Game-Theoretic Approach to Temporal Synthesis

Symbolic Techniques

Antonio Di Stasio Giuseppe Perelli **Shufang Zhu**

2nd European Summer School on Artificial Intelligence
Athens (Greece) 15-19 July 2024

- Synthesis and automata-theoretic approaches to synthesis
- Reduction to games on graphs (automata)
  - Reachability game, safety game, GR(1) game etc.
  - Game solving is linear or poly, wrt the size of the game graph

- The game graph size?
  - LTL_f synthesis, explicit DFA **2EXP** number of states
  - $|\varphi| = 10$, #states $= 2^{2^{10}}$
- Symbolic techniques, compact representation and reasoning

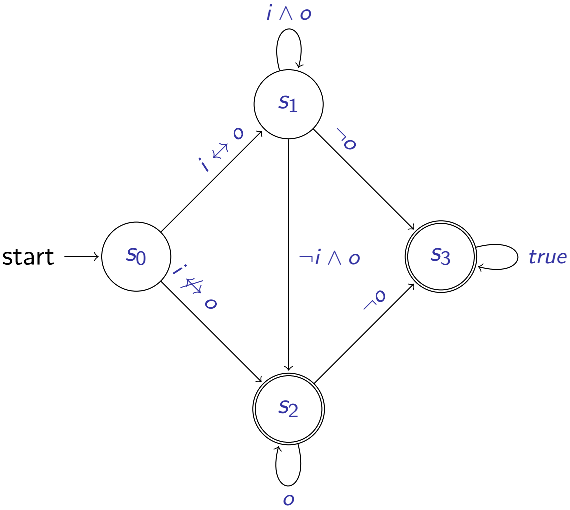
Outline

- Symbolic DFA representation
  - Monolithic representation
  - Partitioned representation
- Symbolic synthesis techniques
  - Symbolic LTL_f synthesis, reachability game [1]
- Binary Decision Diagram (BDD)

[1] Zhu et al.: Symbolic LTL_f Synthesis.

Explicit DFA as a tuple $\mathcal{D} = \{\mathcal{P}, \mathcal{S}, s_0, \delta, \mathcal{F}\}$

- $\mathcal{P}$ 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



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

- $\mathcal{D} = \{\mathcal{P}, \mathcal{S}, s_0, \delta, \mathcal{F}\}$
- State space $\mathcal{S}$
- Answer queries:
  - Which state is the initial state?
  - Is s an accepting states?
  - Consider current state s and transition label α, what is the successor state?
  - ...

- 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? successor state?

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

State	Interpretation Z
s_0	$z_0 = 0, z_1 = 0$
s_1	$z_0 = 0, z_1 = 1$
s_2	$z_0 = 1, z_1 = 0$
s_3	$z_0 = 1, z_1 = 1$

- **EXP** less number of variables

- $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$

- $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$
- Initial state s_0

- $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$
- Initial state $(z_0 = 0, z_1 = 0)$

- $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$
- Initial state $(z_0 = 0, z_1 = 0)$
- Accepting states $\mathcal{F} = \{s_2, s_3\}$

- $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$
- Initial state $(z_0 = 0, z_1 = 0)$
- Accepting states $\mathcal{F} = \{(z_0 = 1, z_1 = 0), (z_0 = 1, z_1 = 1)\}$

- $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$
- Initial state $(z_0 = 0, z_1 = 0)$
- Accepting states $\mathcal{F} = \{(z_0 = 1, z_1 = 0), (z_0 = 1, z_1 = 1)\}$
- $\mathcal{F}$ is an explicit set, not succinct enough

- Queries related to the set of accepting states
  - $\mathcal{F}$: Is s an accepting state? Answers: *Yes*, *No*
  - Boolean formula f over $\mathcal{Z}$: Is interpretation $Z \in 2^{\mathcal{Z}}$ a model of f? Answers: $true$, $false$
  - Encode $\mathcal{F}$ as a Boolean formula f over $\mathcal{Z}$, more succinct than an explicit set

Symbolic Encoding of A Set of States

- Every state $s \in \mathcal{S}$ as a Boolean formula **only** satisfied by the corresponding interpretation $Z \in 2^{\mathcal{Z}}$
  - Through conjunction, refers to a certain state

State	Interpretation Z	Boolean formula
s_0	$z_0 = 0, z_1 = 0$	$\neg z_0 \wedge \neg z_1$
s_1	$z_0 = 0, z_1 = 1$	$\neg z_0 \wedge z_1$
s_2	$z_0 = 1, z_1 = 0$	$z_0 \wedge \neg z_1$
s_3	$z_0 = 1, z_1 = 1$	$z_0 \wedge z_1$

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

- State variables $\mathcal{Z} = \{z_0, z_1\}$
- Transition function $\delta(s, \alpha) = s'$
- Boolean formula η only evaluates to *true* or *false*
- How to use Boolean formula to encode transition function?
  - Monolithic representation
  - Partitioned representation

– What does a transition function do?

– What does a transition function do?

Given: Current state s, transition condition α

– What does a transition function do?

Given: Current state s, transition condition α

Return: Successor state s'

– What does a transition function do?

Given: Current state s, transition condition α

Return: Successor state s'

– What about the following?

- What does a transition function do?

  Given: Current state s, transition condition α

  Return: Successor state s'

- What about the following?

  Given: Interpretation Z, transition condition α, interpretation Z'

- What does a transition function do?

  Given: Current state s, transition condition α

  Return: Successor state s'

- What about the following?

  Given: Interpretation Z, transition condition α, interpretation Z'

  Return: Is (Z, α, Z') a correct transition? *Yes*, *No*

Given: Interpretation Z, transition condition α, interpretation Z'

Return: Is (Z, α, Z') a correct transition? *Yes*, *No*

Given: Interpretation Z, transition condition α, interpretation Z'

Return: Is (Z, α, Z') a correct transition? *Yes*, *No*

– Introduce prime variables $\mathcal{Z}' = \{z' \mid z \in \mathcal{Z}\}$ to differentiate current and successor

Given: Interpretation Z, transition condition α, interpretation Z'

Return: Is (Z, α, Z') a correct transition? *Yes*, *No*

- Introduce prime variables $\mathcal{Z}' = \{z' \mid z \in \mathcal{Z}\}$ to differentiate current and successor
- Transition function as Boolean formula η over $\mathcal{Z} \cup \mathcal{P} \cup \mathcal{Z}'$

Given: Interpretation Z, transition condition α, interpretation Z'

Return: Is (Z, α, Z') a correct transition? *Yes*, *No*

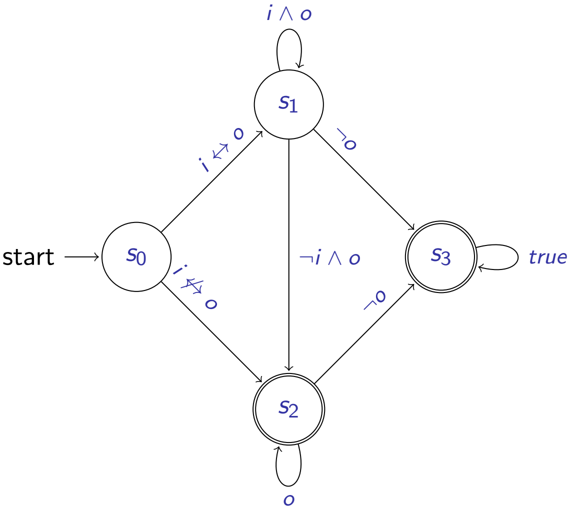
- Introduce prime variables $\mathcal{Z}' = \{z' \mid z \in \mathcal{Z}\}$ to differentiate current and successor
- Transition function as Boolean formula η over $\mathcal{Z} \cup \mathcal{P} \cup \mathcal{Z}'$
  - Evaluates as *true* **only** for correct transitions

Each transition as a conjunction of the corresponding interpretation

- $\delta(s_1, \neg o) = s_3$
- $\underbrace{\neg z_0 \land z_1}_{s_1} \land \neg o \land \underbrace{z_0' \land z_1'}_{s_3}$

