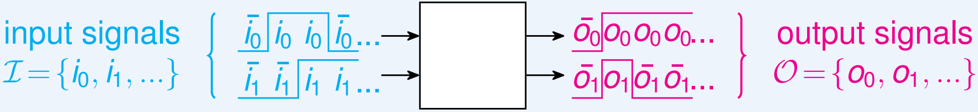
CIAA'25

# Engineering an LTL$_f$ synthesis tool

Alexandre Duret-Lutz Shufang Zhu Nir Piterman
Giuseppe De Giacomo Moshe Y. Vardi

# Reactive Synthesis in a Nutshell

## A **reactive controller** produces output as a reaction to its input



input signals

$\mathcal{I} = \{ i_0, i_1, \ldots \}$

output signals

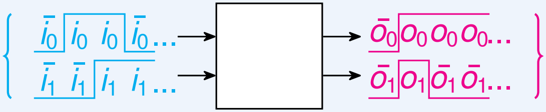
$\mathcal{O} = \{ o_0, o_1, \ldots \}$

# Reactive Synthesis in a Nutshell

## A **reactive controller** produces output as a reaction to its input

input signals

$\mathcal{I} = \{i_0, i_1, \dots\}$



output signals

$\mathcal{O} = \{o_0, o_1, \dots\}$

## The reactive synthesis problems

Given a specification relating input signals and output signals over time:

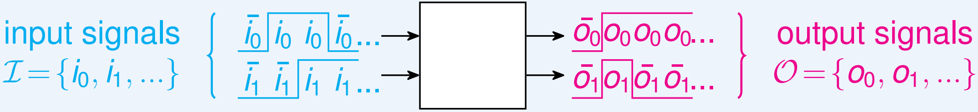
**Realizability:** decide if a controller exists;

our focus

**Synthesis:** construct it (e.g., as an And-Inverter Graph).

# Reactive Synthesis in a Nutshell

## A **reactive controller** produces output as a reaction to its input



input signals

$\mathcal{I} = \{i_0, i_1, \ldots\}$

$\bar{i}_0\; i_0\; i_0\; \bar{i}_0 \ldots$

$\bar{i}_1\; \bar{i}_1\; i_1\; i_1 \ldots$

$\bar{o}_0\; o_0\; o_0\; o_0 \ldots$

$\bar{o}_1\; o_1\; \bar{o}_1\; \bar{o}_1 \ldots$

output signals

$\mathcal{O} = \{o_0, o_1, \ldots\}$

## The reactive synthesis problems

Given a specification relating input signals and output signals over time:

**Realizability:** decide if a controller exists;

our focus

**Synthesis:** construct it (e.g., as an And-Inverter Graph).

## Semantics for an LTL$_f$ specification

Any execution of the controller, seen as an infinite word such as "$\bar{i}_0 \bar{i}_1\, \bar{o}_0 \bar{o}_1$; $i_0 \bar{i}_1\, o_0 o_1$; $i_0 i_1\, o_0 \bar{o}_1$; …", must have a finite prefix satisfying the specification.

# Text-Book Approach

**1. LTL$_f$ specification $\varphi$**

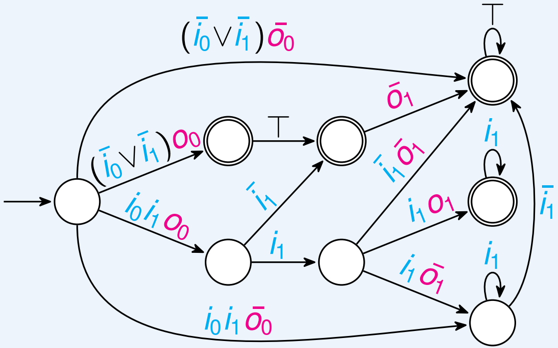
$(i_0 \wedge \mathsf{G}\, i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^! \mathsf{X}^!\, o_1)$

# Text-Book Approach

## 1. LTL$_f$ specification $\varphi$

$(i_0 \wedge \mathsf{G}\, i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^! \mathsf{X}^! o_1)$

## 2. Build a DFA $\mathcal{A}_\varphi$

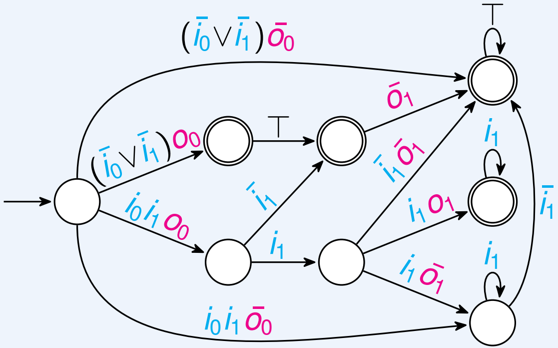


# Text-Book Approach

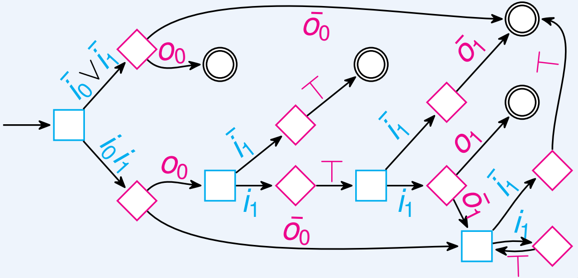
## 1. LTL$_f$ specification $\varphi$

$(i_0 \wedge \mathsf{G}\, i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^{!}\mathsf{X}^{!}\, o_1)$

## 2. Build a DFA $\mathcal{A}_\varphi$



## 3. Make it a Reachability Game

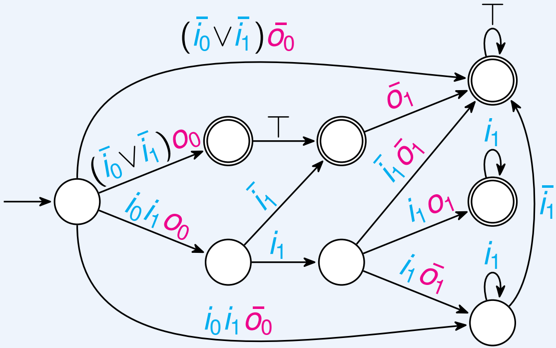


# Text-Book Approach

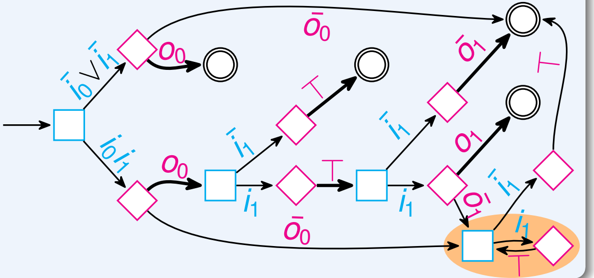
## 1. LTL$_f$ specification $\varphi$

$(i_0 \wedge G\, i_1) \leftrightarrow (o_0 \wedge X^! X^! o_1)$

## 2. Build a DFA $\mathcal{A}_\varphi$



## 3. Make it a Reachability Game



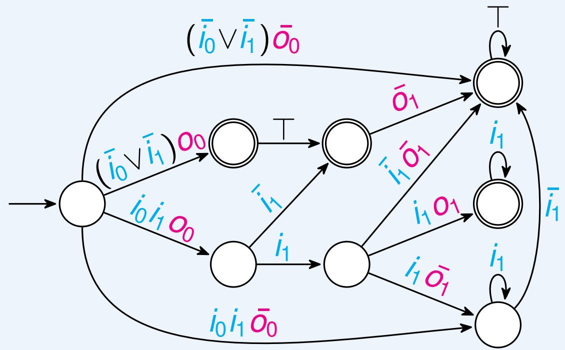
## 4. Solve it

# Text-Book Approach

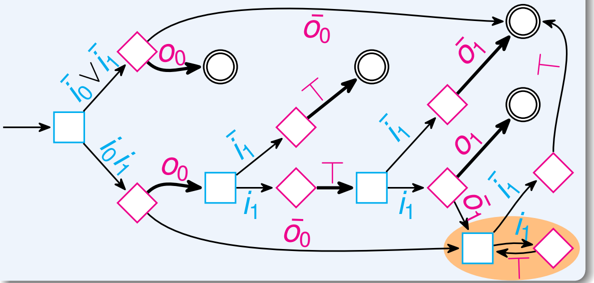
## 1. LTL$_f$ specification $\varphi$

$(i_0 \wedge \mathsf{G}\, i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^! \mathsf{X}^! o_1)$

## 2. Build a DFA $\mathcal{A}_\varphi$

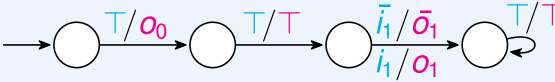


## 3. Make it a Reachability Game



## 4. Solve it

## 5. Extract a Controller if Desired



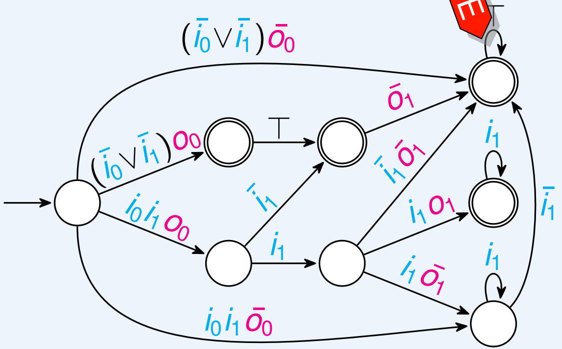
# Text-Book Approach

## 1. LTL$_f$ specification $\varphi$

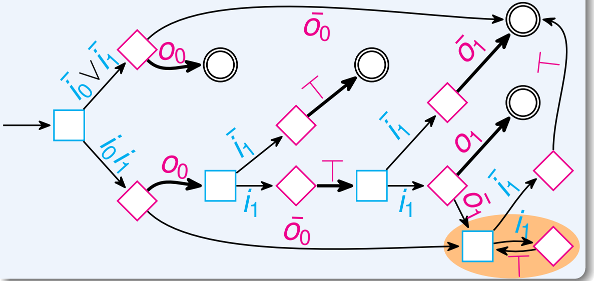
$(i_0 \wedge \mathsf{G}\, i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^! \mathsf{X}^! o_1)$

