mathcal{Z}'} \to \{0, 1\}$.
  - Only returns *True* for correct transitions $(Z, \sigma) \to Z'$.

Monolithic Transition Function Encoding



- $\mathcal{Z} = \{z_0, z_1\}$, $\mathcal{Z}' = \{z'_0, z'_1\}$
- Transition as conjunction
- $(s_1, \neg q) \rightarrow s_3$
- $\underbrace{\neg z_0 \wedge z_1}_{s_1} \wedge \neg q \wedge \underbrace{z'_0 \wedge z'_1}_{s_3}$

Monolithic Transition Function Encoding



- For every transition, consider as a conjunction of $Z \wedge \sigma \wedge Z'$.
- η : disjunction of conjunctions.
    - $\eta = \bigvee(Z \wedge \sigma \wedge Z')$.

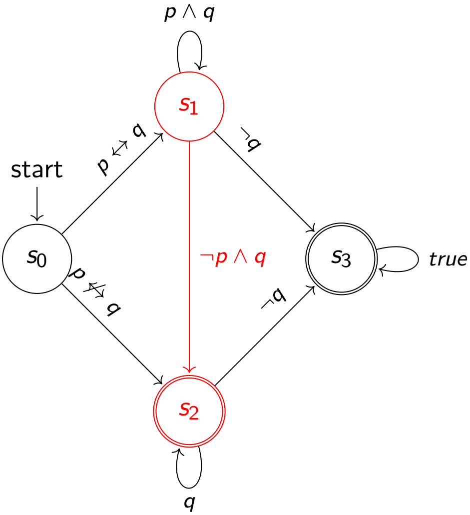
Monolithic DFA Encoding

- $\mathcal{G} = \{\mathcal{P}, \mathcal{Z}, I, \eta, f\}$.
  - $\mathcal{P}$: a set of propositions.
  - $\mathcal{Z}$: a set of state variables, every model on $\mathcal{Z}$ is a state.
  - I: a Boolean function denoting the initial state.
  - $\eta = 2^{\mathcal{Z}} \times 2^{\mathcal{P}} \times 2^{\mathcal{Z}'} \to \{0, 1\}$: a Boolean function encoding the transition function.
  - f: a Boolean function encoding the set of accepting state.

Partitioned Transition Function Encoding

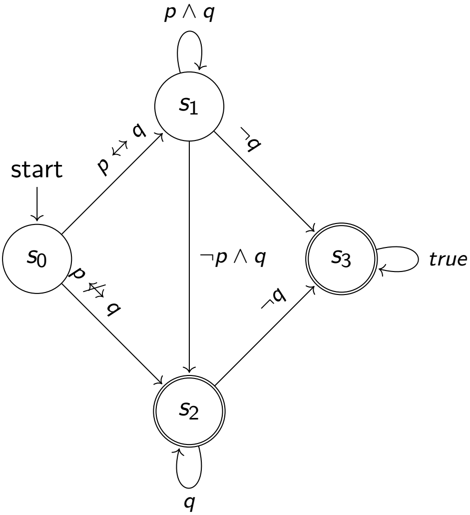
- What does a transition function do?
  - **Given:** Current state Z, transition condition σ.
    Return: Successor state Z'.
- Every state Z, conjunction of a sequence of values of each $z \in \mathcal{Z}$.
  - State s_1, 01, $Z = \neg z_0 \wedge \neg z_1$.
- The value of each $z \in \mathcal{Z}$ is $\{0, 1\}$.
- $\eta = \{\eta_{z_0}, \eta_{z_1}, \ldots\}$, $|\eta| = |\mathcal{Z}|$.
  - $\eta_{z_i} = 2^{\mathcal{Z}} \times 2^{\mathcal{P}} \to \{0, 1\}$.

Partitioned Transition Function Encoding



- $\mathcal{Z} = \{z_0, z_1\}$.
- $(\neg z_0 \wedge z_1, \neg p \wedge q) \rightarrow \underbrace{z_0}_{\eta_{z_0}} \wedge \underbrace{\neg z_1}_{\eta_{z_1}}$.
- η_{z_0} returns $True(1)$.
  η_{z_1} returns $False(0)$.