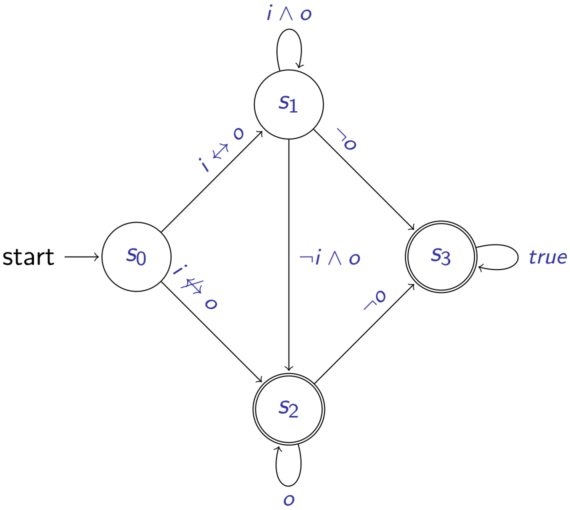
η : disjunction of conjunctions

$$\eta = \bigvee (Z \wedge \alpha \wedge Z')$$

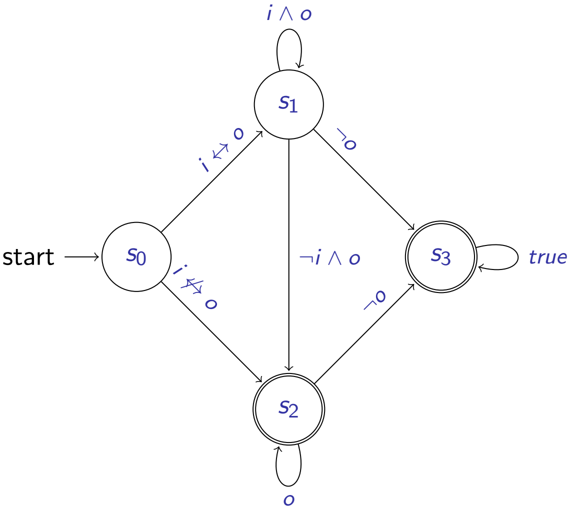


Symbolic $\mathcal{D}_m = (\mathcal{P}, \mathcal{Z}, \mathcal{Z}', \iota, \eta, f)$

Example of Monolithic Representation

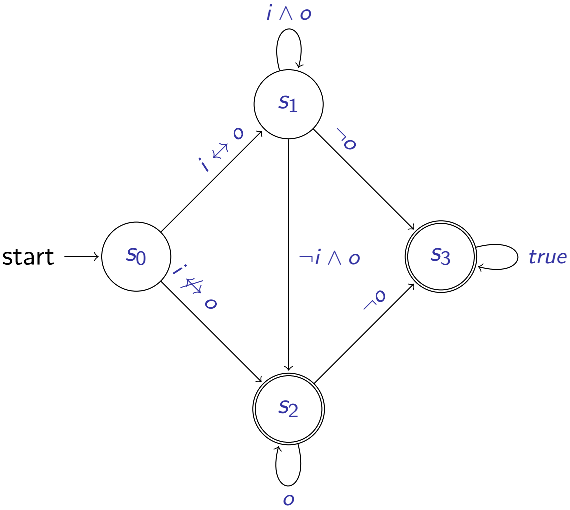


- $\mathcal{Z} = \{z_0, z_1\}$
- $\mathcal{Z}' = \{z'_0, z'_1\}$



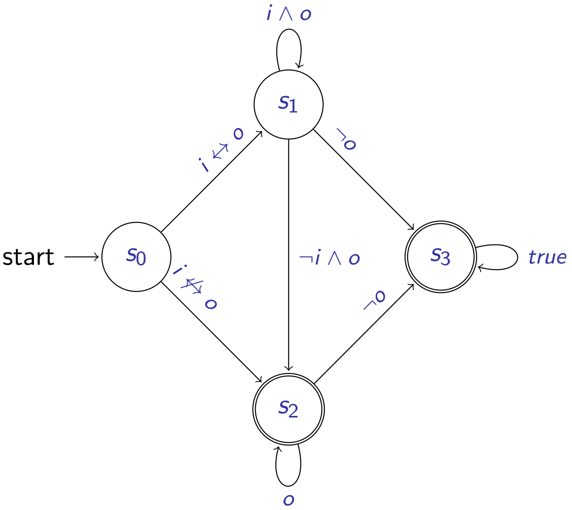
– Initial state

$$\iota = \underbrace{\neg z_0 \wedge \neg z_1}_{s_0(00)}$$



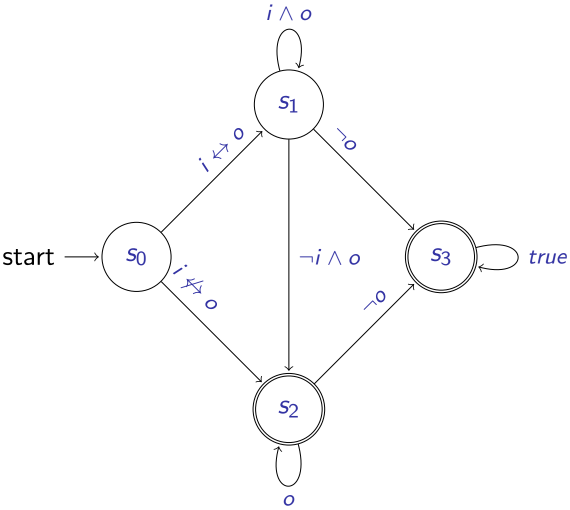
– Accepting states

$$f = \underbrace{(\neg z_0 \wedge z_1)}_{s_1(01)} \vee \underbrace{(z_0 \wedge z_1)}_{s_3(11)}$$



Each transition as a conjunction

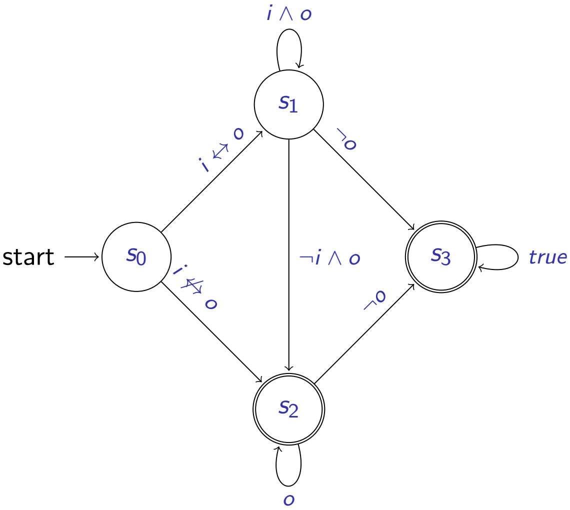
- $(s_1, \neg o) \rightarrow s_3$
- $\underbrace{\neg z_0 \wedge z_1}_{s_1} \wedge \neg o \wedge \underbrace{z_0' \wedge z_1'}_{s_3}$



Each transition as a conjunction

- $(s_1, i \wedge o) \rightarrow s_1$
- $\underbrace{\neg z_0 \wedge z_1}_{s_1} \wedge i \wedge o \wedge \underbrace{\neg z_0' \wedge z_1'}_{s_1}$

Example of Monolithic Representation



η : disjunction of conjunctions

$$\eta = \bigvee(Z \wedge \alpha \wedge Z')$$

$$\mathcal{D}_m = \{\mathcal{P}, \mathcal{Z}, \mathcal{Z}', \iota, \eta, f\}$$

- $\mathcal{P}$ a set of propositions
- $\mathcal{Z}$ a set of state variables, $\mathcal{Z}'$ prime state variables
- ι Boolean formula over $\mathcal{Z}$ denoting the initial state
- η Boolean formula over $\mathcal{Z} \cup \mathcal{P} \cup \mathcal{Z}'$ representing the transition function
- f Boolean formula over $\mathcal{Z}$ representing the set of accepting states