2-EXPTIME

## 2. Build a DFA $\mathcal{A}_\varphi$

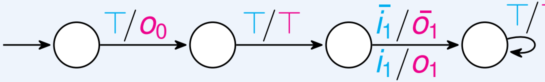


## 3. Make it a Reachability Game



## 4. Solve it

## 5. Extract a Controller if Desired



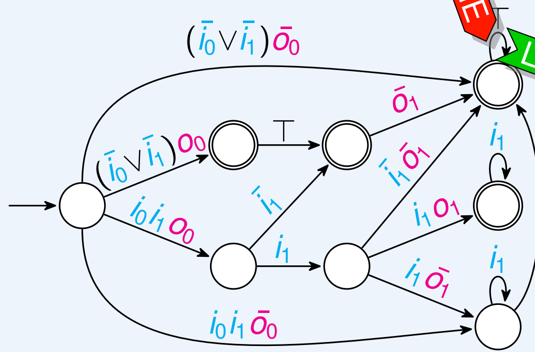
# Text-Book Approach

## 1. LTL$_f$ specification $\varphi$

$(i_0 \wedge \mathsf{G}\, i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^! \mathsf{X}^!\, o_1)$

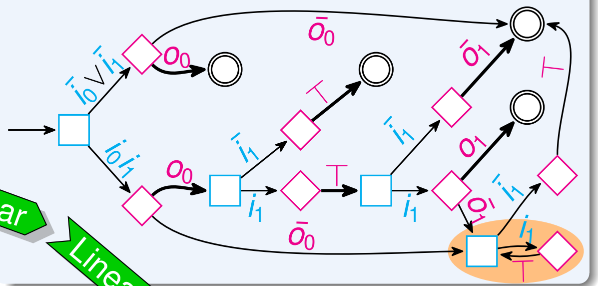
2-EXPTIME

## 2. Build a DFA $\mathcal{A}_\varphi$



Linear

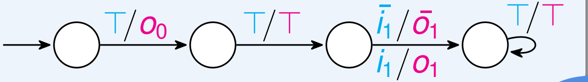
## 3. Make it a Reachability Game



Linear

## 4. Solve it

## 5. Extract a Controller if Desired



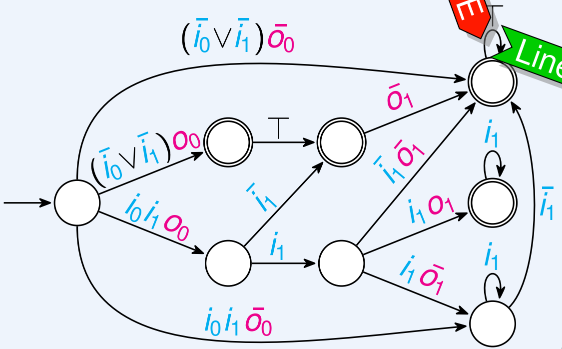
# Text-Book Approach

## 1. LTL$_f$ specification $\varphi$

$(i_0 \wedge \mathsf{G} i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^! \mathsf{X}^! o_1)$

2-EXPTIME

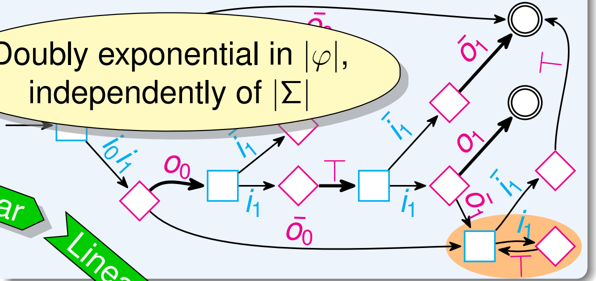
## 2. Build a DFA $\mathcal{A}_\varphi$



Linear

## 3. Make it a Reachability Game

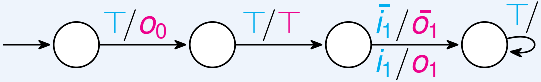
Doubly exponential in $|\varphi|$, independently of $|\Sigma|$



Linear

## 4. Solve it

## 5. Extract a Controller if Desired



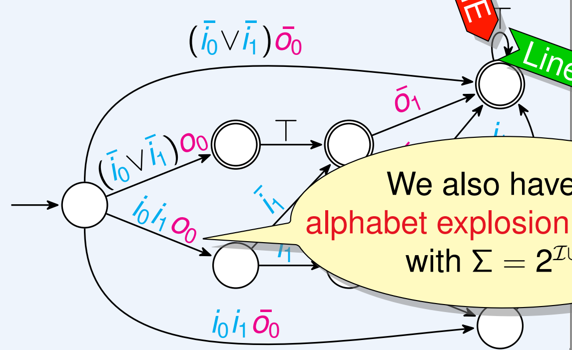
# Text-Book Approach

## 1. LTL$_f$ specification $\varphi$

$(i_0 \wedge \mathsf{G} i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^! \mathsf{X}^! o_1)$

2-EXPTIME

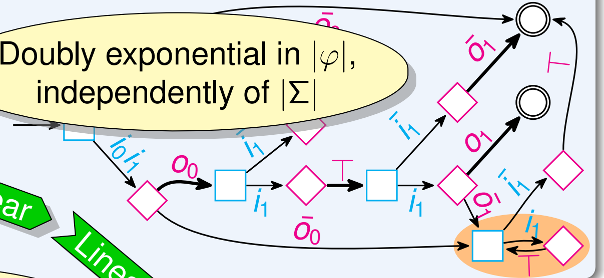
## 2. Build a DFA $\mathcal{A}_\varphi$



We also have an alphabet explosion problem, with $\Sigma = 2^{\mathcal{I} \cup \mathcal{O}}$

Linear

## 3. Make it a Reachability Game

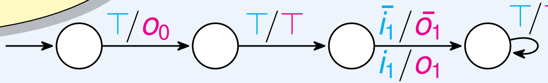


Doubly exponential in $|\varphi|$, independently of $|\Sigma|$

Linear

## lve it

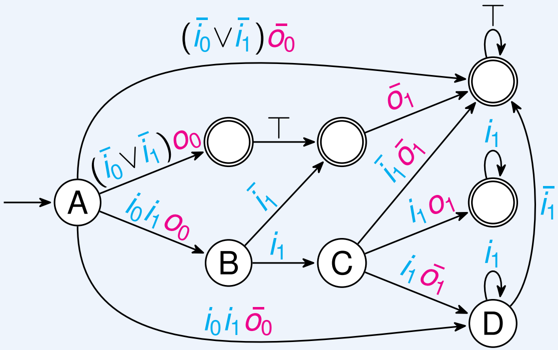
## ct a Controller if Desired



# Stopping the DFA Construction on Final States

The goal is to reach final states: we do not care about what follows.

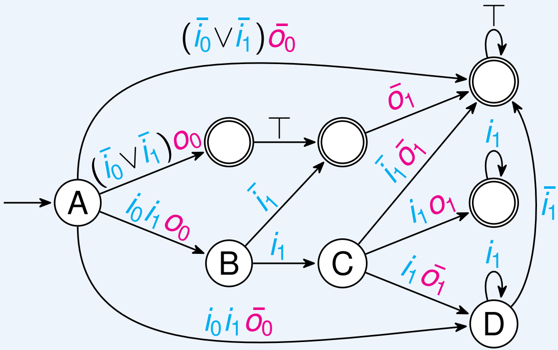
## Overkill



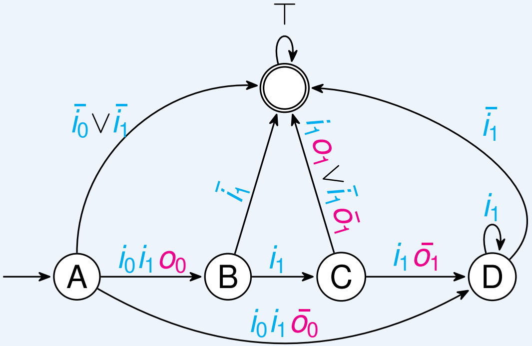
# Stopping the DFA Construction on Final States

The goal is to reach final states: we do not care about what follows.

## Overkill

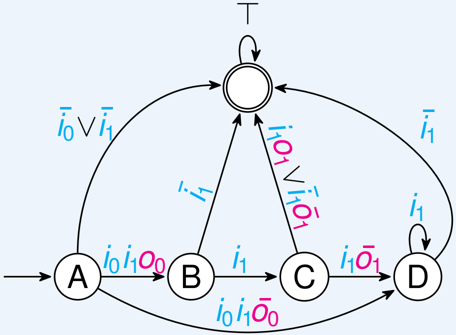


## Better



# Fighting Alphabet Explosion with MTBDDs

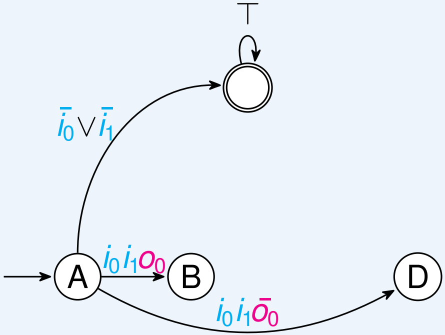
## Explicit representation



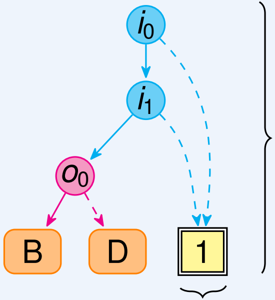
## Semi-symbolic representation: MTBDD

# Fighting Alphabet Explosion with MTBDDs

## Explicit representation



## Semi-symbolic representation: MTBDD

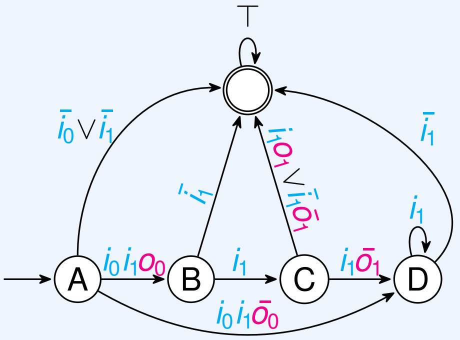


MTBDD representing outgoing edges of Ⓐ (unique for a given variable order)

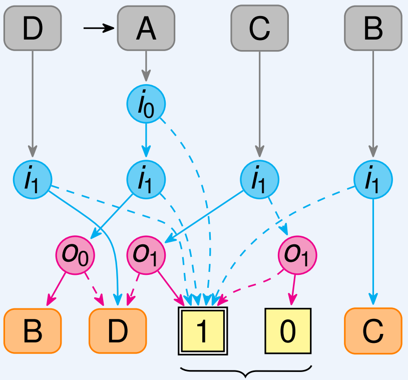
true BDD node
as accepting sink

# Fighting Alphabet Explosion with MTBDDs

## Explicit representation



## Semi-symbolic representation: **MTDFA**



Array of roots

MTBDDs representing outgoing edges (will share nodes)

true/false BDD nodes as accepting/rejecting sinks

# LTL$_f$ Synthesis with MTDFA

## What Other Tools Have Tried

**MTDFA Constructions**:

- Transform LTL$_f$ → FOL, then use Mona for FOL → MTDFA.
- Use Mona's MTDFA library to translate LTL$_f$ to MTDFA by composition.