Partitioned Transition Function Encoding



- Only $\mathcal{Z}$, no prime variables!
- $\eta = \{\eta_z \mid z \in \mathcal{Z}\}$.
- η_{z_i} : disjunction on conjunctions $Z \wedge \sigma$.
  - Transition $(Z, \sigma) \rightarrow Z'$.
  - z_i is *true* in Z'

Partitioned DFA Encoding

- $\mathcal{G} = \{\mathcal{P}, \mathcal{Z}, I, \eta, f\}$.
  - $\mathcal{P}$: a set of propositions.
  - $\mathcal{Z}$: a set of state variables, every model on $\mathcal{Z}$ is a state.
  - I: a Boolean function denoting the initial state.
  - $\eta = 2^{\mathcal{Z}} \times 2^{\mathcal{P}} \to 2^{\mathcal{Z}}$: a sequence of Boolean functions encoding the transition function.
  - f: a Boolean function encoding the set of accepting state.

Monolithic and Partitioned Encoding

	Explicit	Monolithic	Partitioned
Props	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$
St. space	$\lvert\mathcal{S}\rvert = n$	$\lvert\mathcal{Z}\rvert = \log_n$	$\lvert\mathcal{Z}\rvert = \log_n$
Init. state	s_0	$I = \neg z_0 \wedge \neg z_1$	$I = \neg z_0 \wedge \neg z_1$
Acc. states	$\mathcal{F}$	$f = \bigvee \wedge$	$f = \bigvee \wedge$
Trans.	$\mathcal{S} \times 2^{\mathcal{P}} \to \mathcal{S}$	$2^{\mathcal{Z}} \times 2^{\mathcal{P}} \times 2^{\mathcal{Z}'} \to \{0,1\}$	$2^{\mathcal{Z}} \times 2^{\mathcal{P}} \to 2^{\mathcal{Z}}$

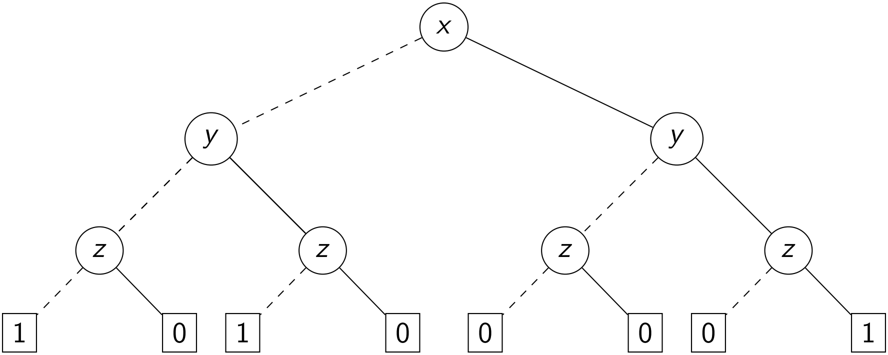
- Encoded as Boolean function.
- How to represent a Boolean function in a more compact way?
- Binary Decision Diagram (BDD).

Why BDD?

- They can be made canonical.
- They can be very compact for many applications.
- Many applications can be converted to sequences of Boolean operations on BDD.

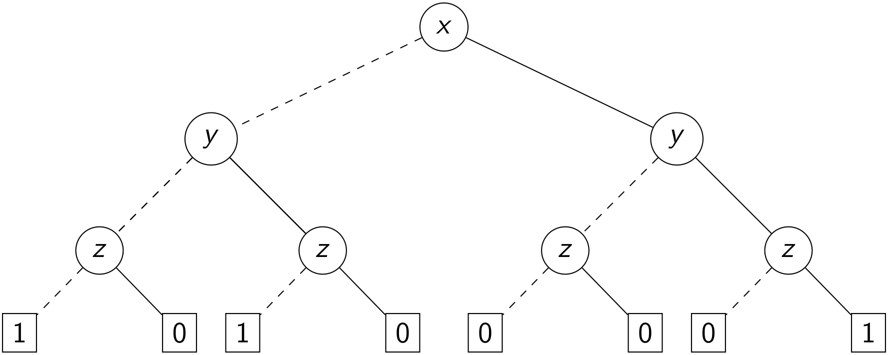
Binary Decision Diagram: Example

- Directed graph representing Boolean functions.
- non-terminal node (circle), terminal node (square).