– Monolithic representation

- Monolithic representation
  - Straightforward, primed variables

- Monolithic representation
  - Straightforward, primed variables
- Partitioned representation

- Monolithic representation
  - Straightforward, primed variables
- Partitioned representation
  - Model Checking

- Monolithic representation
  - Straightforward, primed variables
- Partitioned representation
  - Model Checking
  - LTL_f synthesis

$$\mathcal{D}_p = \{\mathcal{P}, \mathcal{Z}, \iota, \eta, f\}$$

- $\mathcal{P}$ a set of propositions
- $\mathcal{Z}$ a set of state variables
- ι Boolean formula over $\mathcal{Z}$ denoting the initial state
- η **transition function in a partitioned way**
- f Boolean formula over $\mathcal{Z}$ representing the set of accepting states

Given: Current state s, transition condition σ

Return: Successor state s'

- Every state s as interpretation over $\mathcal{Z}$

Given: Current state s, transition condition σ

Return: Successor state s'

- Every state s as interpretation over $\mathcal{Z}$
  - State s_1 corresponds to $z_0 = 0, z_1 = 1$

Given: Current state s, transition condition σ

Return: Successor state s'

- Every state s as interpretation over $\mathcal{Z}$
  - State s_1 corresponds to $z_0 = 0, z_1 = 1$
- Partition the computation of successor state

Partition the computation of successor state s'

- compute the value of $z \in \mathcal{Z}$ one after another

$\eta = \{\eta_{z_0}, \eta_{z_1}, \ldots\}$, $|\eta| = |\mathcal{Z}|$

Partition the computation of successor state s'

- compute the value of $z \in \mathcal{Z}$ one after another

$\eta = \{\eta_{z_0}, \eta_{z_1}, \ldots\}$, $|\eta| = |\mathcal{Z}|$

- η_{z_i} Boolean formula over $\mathcal{Z} \cup \mathcal{P}$

Partition the computation of successor state s'

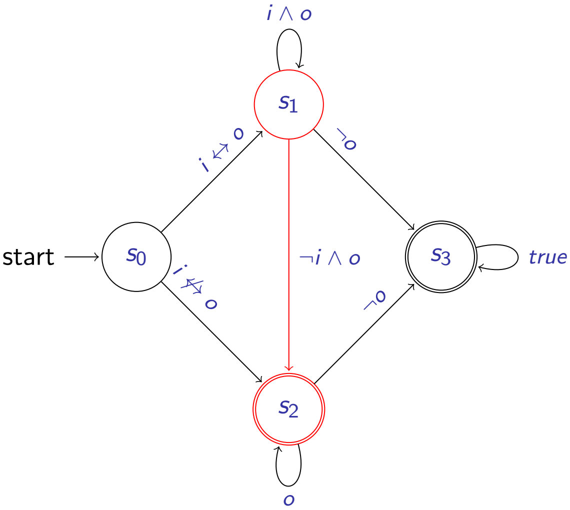
- compute the value of $z \in \mathcal{Z}$ one after another

$\eta = \{\eta_{z_0}, \eta_{z_1}, \ldots\}$, $|\eta| = |\mathcal{Z}|$

- η_{z_i} Boolean formula over $\mathcal{Z} \cup \mathcal{P}$
- $\eta_{z_i}(Z, \sigma)$ evaluates to *true* iff $z_i = 1$ in the corresponding successor state of outgoing edge (Z, σ)

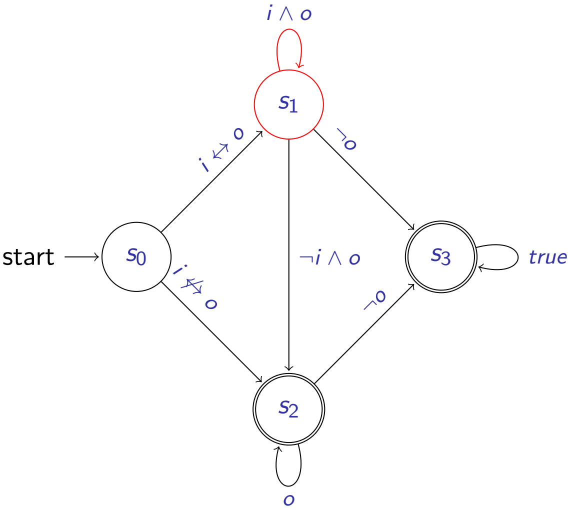
$\eta = \{\eta_{z_0}, \eta_{z_1}, \ldots\}$, $|\eta| = |\mathcal{Z}|$

- η_{z_i} : disjunction of conjunctions
  - every conjunction, an outgoing edge (Z, σ), which makes $z_i = 1$ in the corresponding successor state

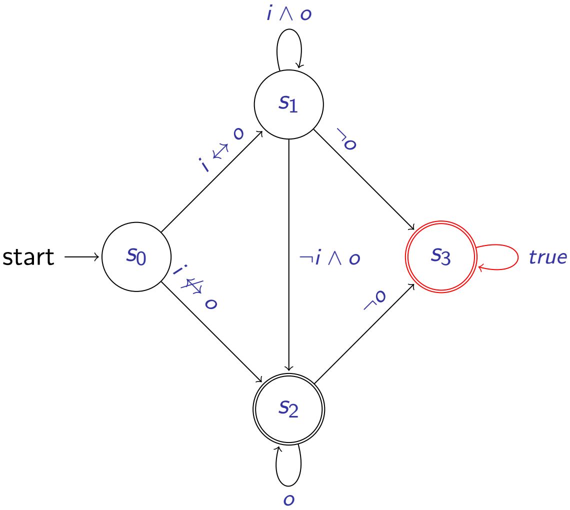


$\mathcal{Z} = \{z_0, z_1\}$

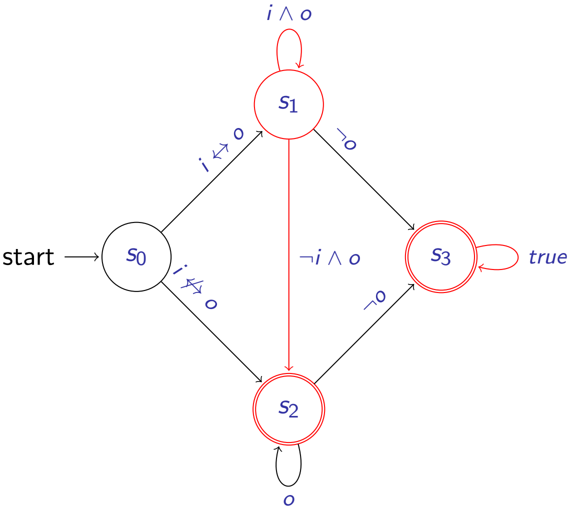
- $(\underbrace{\neg z_0, z_1}_{s_1(01)}, \neg p, q) \rightarrow \underbrace{z_0, \neg z_1}_{s_2(10)}$
- $\eta_{z_0}(\neg z_0, z_1, \neg i, o)$ evaluates to *true*
  $\eta_{z_1}(\neg z_0, z_1, \neg i, o)$ evaluates to *false*



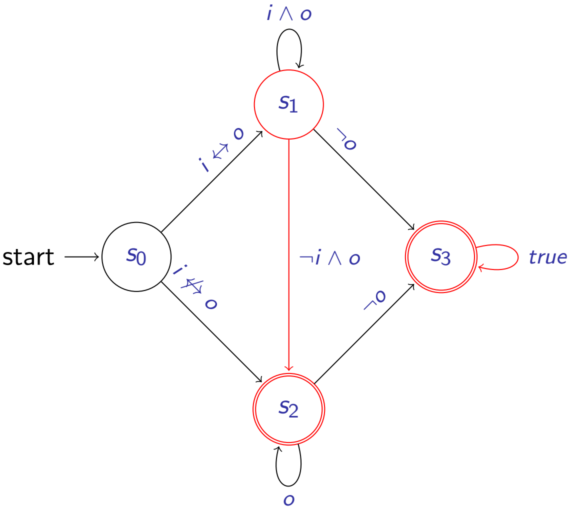
- $\underbrace{(\neg z_0, z_1}_{s_1(01)}, i, o) \rightarrow \underbrace{\neg z_0, z_1}_{s_1(01)}$
- $\eta_{z_0}(\neg z_0, z_1, i, o)$ evaluates to *false*
  $\eta_{z_1}(\neg z_0, z_1, i, o)$ evaluates to *true*