**Game Solving**: convert MTDFA to BDD, and solve symbolically.

**Also exist on-the-fly approaches** that do not go through MTDFAs.

## What we Suggest

1. Direct translation from LTL$_f$ to MTBDD, building the MTDFA one state at a time.
2. Solving the game on the MTDFA directly.
3. Doing those on-the-fly.

# $\mathrm{LTL}_f$ Synthesis with MTDFA

## What Other Tools Have Tried

**MTDFA Constructions**:

- Transform $\mathrm{LTL}_f \rightarrow$ FOL, then use Mona for FOL $\rightarrow$ MTDFA.
- Use Mona's MTDFA library to translate $\mathrm{LTL}_f$ to MTDFA by composition.

**Game Solving**: convert MTDFA to BDD, and solve symbolically.

**Also exist on-the-fly approaches** that do not go through MTDFAs.

## What we Suggest

① Direct translation from $\mathrm{LTL}_f$ to MTBDD, building the MTDFA one state at a time.

② Solving the game on the MTDFA directly.

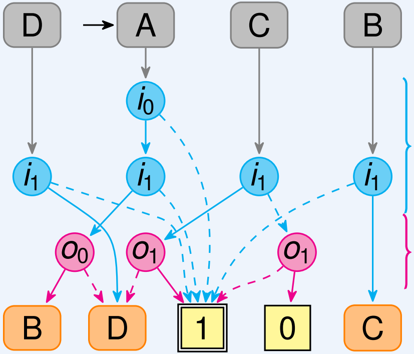
next slide

③ Doing those on-the-fly.

# Seeing the MTDFA as a Game Arena

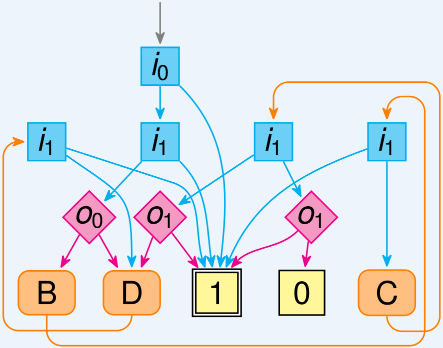
Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (Moore/Mealy).

## The MTDFA



inputs
above
outputs
for Mealy
semantics

## The Game Interpretation

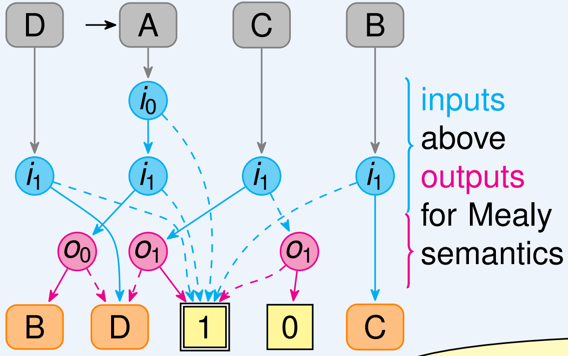


# Seeing the MTDFA as a Game Arena

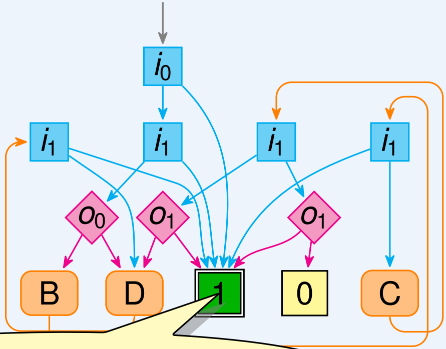
Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

Actually stores reversed edges.

## The MTDFA



## The Game Interpretation



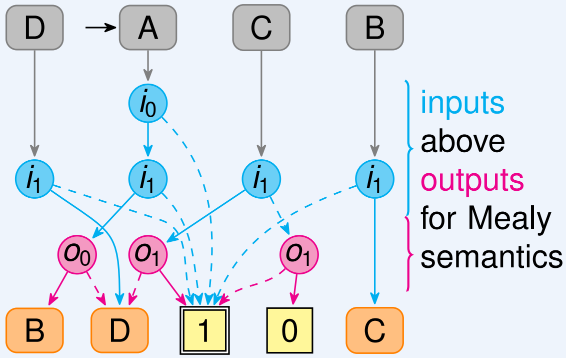
Solve by backpropagation from 1.

# Seeing the MTDFA as a Game Arena

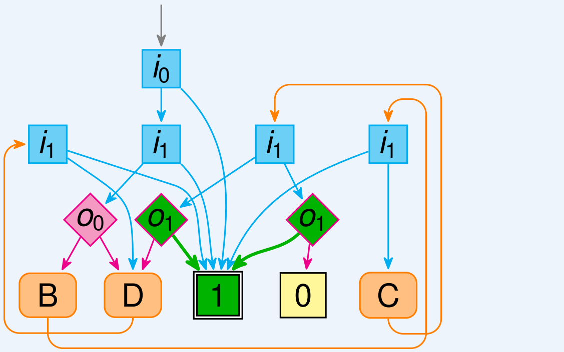
Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

Actually stores reversed edges.

## The MTDFA



## The Game Interpretation

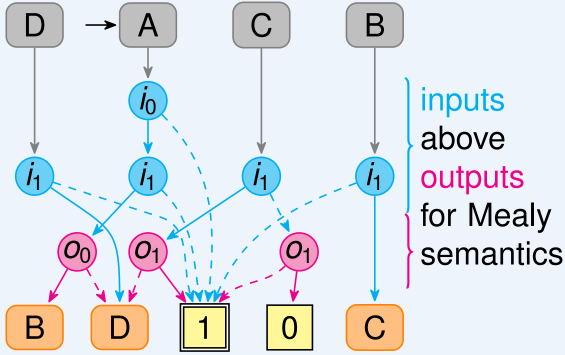


# Seeing the MTDFA as a Game Arena

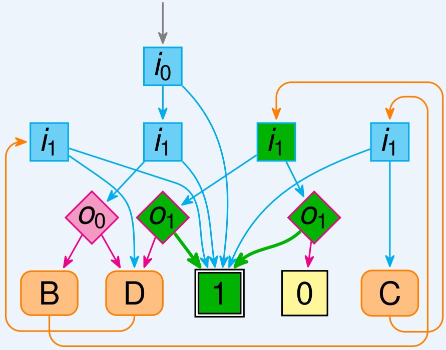
Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

Actually stores reversed edges.

## The MTDFA



## The Game Interpretation

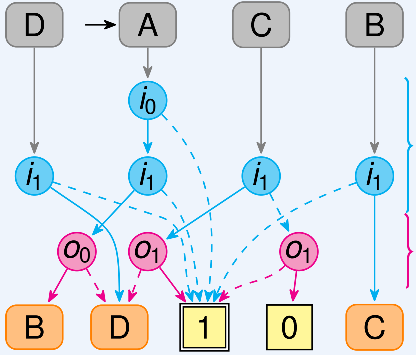


# Seeing the MTDFA as a Game Arena

Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

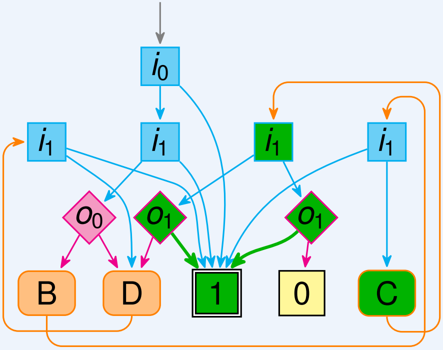
Actually stores reversed edges.

## The MTDFA



inputs above outputs for Mealy semantics

## The Game Interpretation

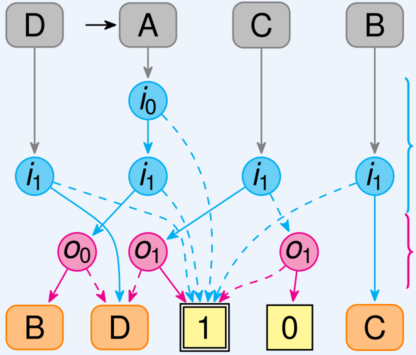


# Seeing the MTDFA as a Game Arena

Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

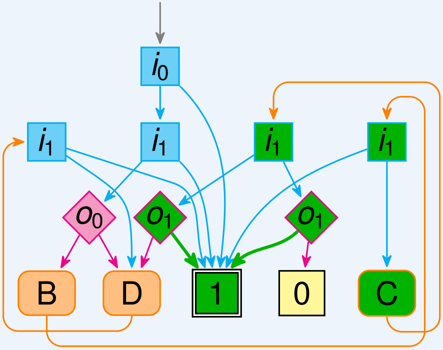
Actually stores reversed edges.