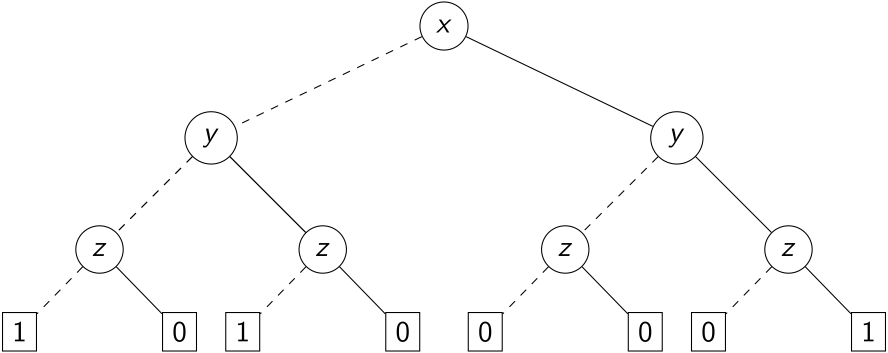
Binary Decision Diagram: Example

- non-terminal node (circle), marked with variables x, y, z.
- terminal node (square), marked with values $0, 1$.



Binary Decision Diagram: Example

- solid line: *high*(v), variable assigned as *true*.
- dashed line: *low*(v), variable assigned as *false*.

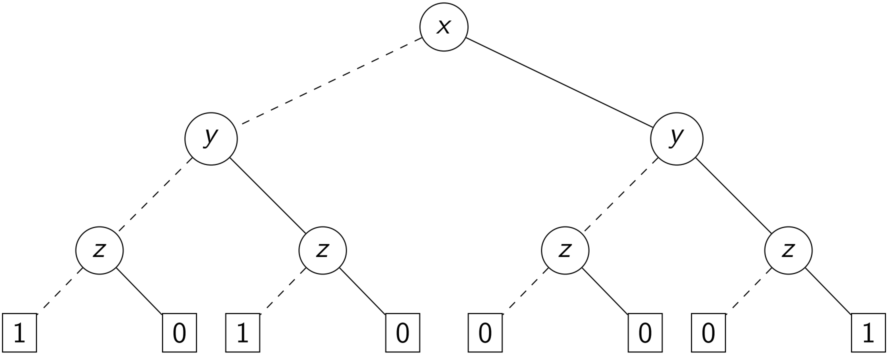


Boolean Function As BDD

- $f = (x \wedge y \wedge z) \vee (\neg x \wedge \neg y)$.
- **Given:** A model $\neg x, y, z$.
  Return: $\mathit{False}(0)$.
- **Given:** A model x, y, z.
  Return: $\mathit{True}(1)$.
- The BDD should do the same job.

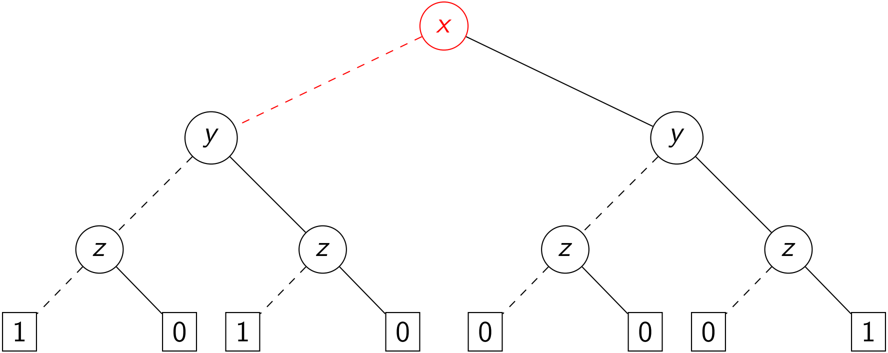
Boolean Function As BDD: Example

- $f = (x \wedge y \wedge z) \vee (\neg x \wedge \neg y)$.
- Model $\neg x, y, z$.