- $(\underbrace{z_0, z_1}_{s_3(11)}, true) \rightarrow \underbrace{z_0, z_1}_{s_3(11)}$
- $\eta_{z_0}(z_0, z_1, true)$ evaluates to $true$
  $\eta_{z_1}(z_0, z_1, true)$ evaluates to $true$

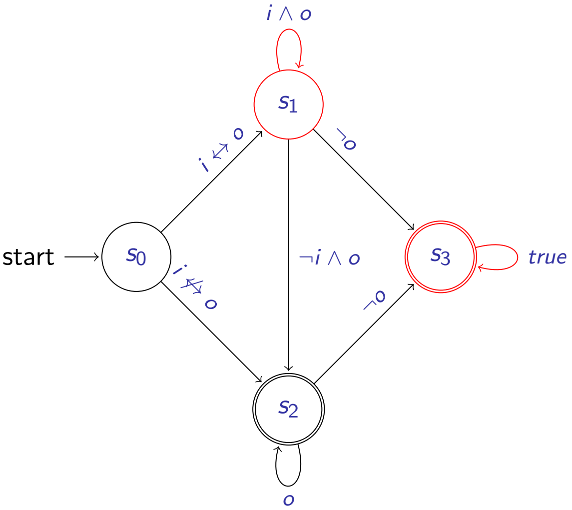


- $\eta_{z_0}(\neg z_0, z_1, \neg i, o)$ evaluates to *true*
  $\eta_{z_1}(\neg z_0, z_1, \neg i, o)$ evaluates to *false*
- $\eta_{z_0}(\neg z_0, z_1, i, o)$ evaluates to *false*
  $\eta_{z_1}(\neg z_0, z_1, i, o)$ evaluates to *true*
- $\eta_{z_0}(z_0, z_1, true)$ evaluates to *true*
  $\eta_{z_1}(z_0, z_1, true)$ evaluates to *true*
- ...



- $\eta_{z_0}(\neg z_0, z_1, \neg i, o)$ evaluates to *true*
- $\eta_{z_0}(z_0, z_1, true)$ evaluates to *true*
- ...

$\eta_{z_0} =$
$(\neg z_0 \wedge z_1 \wedge \neg i \wedge o) \vee (z_0 \wedge z_1 \wedge true) \vee \ldots$



- $\eta_{z_1}(\neg z_0, z_1, i, o)$ evaluates to *true*
- $\eta_{z_1}(z_0, z_1, true)$ evaluates to *true*
- ...

$$\eta_{z_1} = (\neg z_0 \wedge z_1 \wedge i \wedge o) \vee (z_0 \wedge z_1 \wedge true) \vee \ldots$$

$\mathcal{D}_p = \{\mathcal{P}, \mathcal{Z}, \iota, \eta, f\}$

- $\mathcal{P}$ a set of propositions
- $\mathcal{Z}$ a set of state variables
- ι Boolean formula over $\mathcal{Z}$ denoting the initial state
- $\eta = \{\eta_z \mid z \in \mathcal{Z}\}$ a sequence of Boolean formulas over $\mathcal{Z} \cup \mathcal{P}$ encoding the transition function
- f Boolean formula over $\mathcal{Z}$ representing the set of accepting states

	Explicit	Monolithic	Partitioned
Props	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$
States	$\lvert\mathcal{S}\rvert = n$	$\lvert\mathcal{Z}\rvert = \lvert\mathcal{Z}'\rvert = \log_n$	$\lvert\mathcal{Z}\rvert = \log_n$
Init.	s_0	$\iota = \neg z_0 \wedge \neg z_1$	$\iota = \neg z_0 \wedge \neg z_1$
Acc.	$\mathcal{F}$	$f = \bigvee \wedge$	$f = \bigvee \wedge$
Transition	$\delta : \mathcal{S} \times 2^{\mathcal{P}} \rightarrow \mathcal{S}$	$\eta(\mathcal{Z} \cup \mathcal{P} \cup \mathcal{Z}')$	$\eta = \{\eta_z(\mathcal{Z} \cup \mathcal{P}) \mid z \in \mathcal{Z}\}$

- Synthesis as two-player games
  - LTL_f synthesis, reachability games
  - Synthesis under LTL specifications, parity games

- Synthesis as two-player games
  - LTL_f synthesis, reachability games
  - Synthesis under LTL specifications, parity games
- Two-player games
  - Fixpoint computation on game arena
  - Symbolic fixpoint computation

LTL_f synthesis

- Reachability game on DFA, agent o and environment i
- Agn: visit accepting states

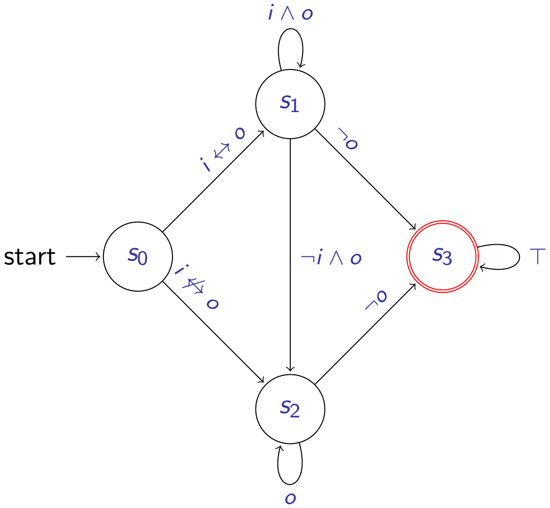
Recap on LTL_f Synthesis

Algorithm 1 Reachability game on DFA $\mathcal{D}_p = (\mathcal{I}, \mathcal{O}, \mathcal{S}, s_0, \delta, \mathcal{F})$

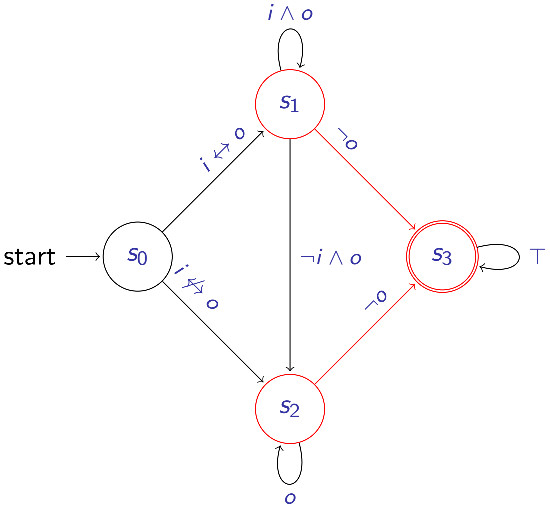
```
Win := 𝓕
while Win ≠ Win ∪ force_ag(Win) do
    Win := Win ∪ force_ag(Win)
end while
return Win
```

$$\mathsf{force}_{ag}(\mathsf{Win}) = \{s \mid \exists O \forall I \delta(s, I \cup O \in \mathsf{Win})\}$$