## The MTDFA



inputs above outputs for Mealy semantics

## The Game Interpretation

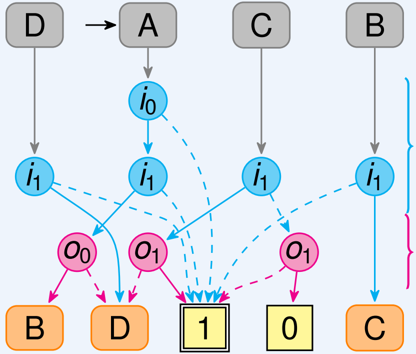


# Seeing the MTDFA as a Game Arena

Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

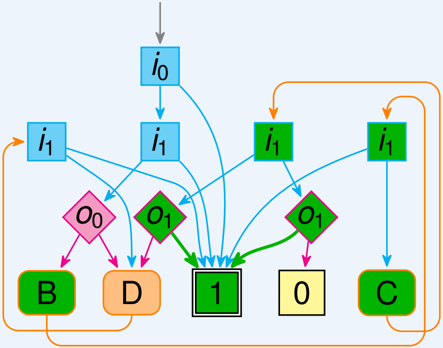
Actually stores reversed edges.

## The MTDFA



inputs above outputs for Mealy semantics

## The Game Interpretation

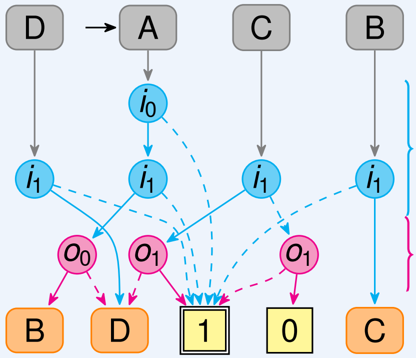


# Seeing the MTDFA as a Game Arena

Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

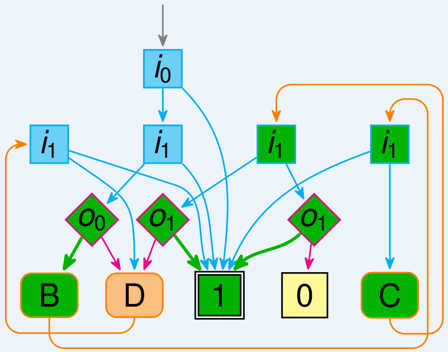
Actually stores reversed edges.

## The MTDFA



inputs above outputs for Mealy semantics

## The Game Interpretation

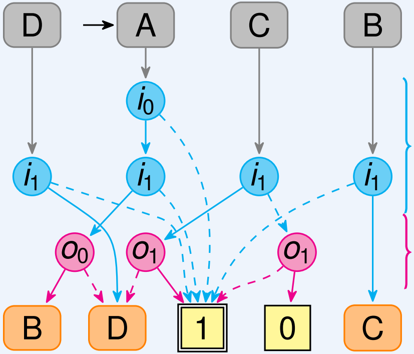


# Seeing the MTDFA as a Game Arena

Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

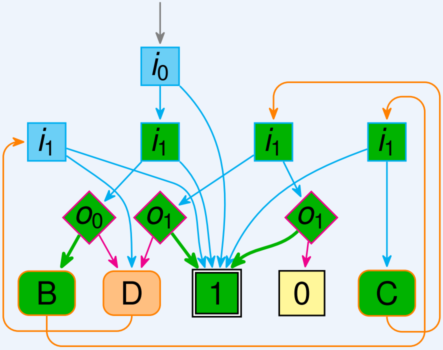
Actually stores reversed edges.

## The MTDFA



inputs above outputs for Mealy semantics

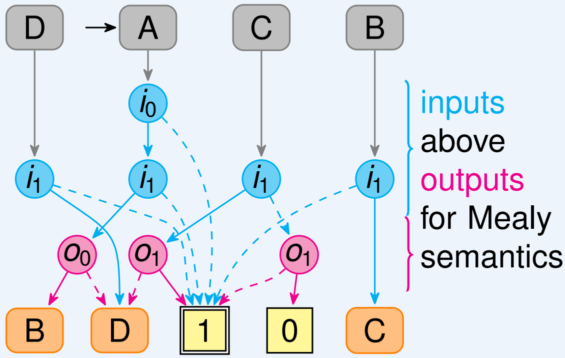
## The Game Interpretation



# Seeing the MTDFA as a Game Arena

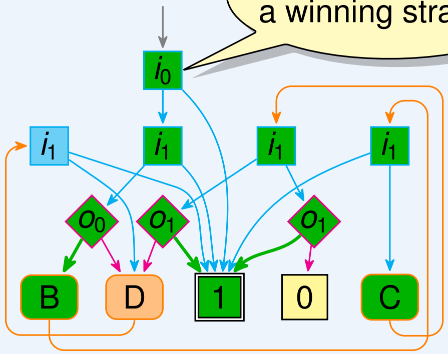
Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (Moore/Mealy).

## The MTDFA



inputs
above
outputs
for Mealy
semantics

## The Game Int



Output player has
a winning strategy.

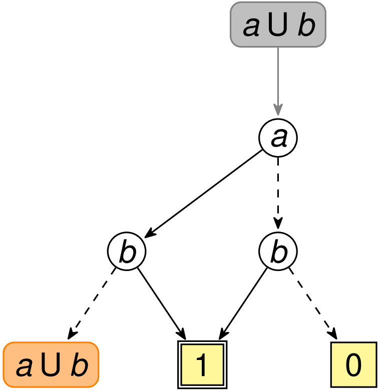
# LTL$_f$ Synthesis with MTDFA

## What we Suggest

1. Direct translation from LTL$_f$ to MTBDD, building the MTDFA one state at a time. up next!
2. Solving the game on the MTDFA directly. ✓
3. Doing those on-the-fly.

# From LTL$_f$ to MTBDD (not yet MTDFA)

Use LTL$_f$ formulas as terminals.
Assume we know an MTBDD for the successors of $a \,\mathsf{U}\, b$

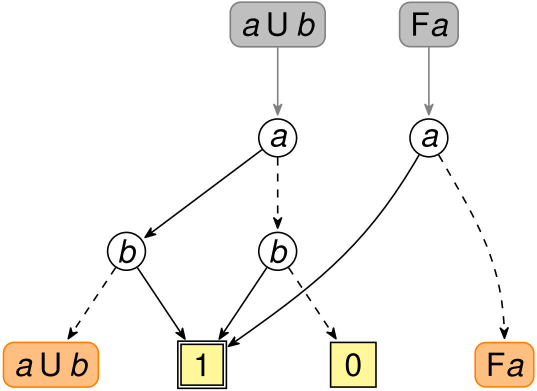


This is a deterministic representation of the *next normal form* (XNF):

$$a \,\mathsf{U}\, b \equiv b \vee (a \wedge \mathsf{X}^!(a \,\mathsf{U}\, b))$$

# From LTL$_f$ to MTBDD (not yet MTDFA)

Use LTL$_f$ formulas as terminals.
Assume we know an MTBDD for the successors of $a \,\mathsf{U}\, b$, and another for $\mathsf{F}a$.



This is a deterministic representation of the *next normal form* (XNF):

$$a \,\mathsf{U}\, b \equiv b \vee (a \wedge \mathsf{X}^!(a \,\mathsf{U}\, b))$$

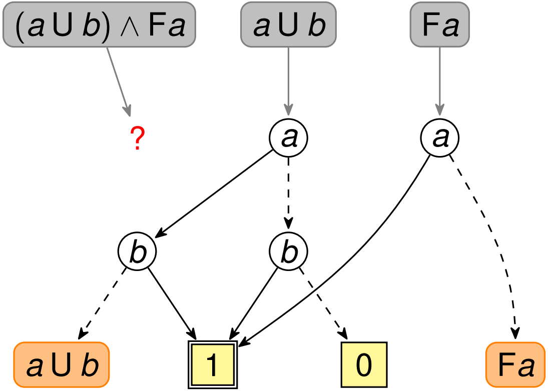
$$\mathsf{F}a \equiv a \vee \mathsf{X}^! \mathsf{F}a$$

# From LTL$_f$ to MTBDD (not yet MTDFA)

Use LTL$_f$ formulas as terminals.

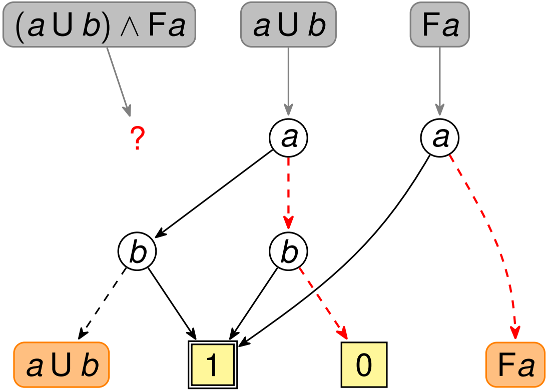
Assume we know an MTBDD for the successors of $a \, \mathsf{U} \, b$, and another for $\mathsf{F}a$.

We want to compute an MTBDD for $(a \, \mathsf{U} \, b) \wedge \mathsf{F}a$:



# From LTL$_f$ to MTBDD (not yet MTDFA)

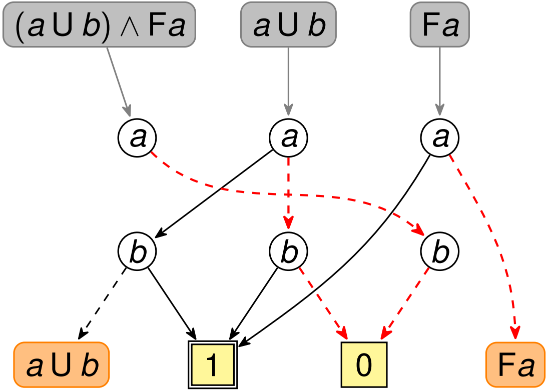
Use LTL$_f$ formulas as terminals.
Assume we know an MTBDD for the successors of $a \mathbin{\mathsf{U}} b$, and another for $\mathsf{F}a$.
We want to compute an MTBDD for $(a \mathbin{\mathsf{U}} b) \land \mathsf{F}a$:



Classical BDD `apply` procedure, but combine terminals with “$\land$”.