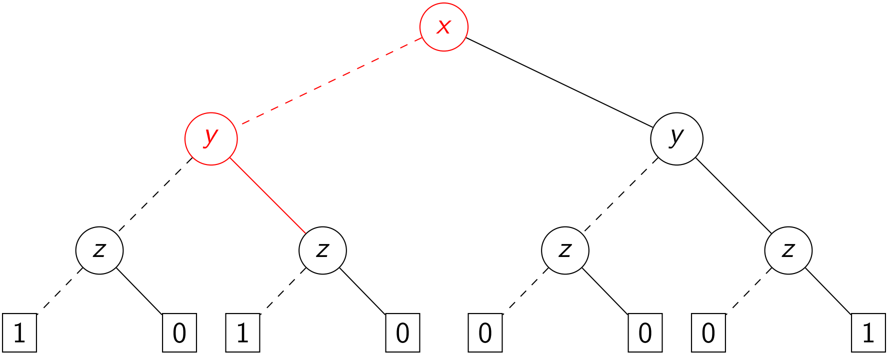
Boolean Function As BDD: Example

- $f = (x \wedge y \wedge z) \vee (\neg x \wedge \neg y)$.
- Model $\neg x, y, z$.



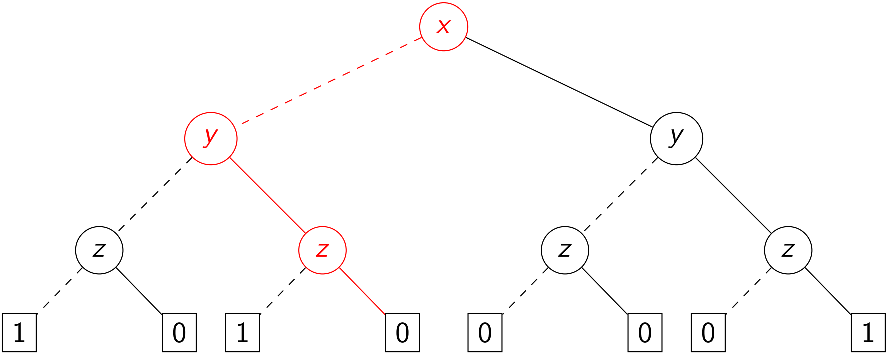
Boolean Function As BDD: Example

- $f = (x \wedge y \wedge z) \vee (\neg x \wedge \neg y)$.
- Model $\neg x, y, z$.



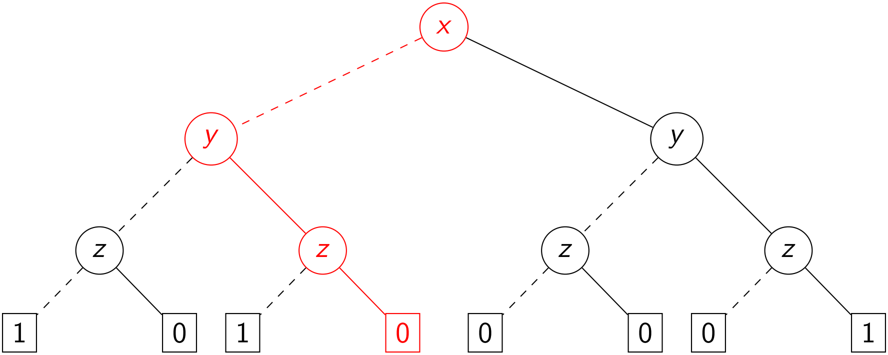
Boolean Function As BDD: Example

- $f = (x \wedge y \wedge z) \vee (\neg x \wedge \neg y)$.
- Model $\neg x, y, z$.



Boolean Function As BDD: Example

- $f = (x \wedge y \wedge z) \vee (\neg x \wedge \neg y)$.
- Model $\neg x, y, z$.

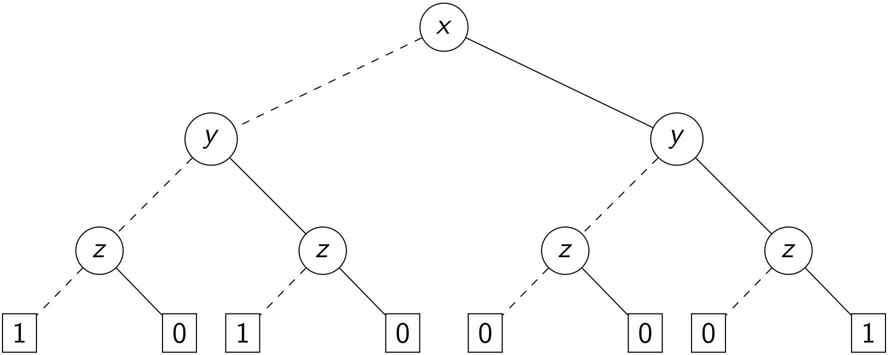


Boolean Function As BDD

- BDD is able to encode a Boolean function.
- BDD: Compact representation.
  - **Elimination rule**.
  - **Isomorphism rule**.