– O a winning output of state s

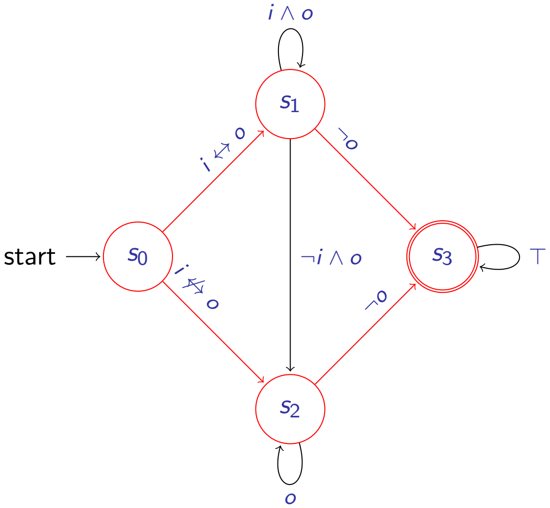


$W_0 = \{s_3\}$, accepting states

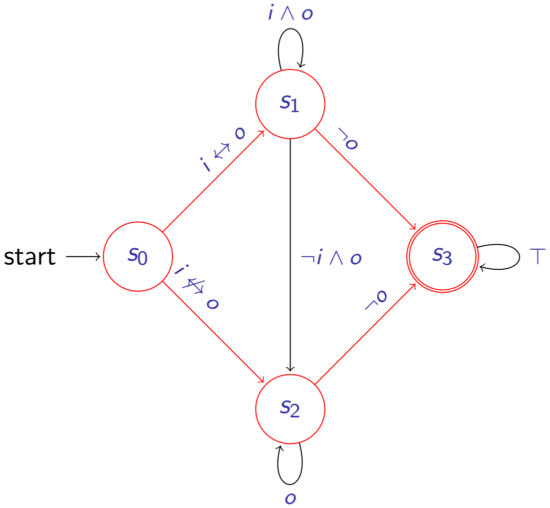


- $W_0 = \{s_3\}$
- There exists o, for every i

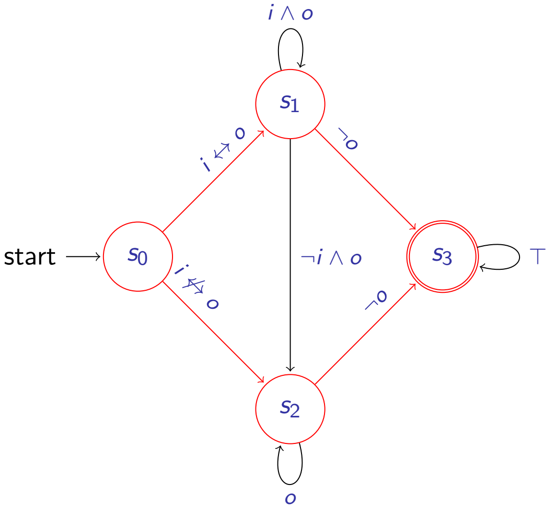
  $W_1 = \{s_3, s_1, s_2\}$



- $W_0 = \{s_3\}$
- $W_1 = \{s_3, s_1, s_2\}$
- $W_2 = \{s_3, s_1, s_2, s_0\}$



- $W_0 = \{s_3\}$
- $W_1 = \{s_3, s_1, s_2\}$
- $W_2 = \{s_3, s_1, s_2, s_0\}$
- $W_3 = \{s_3, s_1, s_2, s_0\}$
- $W_3 = W_2$, fixpoint



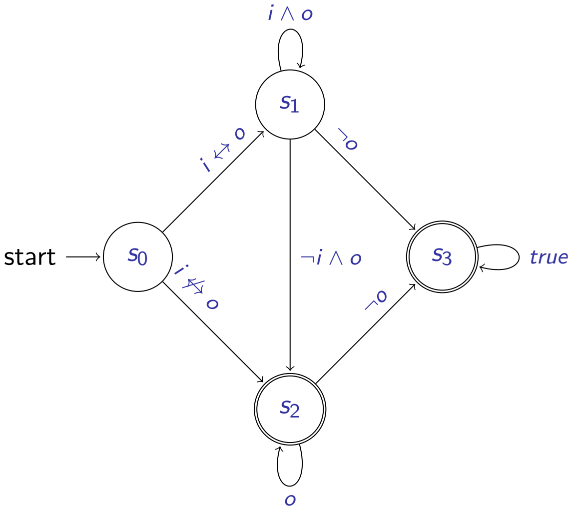
- $s_0 \in W = \{s_3, s_1, s_2, s_0\}$
- Realizable
- Winning strategy as a transducer

Winning strategy as an explicit transducer $\mathcal{T} = (2^{\mathcal{I}}, 2^{\mathcal{O}}, \mathsf{Win}, s_0, \varrho, \omega)$

- $\mathsf{Win} \subseteq \mathcal{S}$ is the set of winning states
- $\omega : \mathsf{Win} \rightarrow 2^{\mathcal{O}}$ is the output function such that $\omega(s)$ is a winning output of s

Winning strategy $\omega : \text{Win} \to 2^{\mathcal{O}}$

- $\omega(s_0) = o$
- $\omega(s_1) = \neg o$



Reachability game on symbolic DFA $\mathcal{D}_p = (\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f)$

- A Boolean formula w over $\mathcal{Z}$ for winning states
- A Boolean formula t over $\mathcal{Z} \cup \mathcal{O}$ for (winning state, winning output) pairs

Reachability game on symbolic DFA $\mathcal{D}_p = (\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f)$

- $w_0 = f$ every accepting state is a winning state
- $t_0 = f$ the agent can do anything (*true*) after reaching accepting states

Reachability game on symbolic DFA $\mathcal{D}_p = (\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f)$

- $t_{i+1} = t_i \vee (\neg w_i \wedge \forall I.w_i(\eta))$
- $w_{i+1} = \exists O.t_{i+1}$

$$t_{i+1} = t_i \vee (\neg w_i \wedge \forall I . w_i(\eta))$$

- (Z, O) satisfies t_i
- Z was not yet a winning state, and for every I we can move from Z to an already-identified winning state

$$w_{i+1} = \exists O. t_{i+1}$$

- Z satisfies w_i
- Z was not yet a winning state, and there exists O such that for every I we can move from Z to an already-identified winning state

Why not the following?

- $w_{i+1} = w_i \vee (\neg w_i \wedge \exists O.\forall I.w_i(\eta))$

Why not the following?

- $w_{i+1} = w_i \vee (\neg w_i \wedge \exists O.\forall I.w_i(\eta))$

Why not the following?

- $w_{i+1} = w_i \vee (\neg w_i \wedge \exists O.\forall I.w_i(\eta))$

Reachability game on symbolic DFA $\mathcal{D}_p = (\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f)$

- $w_{i+1} \equiv w_i$, fixpoint w_∞

Explicit finite-state transducer $\mathcal{T} = (2^{\mathcal{I}}, 2^{\mathcal{O}}, \mathsf{Win}, s_0, \varrho, \omega)$

- $\mathsf{Win} \subseteq \mathcal{S}$ is the set of winning states
- $\omega : \mathsf{Win} \to 2^{\mathcal{O}}$ is the output function such that $\omega(s)$ is a winning output of s

Function $\omega : \mathsf{Win} \to 2^{\mathcal{O}}$

- Input: winning state s
- Output: winning output O of s

Function ω : Win $\rightarrow 2^{\mathcal{O}}$

- Input: winning state s
- Output: winning output O of s

We have Boolean formula t over $\mathcal{Z} \cup \mathcal{O}$

- $(Z \cup O) \models t$ iff Z is a winning state and O is a winning output of Z

A function $\tau : 2^{\mathcal{Z}} \to 2^{\mathcal{O}}$

- Input: winning state Z
- Output: winning output O of Z

Boolean synthesis procedure

Given: two disjoint proposition sets $\mathcal{Z}$, $\mathcal{O}$ of input and output variables, respectively, and a Boolean formula t over $\mathcal{Z} \cup \mathcal{O}$

Return: a function $\tau : 2^{\mathcal{Z}} \to 2^{\mathcal{O}}$

- for every $Z \in 2^{\mathcal{Z}}$, if there exists $O \in 2^{\mathcal{O}}$ such that $Z \cup O \models t$, then $Z \cup \tau(Z) \models t$

t over $\mathcal{Z} \cup \mathcal{O}$ as the input formula to a Boolean synthesis procedure