# From LTL$_f$ to MTBDD (not yet MTDFA)

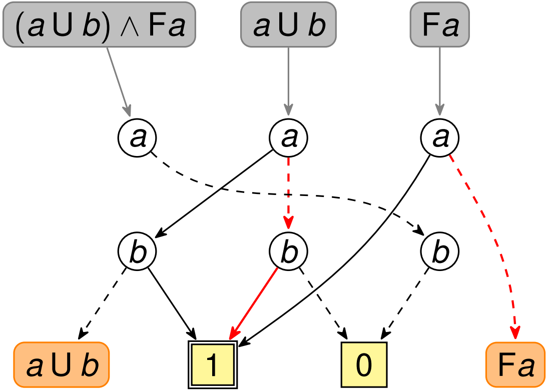
Use LTL$_f$ formulas as terminals.
Assume we know an MTBDD for the successors of $a \,\mathsf{U}\, b$, and another for $\mathsf{F}a$.
We want to compute an MTBDD for $(a \,\mathsf{U}\, b) \land \mathsf{F}a$:



Classical BDD `apply` procedure, but combine terminals with “$\land$”.

# From LTL$_f$ to MTBDD (not yet MTDFA)

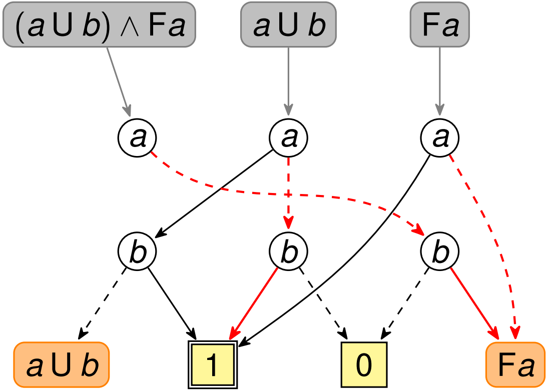
Use LTL$_f$ formulas as terminals.
Assume we know an MTBDD for the successors of $a \mathbin{\mathsf{U}} b$, and another for $\mathsf{F}a$.
We want to compute an MTBDD for $(a \mathbin{\mathsf{U}} b) \land \mathsf{F}a$:



Classical BDD `apply` procedure, but combine terminals with “$\land$”.

# From LTL$_f$ to MTBDD (not yet MTDFA)

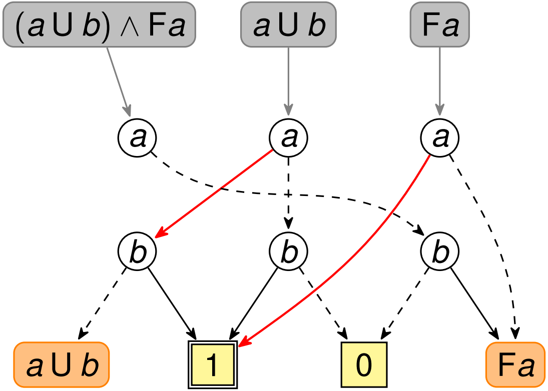
Use LTL$_f$ formulas as terminals.
Assume we know an MTBDD for the successors of $a \mathbin{\mathsf{U}} b$, and another for $\mathsf{F}a$.
We want to compute an MTBDD for $(a \mathbin{\mathsf{U}} b) \land \mathsf{F}a$:



Classical BDD `apply` procedure, but combine terminals with “$\land$”.

# From LTL$_f$ to MTBDD (not yet MTDFA)

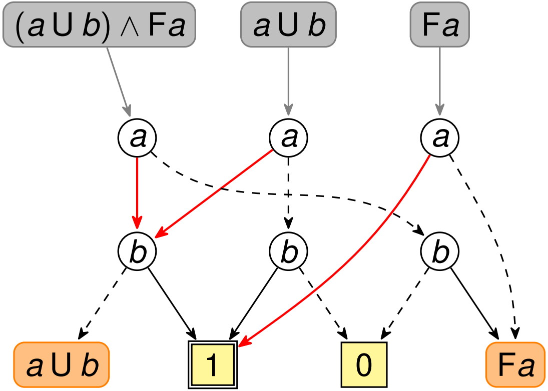
Use LTL$_f$ formulas as terminals.
Assume we know an MTBDD for the successors of $a\,\mathsf{U}\,b$, and another for $\mathsf{F}a$.
We want to compute an MTBDD for $(a\,\mathsf{U}\,b) \land \mathsf{F}a$:



Classical BDD `apply` procedure, but combine terminals with "∧".
Leaves $\boxed{0}$ and $\boxed{1}$ can help shortcut the recursion.

# From LTL$_f$ to MTBDD (not yet MTDFA)

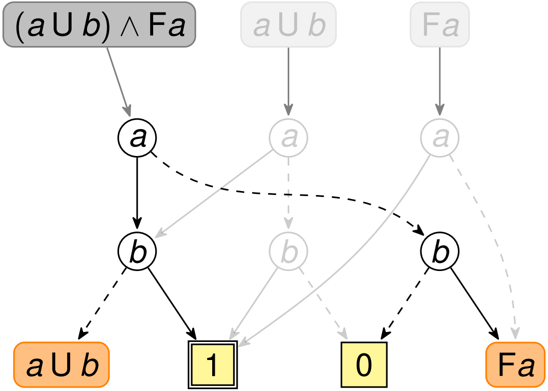
Use LTL$_f$ formulas as terminals.
Assume we know an MTBDD for the successors of $a \,\mathsf{U}\, b$, and another for $\mathsf{F}a$.
We want to compute an MTBDD for $(a \,\mathsf{U}\, b) \wedge \mathsf{F}a$:



Classical BDD `apply` procedure, but combine terminals with “$\wedge$”.
Leaves $\boxed{0}$ and $\boxed{1}$ can help shortcut the recursion.

# From LTL$_f$ to MTBDD (not yet MTDFA)

Use LTL$_f$ formulas as terminals.
Assume we know an MTBDD for the successors of $a \,\mathsf{U}\, b$, and another for $\mathsf{F}a$.
We want to compute an MTBDD for $(a \,\mathsf{U}\, b) \land \mathsf{F}a$:



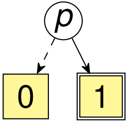
MTBDDs for subformulas are cached (i.e., not thrown away) in case they are needed later during the construction.

# From LTL$_f$ to MTBDD: Formal Definition

$\mathrm{tr}(\mathrm{ff}) = \boxed{0}$

$\mathrm{tr}(\mathrm{tt}) = \boxed{1}$

$\mathrm{tr}(p) =$



for $p \in \mathcal{I} \cup \mathcal{O}$

$\mathrm{tr}(\mathsf{X}\alpha) = \boxed{\alpha}$

$\mathrm{tr}(\mathsf{X}^{!}\alpha) = \boxed{\alpha}$

$\mathrm{tr}(\neg\alpha) = \neg\mathrm{tr}(\alpha)$

LTL$_f$ operator

MTBDD operator

$\mathrm{tr}(\alpha \odot \beta) = \mathrm{tr}(\alpha) \odot \mathrm{tr}(\beta)$ for any $\odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$

previous slide

$\mathrm{tr}(\alpha\, \mathsf{U}\, \beta) = \mathrm{tr}(\beta) \vee (\mathrm{tr}(\alpha) \wedge \boxed{\alpha\, \mathsf{U}\, \beta})$

$\mathrm{tr}(\mathsf{F}\alpha) = \mathrm{tr}(\alpha) \vee \boxed{\mathsf{F}\alpha}$

$\mathrm{tr}(\alpha\, \mathsf{R}\, \beta) = \mathrm{tr}(\beta) \wedge (\mathrm{tr}(\alpha) \vee \boxed{\alpha\, \mathsf{R}\, \beta})$

$\mathrm{tr}(\mathsf{G}\alpha) = \mathrm{tr}(\alpha) \wedge \boxed{\mathsf{G}\alpha}$

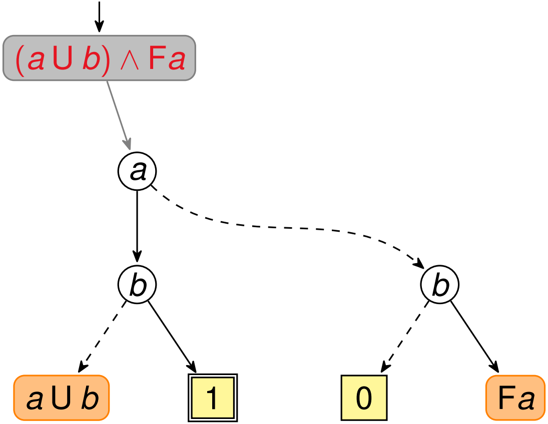
With the convention that $\boxed{\alpha} \wedge \boxed{\beta} = \boxed{\alpha \wedge \beta}$, $\boxed{\alpha} \vee \boxed{\beta} = \boxed{\alpha \vee \beta}$, ...

# From LTL$_f$ to MTDFA: Build States One at a Time

To translate $(a \mathbin{\mathsf{U}} b) \land \mathsf{F} a$:

1. Compute successors of the initial state: $\mathrm{tr}((a \mathbin{\mathsf{U}} b) \land \mathsf{F} a)$.

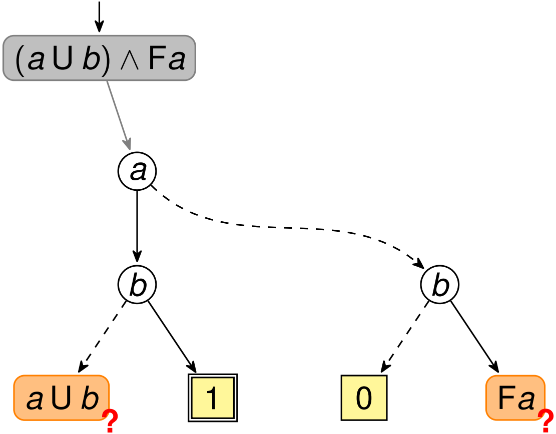
# From LTL$_f$ to MTDFA: Build States One at a Time



To translate $(a \mathbin{\mathsf{U}} b) \land \mathsf{F}a$:

① Compute successors of the initial state: $\mathrm{tr}((a \mathbin{\mathsf{U}} b) \land \mathsf{F}a)$.