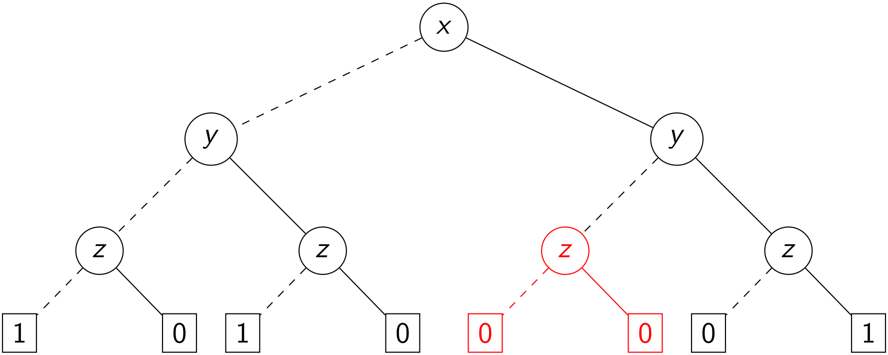
Elimination Rule

- **Elimination rule:** If low(v) = high(v) = w, eliminate v and redirect all incoming edges to v to node w.



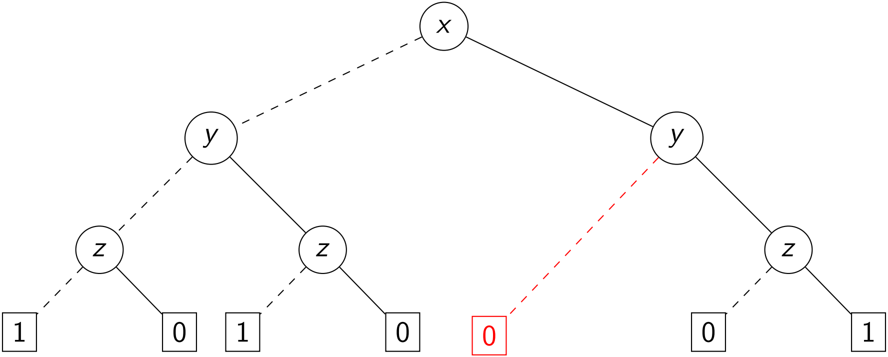
Elimination Rule

- **Elimination rule:** If $\text{low}(v) = \text{high}(v) = w$, eliminate v and redirect all incoming edges to v to node w.



Elimination Rule

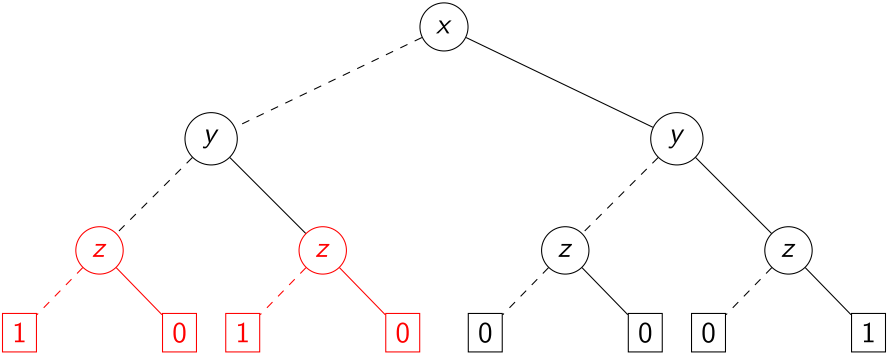
- **Elimination rule:** If low(v) = high(v) = w, eliminate v and redirect all incoming edges to v to node w.