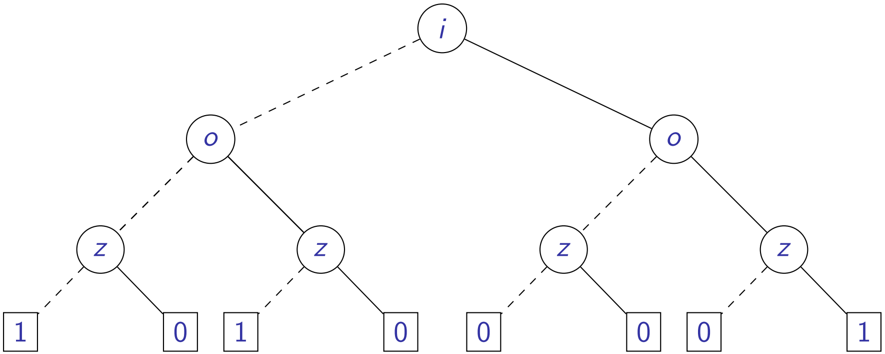
- function $\tau : 2^{\mathcal{Z}} \to 2^{\mathcal{O}}$

- Symbolic least-fixpoint computation
- Abstract winning strategy via Boolean synthesis
- Extend to great-fixpoint, nested-fixpoint computation in different synthesis settings

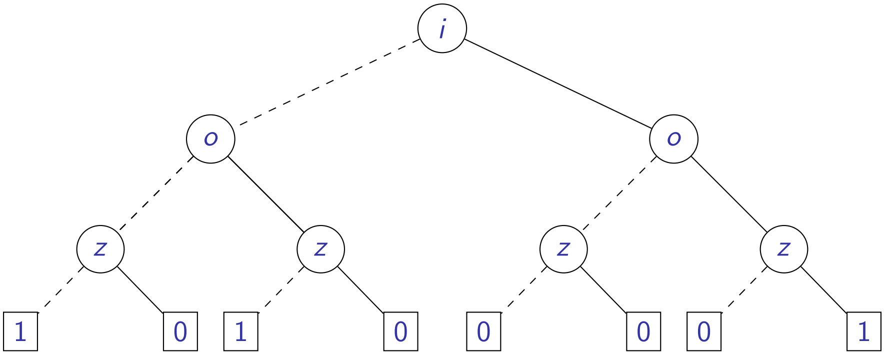
- Symbolic LTL_f synthesis
- Binary Decision Diagrams (BDDs)

- They can be made canonical
- They can be very compact for many applications
- Various computations can be converted to suitable operations on BDD

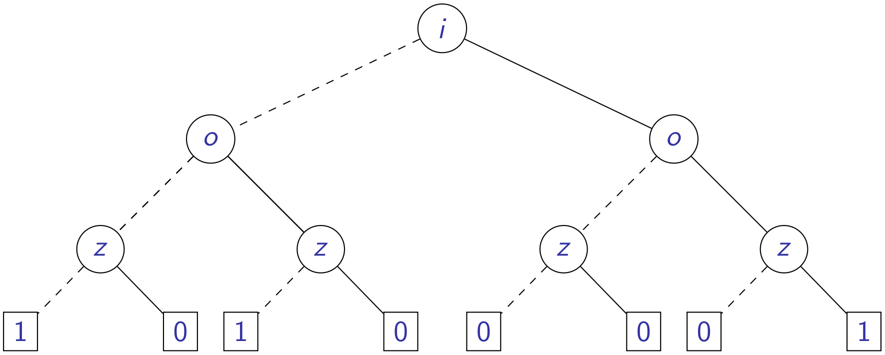
- Directed graph representing Boolean functions
- non-terminal node (circle), terminal node (square)



- non-terminal node (circle), marked with variables i, o, z
- terminal node (square), marked with values $0, 1$



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



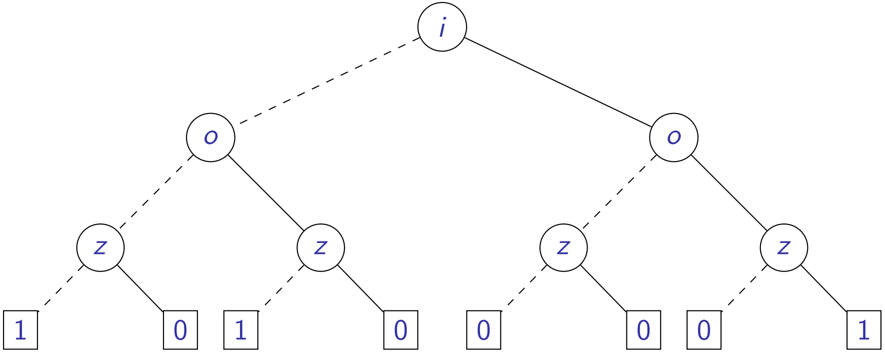
- $f = (i \wedge o \wedge z) \vee (\neg i \wedge \neg o)$

- **Given:** A model $\neg i, o, z$
  Evaluation: $\mathit{false}(0)$

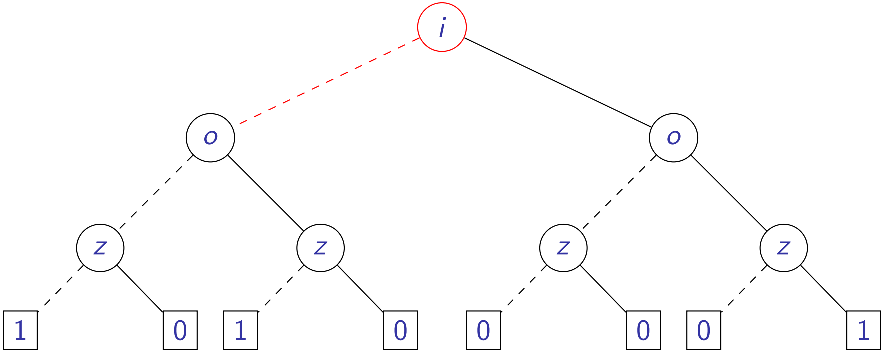
- **Given:** A model i, o, z
  Evaluation: $\mathit{true}(1)$

Example of Boolean Formula Represented in BDDs

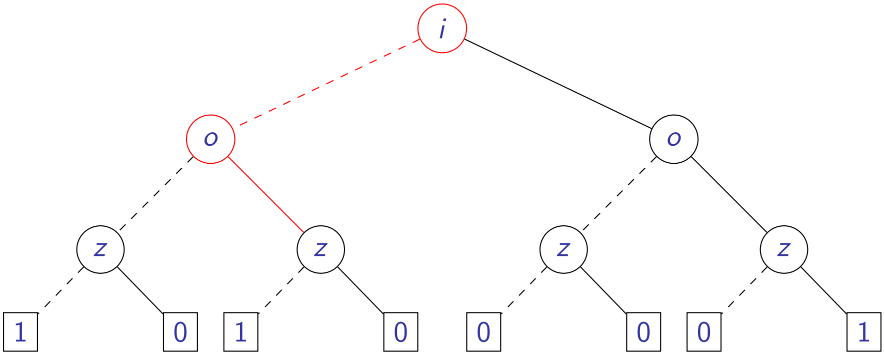
- $f = (i \wedge o \wedge z) \vee (\neg i \wedge \neg o)$
- Interpretation $\neg i, o, z$



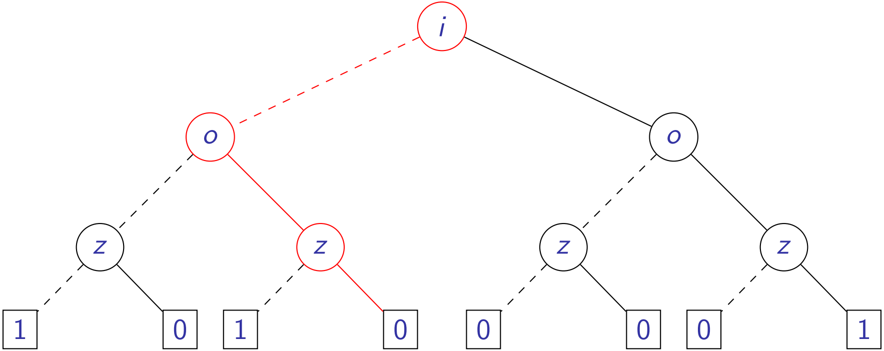
- $f = (i \wedge o \wedge z) \vee (\neg i \wedge \neg o)$
- Interpretation $\neg i, o, z$