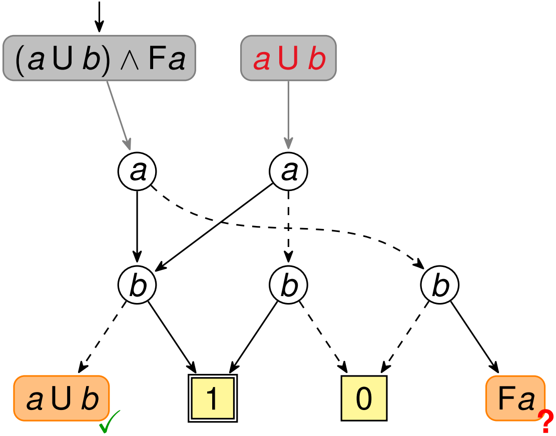
# From LTL$_f$ to MTDFA: Build States One at a Time



To translate $(a \mathbin{\mathsf{U}} b) \land \mathsf{F}a$:

1. Compute successors of the initial state: $\mathrm{tr}((a \mathbin{\mathsf{U}} b) \land \mathsf{F}a)$.
2. Compute successors for each new terminal:

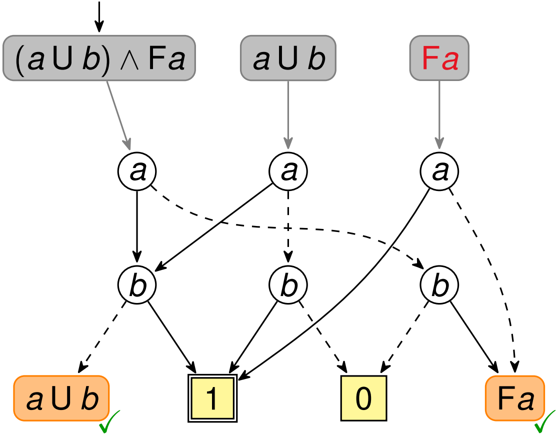
# From LTL$_f$ to MTDFA: Build States One at a Time



To translate $(a \mathbin{\mathsf{U}} b) \land \mathsf{F}a$:

1. Compute successors of the initial state: $\mathrm{tr}((a \mathbin{\mathsf{U}} b) \land \mathsf{F}a)$.
2. Compute successors for each new terminal:
    - $\mathrm{tr}(a \mathbin{\mathsf{U}} b)$ (cached)

# From LTL$_f$ to MTDFA: Build States One at a Time



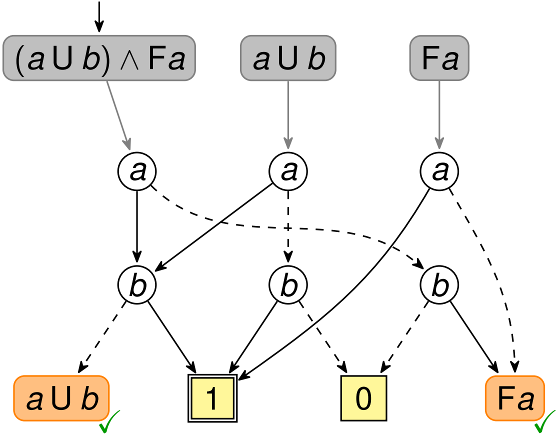
To translate $(a \, \mathsf{U} \, b) \wedge \mathsf{F}a$:

① Compute successors of the initial state: $\mathrm{tr}((a \, \mathsf{U} \, b) \wedge \mathsf{F}a)$.

② Compute successors for each new terminal:

- $\mathrm{tr}(a \, \mathsf{U} \, b)$ (cached)
- $\mathrm{tr}(\mathsf{F}a)$ (cached)

# From LTL$_f$ to MTDFA: Build States One at a Time

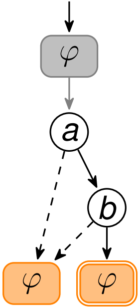


To translate $(a \mathbin{\mathsf{U}} b) \land \mathsf{F}a$:

1. Compute successors of the initial state: $\mathrm{tr}((a \mathbin{\mathsf{U}} b) \land \mathsf{F}a)$.
2. Compute successors for each new terminal:
   - $\mathrm{tr}(a \mathbin{\mathsf{U}} b)$ (cached)
   - $\mathrm{tr}(\mathsf{F}a)$ (cached)
3. Done.

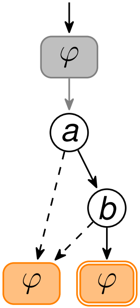
# Quick Note on Accepting Terminals

Our use of accepting terminals differs from Mona’s implementation of MTDFAs.

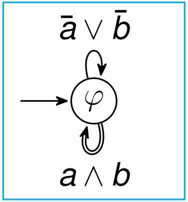


# Quick Note on Accepting Terminals

Our use of accepting terminals differs from Mona's implementation of MTDFAs.

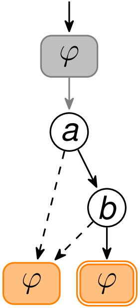


If you interpret the MTBDD roots as states, you get a transition-based DFA:

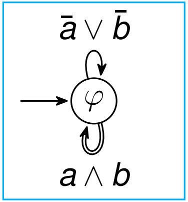


# Quick Note on Accepting Terminals

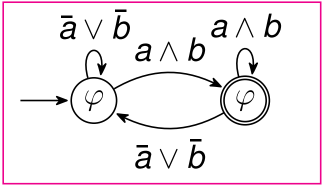
Our use of accepting terminals differs from Mona’s implementation of MTDFAs.



If you interpret the MTBDD roots as states, you get a transition-based DFA:

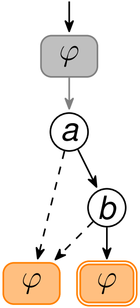


If you interpret the MTBDD terminals as states, you get a state-based DFA:

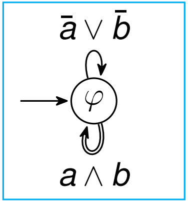


# Quick Note on Accepting Terminals

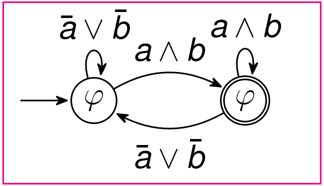
Our use of accepting terminals differs from Mona’s implementation of MTDFAs.



If you interpret the MTBDD roots as states, you get a transition-based DFA:



If you interpret the MTBDD terminals as states, you get a state-based DFA:



In any case, when using an MTDFA for synthesis, accepting terminals can all be replaced by the accepting sink $\boxed{1}$.

# $\mathrm{LTL}_f$ Synthesis with MTDFA

## What we Suggest

1. Direct translation from $\mathrm{LTL}_f$ to MTBDD, building the MTDFA one state at a time. ✓
2. Solving the game on the MTDFA directly. ✓
3. Doing those on-the-fly. next slide

# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \land \mathsf{G} i_1) \leftrightarrow (o_0 \land \mathsf{X}^! \mathsf{X}^! o_1)$$

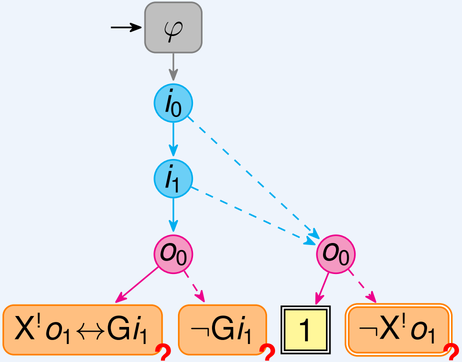
## The MTDFA

## The Game Interpretation

# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge \mathsf{G} i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^! \mathsf{X}^! o_1)$$

## The MTDFA

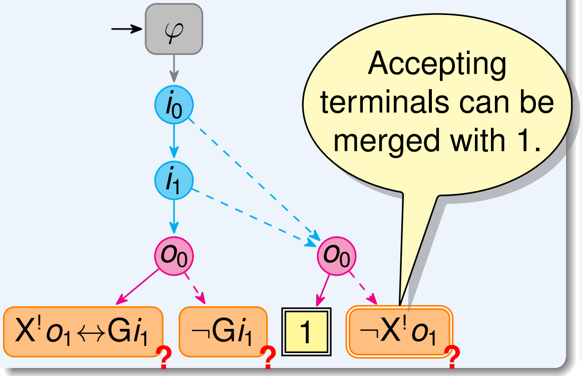


## The Game Interpretation

# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \land \mathsf{G}\, i_1) \leftrightarrow (o_0 \land \mathsf{X}^{!}\mathsf{X}^{!} o_1)$$

## The MTDFA



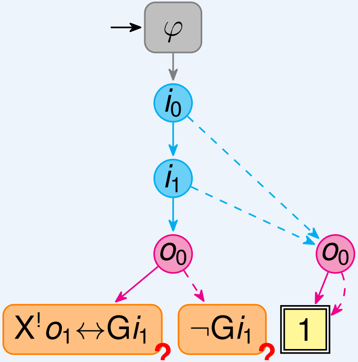
Accepting terminals can be merged with 1.