Isomorphism Rule

Isomorphism rule:

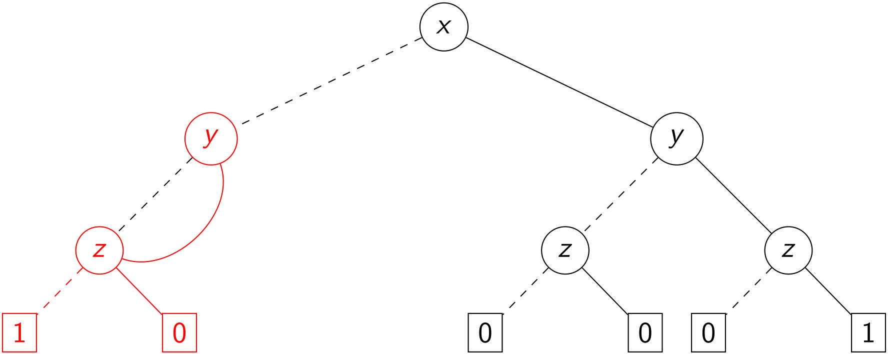
- If $v \neq w$ are roots of isomorphic subtrees, remove v, and redirect all incoming edges to v to node w.
- Combine all 0/1-leaves, redirect all incoming edges.



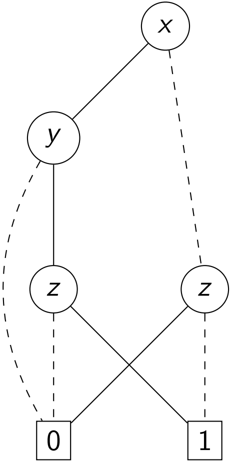
Isomorphism Rule

Isomorphism rule:

- If $v \neq w$ are roots of isomorphic subtrees, remove v, and redirect all incoming edges to v to node w.
- Combine all 0/1-leaves, redirect all incoming edges.

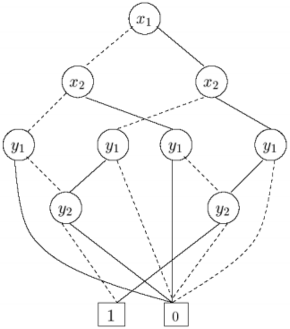


Binary Decision Diagram: Reduced



Binary Decision Diagram: Variable Ordering

- BDD size: #nodes.
- BDD size highly depends on the variable ordering.
- $f = (x_1 \wedge x_2 \wedge y_1 \wedge y_2) \vee (\neg x_1 \wedge x_2 \wedge \neg y_1 \wedge y_2) \vee (x_1 \wedge \neg x_2 \wedge y_1 \wedge \neg y_2) \vee (\neg x_1 \wedge \neg x_2 \wedge \neg y_1 \wedge \neg y_2)$.



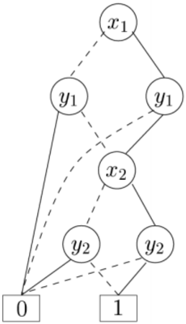


Figure: Different variable ordering

Binary Decision Diagram: Canonicity

- Canonicity: variable ordering.
- BDDs are canonical with a fixed variable ordering.
- Canonicity checking takes constant time.
- Example:
  - **Given:** Boolean functions f and g.
  - **Answer:** Whether $f \equiv g$?
  - **How:** Construct B_f and B_g, $B_f \equiv B_g$, constant time.

BDD Libraries

- Buddy, CUDD, etc.
- Rich API functions for manipulating BDDs, elimination rules and isomorphism rules are applied automatically.
- Logic operations on BDDs, conjunction, disjunction, quantifier elimination etc.

Symbolic DFA Represented In BDDs

	Explicit	Monolithic	Partitioned
Props	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$
St. space	$\|\mathcal{S}\| = n$	$\|\mathcal{Z}\| = \log_n$	$\|\mathcal{Z}\| = \log_n$
Init. state	s_0	$B_I = \neg z_0 \wedge \neg z_1$	$B_I = \neg z_0 \wedge \neg z_1$
Acc. states	$\mathcal{F}$	$B_f = \bigvee \wedge$	$B_f = \bigvee \wedge$
Trans.	$\mathcal{S} \times 2^{\mathcal{P}} \to \mathcal{S}$	BDD B $2^{\mathcal{Z}} \times 2^{\mathcal{P}} \times 2^{\mathcal{Z}'} \to \{0,1\}$	$\|\mathcal{Z}\|$ BDDs $B_1, B_2, \ldots$ $2^{\mathcal{Z}} \times 2^{\mathcal{P}} \to 2^{\mathcal{Z}}$

Questions to Answer

- Which encoding leads to better performance?
  - Symbolic DFA of LTL_f.
  - Smaller BDDs.
  - Games on DFA, faster solving.