- $f = (i \wedge o \wedge z) \vee (\neg i \wedge \neg o)$
- Interpretation $\neg i, o, z$

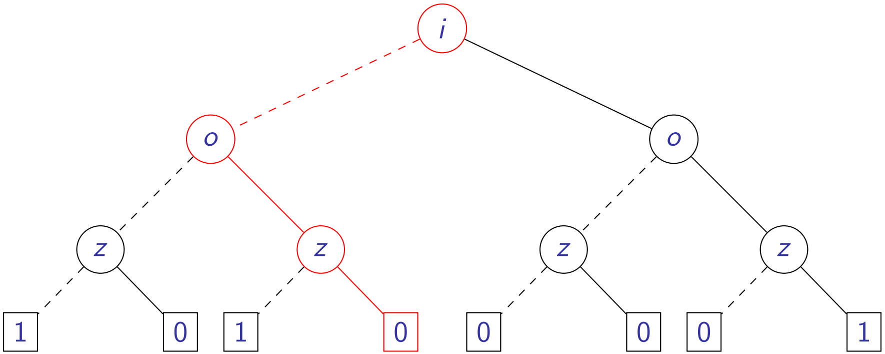


- $f = (i \land o \land z) \lor (\neg i \land \neg o)$
- Interpretation $\neg i, o, z$



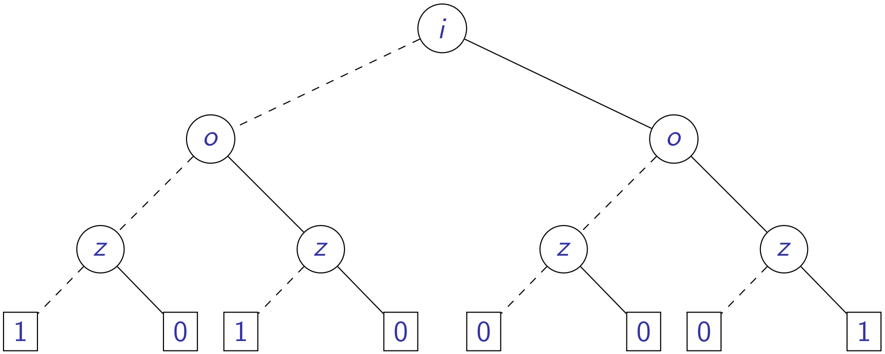
Example of Boolean Formula Represented in BDDs

- $f = (i \wedge o \wedge z) \vee (\neg i \wedge \neg o)$
- Interpretation $\neg i, o, z$

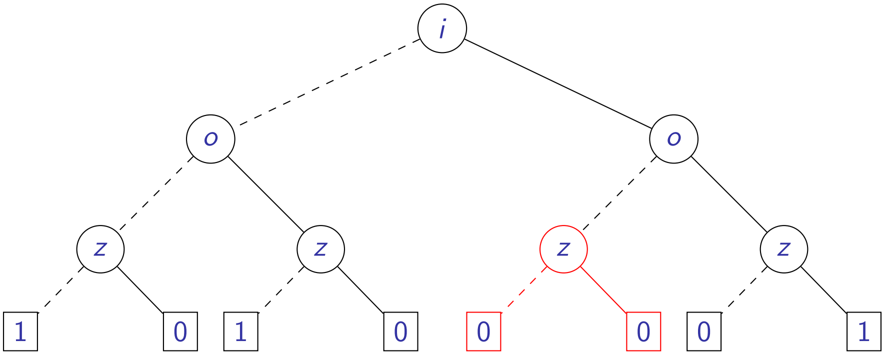


- BDD is able to represent a Boolean formula
- BDD: Compact representation
  - **Elimination rule**
  - **Isomorphism rule**

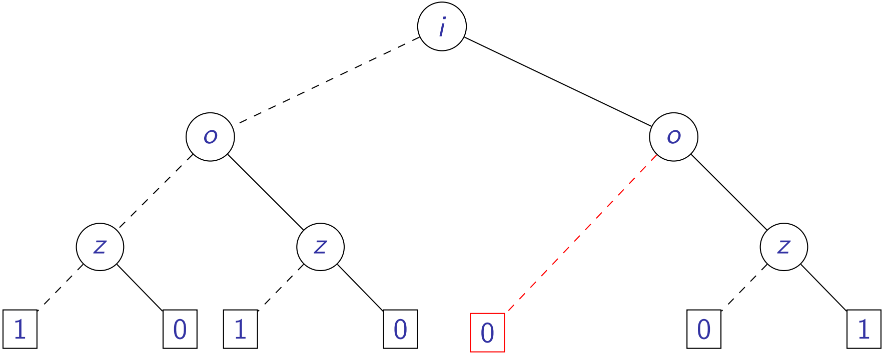
Elimination rule: If low(v) = high(v) = w, eliminate v and redirect all incoming edges to v to node w.



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



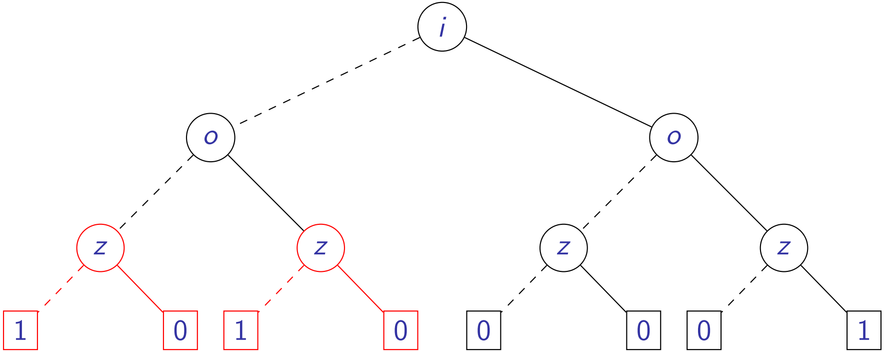
Elimination rule: If low(v) = high(v) = w, eliminate v and redirect all incoming edges to v to node w.



Isomorphism rule:

If $v \neq w$ are roots of isomorphic subtrees, remove v, and redirect all incoming edges to v to node w.

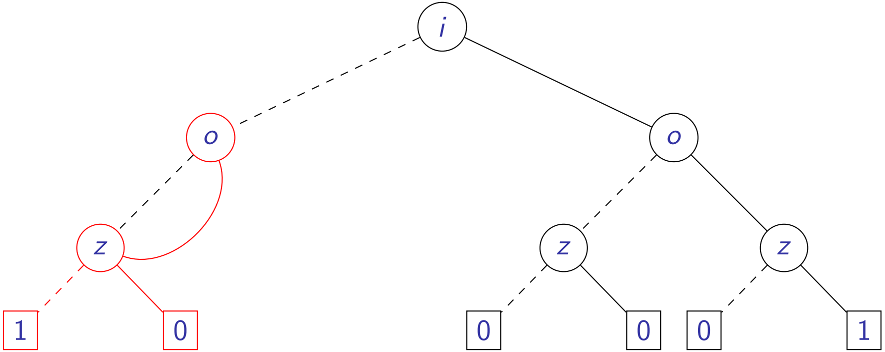
Combine all 0/1-leaves, redirect all incoming edges.



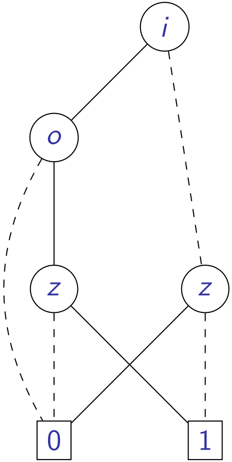
Isomorphism rule:

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.