## The Game Interpretation

# Building the MTDFA & Solving the Game On-The-Fly

$\varphi = (i_0 \land \mathsf{G} i_1) \leftrightarrow (o_0 \land \mathsf{X}^! \mathsf{X}^! o_1)$

## The MTDFA

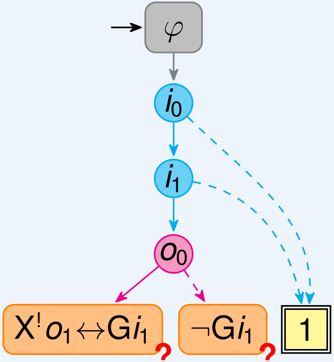


## The Game Interpretation

# Building the MTDFA & Solving the Game On-The-Fly

$\varphi = (i_0 \land \mathsf{G} i_1) \leftrightarrow (o_0 \land \mathsf{X}^! \mathsf{X}^! o_1)$

## The MTDFA

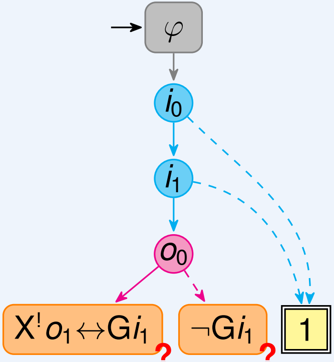


## The Game Interpretation

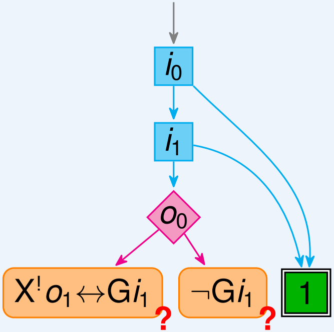
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge \mathsf{G}\, i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^{!}\mathsf{X}^{!}\, o_1)$$

## The MTDFA



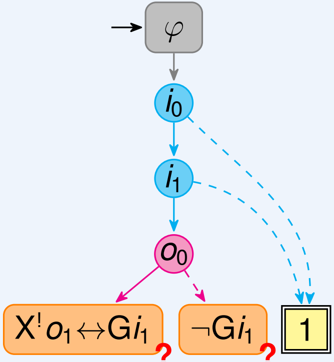
## The Game Interpretation



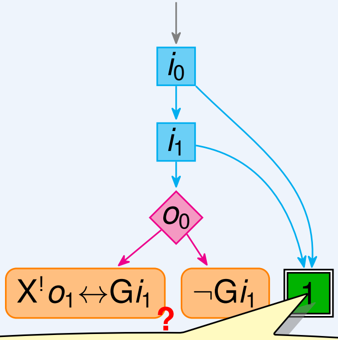
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \land \mathsf{G}\, i_1) \leftrightarrow (o_0 \land \mathsf{X}^!\mathsf{X}^!\, o_1)$$

## The MTDFA



## The Game Interpretation

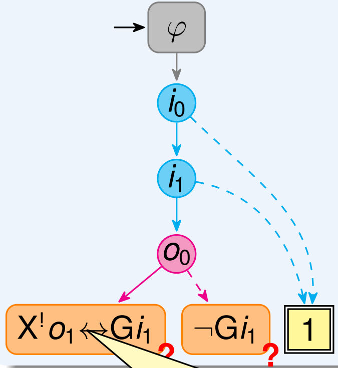


Backpropagation runs during construction.

# Building the MTDFA & Solving the Game On-The-Fly

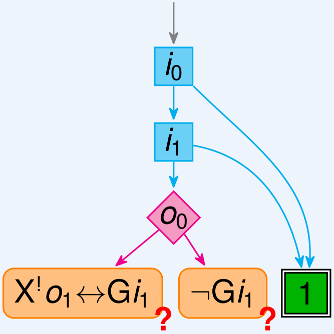
$\varphi = (i_0 \wedge \mathsf{G} i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^{!}\mathsf{X}^{!} o_1)$

## The MTDFA



Pick another terminal to develop.

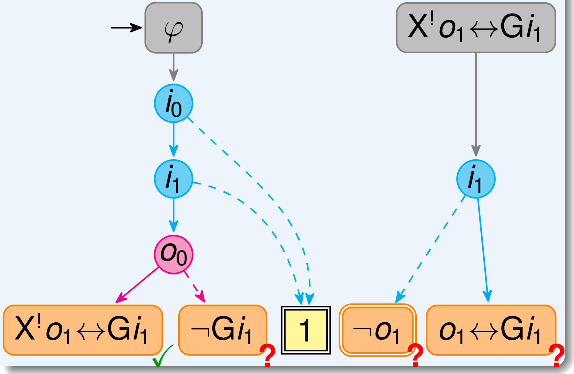
## The Game Interpretation



# Building the MTDFA & Solving the Game On-The-Fly

$\varphi = (i_0 \wedge \mathsf{G}\, i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^{!}\mathsf{X}^{!}\, o_1)$

### The MTDFA



$\varphi$

$\mathsf{X}^{!}o_1 \leftrightarrow \mathsf{G}i_1$

$i_0$

$i_1$

$i_1$

$o_0$

$\mathsf{X}^{!}o_1 \leftrightarrow \mathsf{G}i_1$ ✓

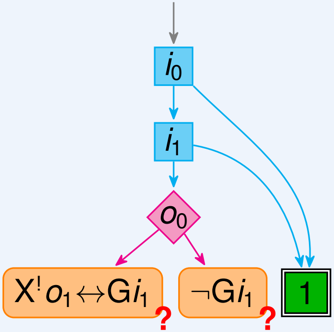
$\neg \mathsf{G}i_1$ ?

1

$\neg o_1$ ?

$o_1 \leftrightarrow \mathsf{G}i_1$ ?

### The Game Interpretation



$i_0$

$i_1$

$o_0$

$\mathsf{X}^{!}o_1 \leftrightarrow \mathsf{G}i_1$ ?

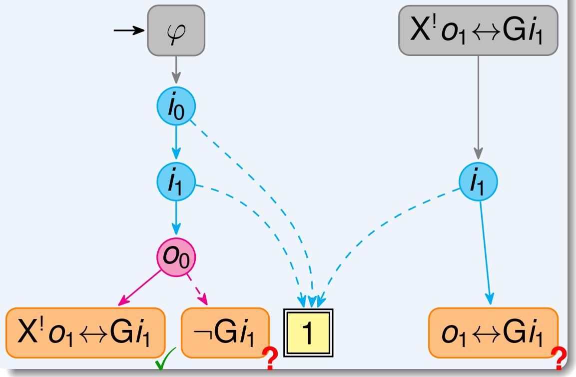
$\neg \mathsf{G}i_1$ ?

1

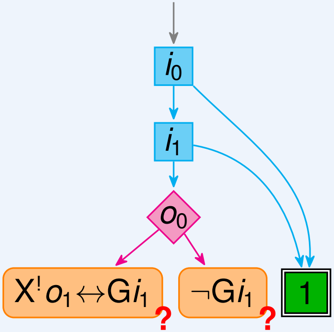
# Building the MTDFA & Solving the Game On-The-Fly

$\varphi = (i_0 \land \mathsf{G} i_1) \leftrightarrow (o_0 \land \mathsf{X}^! \mathsf{X}^! o_1)$

## The MTDFA



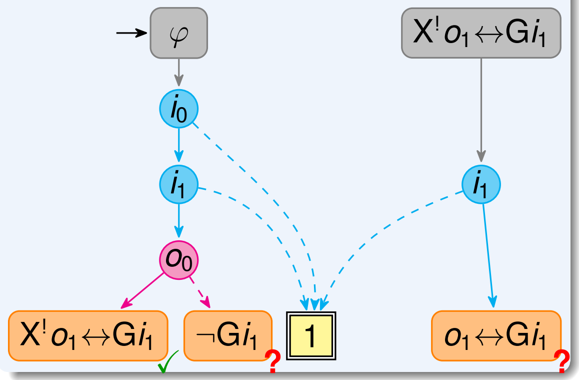
## The Game Interpretation



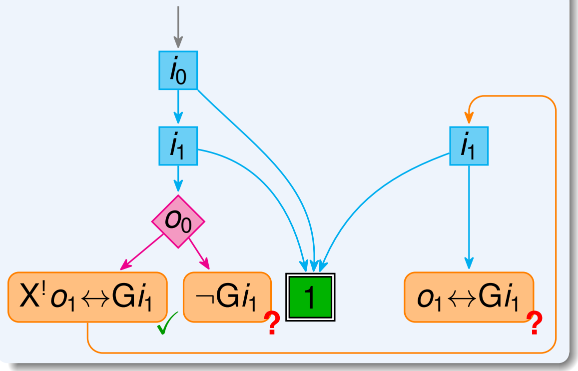
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge \mathsf{G} i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^! \mathsf{X}^! o_1)$$

## The MTDFA



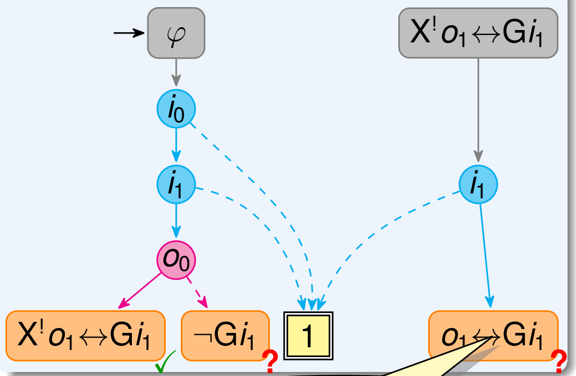
## The Game Interpretation



# Building the MTDFA & Solving the Game On-The-Fly

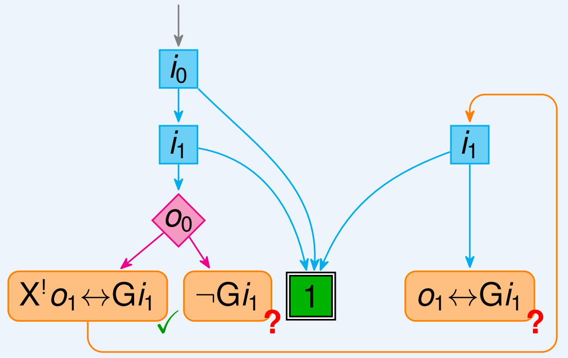
$$\varphi = (i_0 \wedge \mathsf{G}\, i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^{!}\mathsf{X}^{!} o_1)$$

## The MTDFA



Let’s develop this one next.