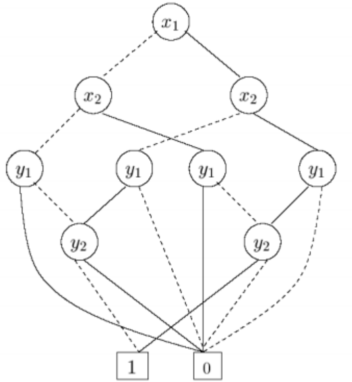
Binary Decision Diagram: Reduced

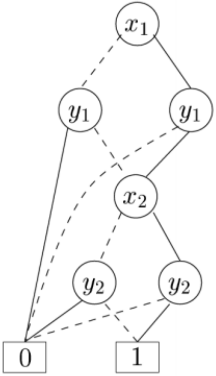


BDD size: #nodes.

BDD size highly depends on the variable ordering.

$f =$
$(x_1 \wedge x_2 \wedge y_1 \wedge y_2) \vee (\neg x_1 \wedge x_2 \wedge \neg y_1 \wedge y_2) \vee (x_1 \wedge \neg x_2 \wedge y_1 \wedge \neg y_2) \vee (\neg x_1 \wedge \neg x_2 \wedge \neg y_1 \wedge \neg y_2)$.





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

- 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.

- 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
- Reachability games in BDDs

$\mathcal{D}_p = \{\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f\}$

- $\mathcal{I}, \mathcal{O}$ environment and agent variables

$\mathcal{D}_p = \{\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f\}$

- $\mathcal{I}, \mathcal{O}$ BDD variables of the environment and the agent

$\mathcal{D}_p = \{\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f\}$

- $\mathcal{I}, \mathcal{O}$ BDD variables of the environment and the agent
- $\mathcal{Z}$ a set of state variables

$\mathcal{D}_p = \{\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f\}$

- $\mathcal{I}, \mathcal{O}$ BDD variables of the environment and the agent
- $\mathcal{Z}$ BDD variables

$\mathcal{D}_p = \{\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f\}$

- $\mathcal{I}, \mathcal{O}$ BDD variables of the environment and the agent
- $\mathcal{Z}$ BDD variables
- ι Boolean formula over $\mathcal{Z}$ denoting the initial state

$\mathcal{D}_p = \{\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f\}$

- $\mathcal{I}, \mathcal{O}$ BDD variables of the environment and the agent
- $\mathcal{Z}$ BDD variables
- ι BDD B_ι over $\mathcal{Z}$ denoting the initial state

$\mathcal{D}_p = \{\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f\}$

- $\mathcal{I}, \mathcal{O}$ BDD variables of the environment and the agent
- $\mathcal{Z}$ BDD variables
- ι BDD B_ι over $\mathcal{Z}$ denoting the initial state
- $\eta = \{\eta_z \mid z \in \mathcal{Z}\}$ a sequence of Boolean formulas over $\mathcal{Z} \cup \mathcal{P}$ encoding the transition function

$\mathcal{D}_p = \{\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f\}$

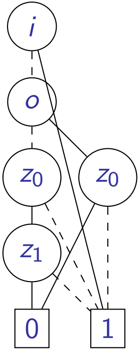
- $\mathcal{I}, \mathcal{O}$ BDD variables of the environment and the agent
- $\mathcal{Z}$ BDD variables
- ι BDD B_ι over $\mathcal{Z}$ denoting the initial state
- η a sequence of BDDs over $\mathcal{Z} \cup \mathcal{P}$ encoding the transition function

$\mathcal{D}_p = \{\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f\}$

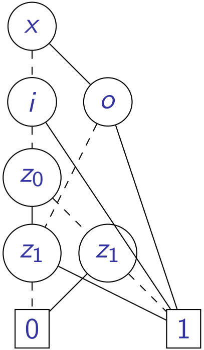
- $\mathcal{I}, \mathcal{O}$ BDD variables of the environment and the agent
- $\mathcal{Z}$ BDD variables
- ι BDD B_ι over $\mathcal{Z}$ denoting the initial state
- η a sequence of BDDs over $\mathcal{Z} \cup \mathcal{P}$ encoding the transition function
- f Boolean formula over $\mathcal{Z}$ representing the set of accepting states

$\mathcal{D}_p = \{\mathcal{I}, \mathcal{O}, \mathcal{Z}, \iota, \eta, f\}$

- $\mathcal{I}, \mathcal{O}$ BDD variables of the environment and the agent
- $\mathcal{Z}$ BDD variables
- ι BDD B_ι over $\mathcal{Z}$ denoting the initial state
- η a sequence of BDDs over $\mathcal{Z} \cup \mathcal{P}$ encoding the transition function
- f BDD B_f over $\mathcal{Z}$ representing the set of accepting states



BDD of η_{z_0}



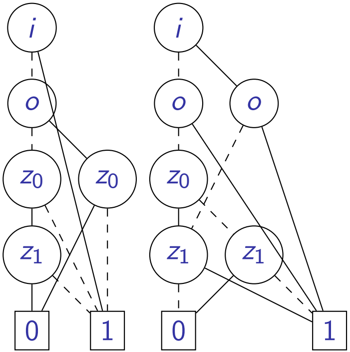
BDD of η_{z_1}

Reachability game on symbolic DFA $\mathcal{D}_p = (\mathcal{X}, \mathcal{Y}, \mathcal{Z}, B_\iota, \eta, B_f)$ in BDDs

- $B_{w_0} = B_f$
- $B_{t_0} = B_f$

$$t_{i+1} = t_i \vee (\neg w_i \wedge \forall I. w_i(\eta))$$

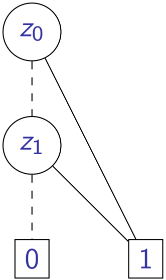
- $\eta = \{\eta_z \mid z \in \mathcal{Z}\}$



BDD of η_{z_0} BDD of η_{z_1}

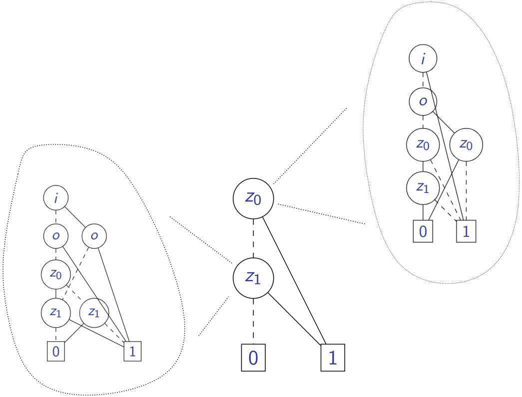
$$t_{i+1} = t_i \vee (\neg w_i \wedge \forall I . w_i(\eta))$$

- B_{w_i}



$$t_{i+1} = t_i \vee (\neg w_i \wedge \forall I . w_i(\eta))$$

- $w_i(\eta)$ transitions leading to states in w_i



BDD Compose

$$t_{i+1} = t_i \lor (\neg w_i \land \forall I . w_i(\eta))$$

– Universal Quantification

$$t_{i+1} = t_i \vee (\neg w_i \wedge \forall I.w_i(\eta))$$

- Conjunction, Negation, and Disjunction

$w_{i+1} = \exists O . t_{i+1}$

– Existential Quantification

Fixpoint check $w_{i+1} \equiv w_i$

– Equivalence check, constant time

Strategy abstraction $\tau : 2^{\mathcal{Z}} \rightarrow 2^{\mathcal{O}}$

- SolveEqn

- Symbolic synthesis techniques
  - LTL_f synthesis with partitioned representation in BDDs

Future directions to explore:

- Symbolic synthesis with monolithic representation?
- Using SAT instead of BDD?

1- Introduction to Planning and Synthesis (Giuseppe Perelli)

2- Planning with temporally extended goals (Giuseppe Perelli)

3- LTL_f synthesis under LTL specifications (Antonio Di Stasio)

4- Notable cases of LTL_f synthesis under LTL specifications (Shufang Zhu)

5- Symbolic Synthesis (Shufang Zhu)