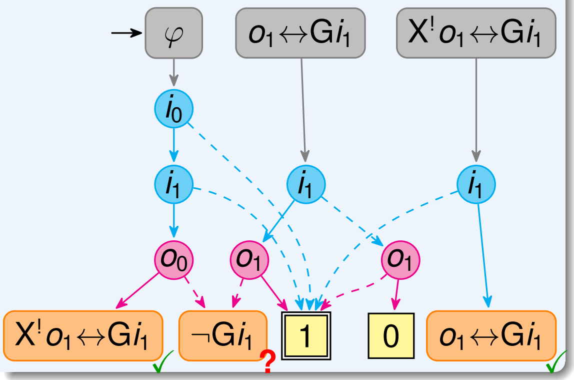
## The Game Interpretation



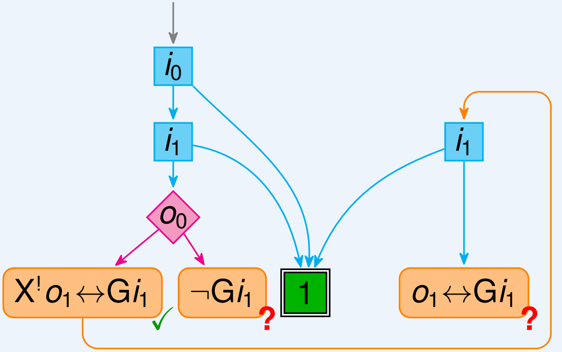
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge \mathsf{G}\, i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^{!}\mathsf{X}^{!} o_1)$$

## The MTDFA



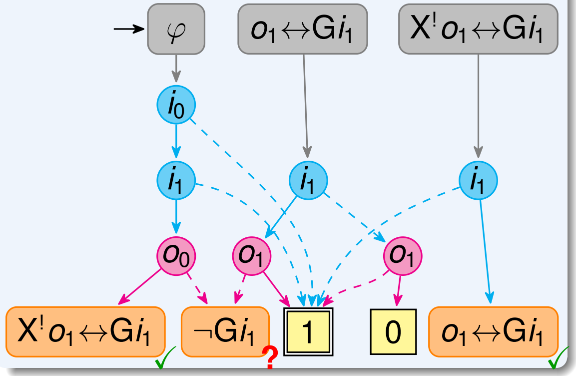
## The Game Interpretation



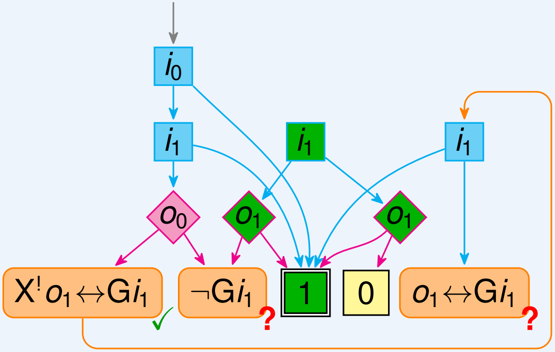
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge \mathsf{G}\, i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^{!}\mathsf{X}^{!}\, o_1)$$

## The MTDFA



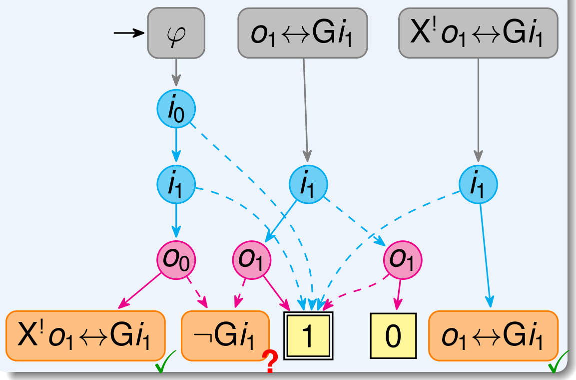
## The Game Interpretation



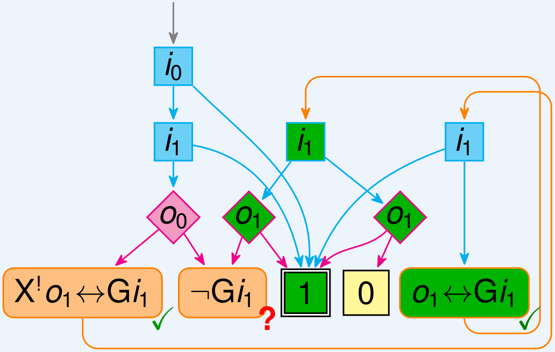
# Building the MTDFA & Solving the Game On-The-Fly

$\varphi = (i_0 \wedge \mathsf{G} i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^! \mathsf{X}^! o_1)$

## The MTDFA



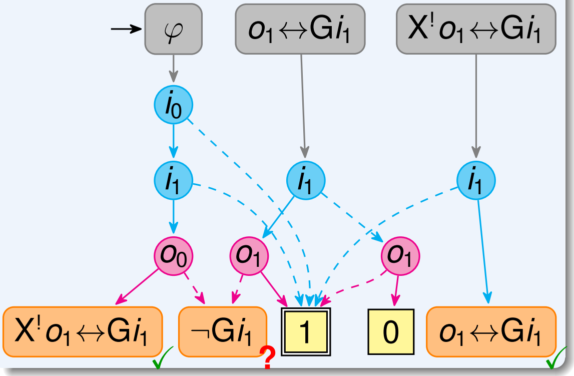
## The Game Interpretation



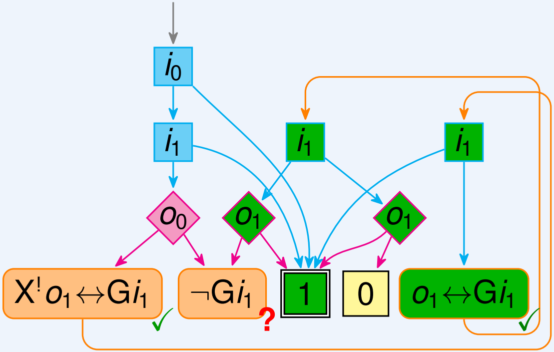
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \land \mathsf{G}\, i_1) \leftrightarrow (o_0 \land \mathsf{X}^!\mathsf{X}^!\, o_1)$$

## The MTDFA



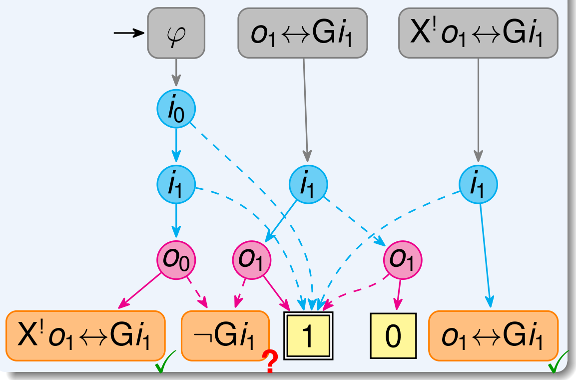
## The Game Interpretation



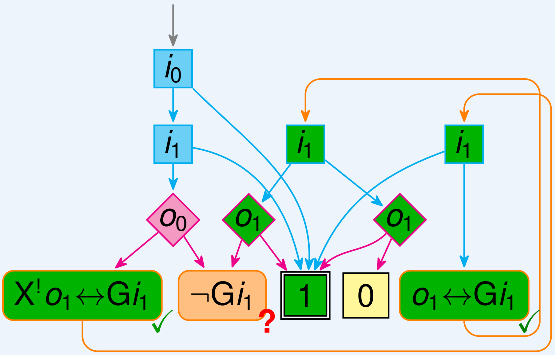
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \wedge \mathsf{G} i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^! \mathsf{X}^! o_1)$$

## The MTDFA



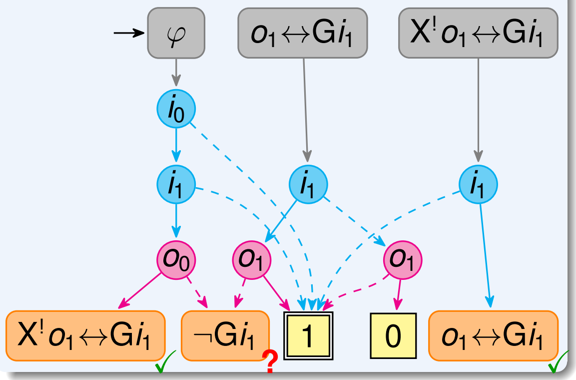
## The Game Interpretation



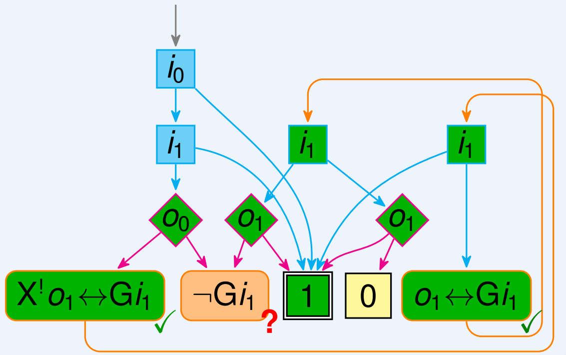
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \land \mathsf{G}\, i_1) \leftrightarrow (o_0 \land \mathsf{X}^!\mathsf{X}^!\, o_1)$$

## The MTDFA



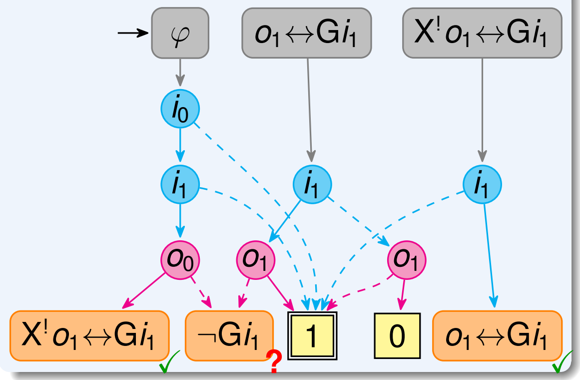
## The Game Interpretation



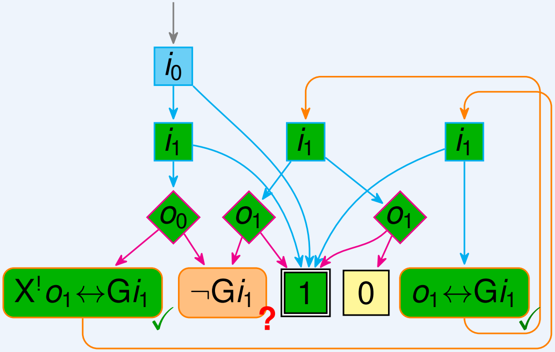
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \land \mathsf{G} i_1) \leftrightarrow (o_0 \land \mathsf{X}^! \mathsf{X}^! o_1)$$

## The MTDFA



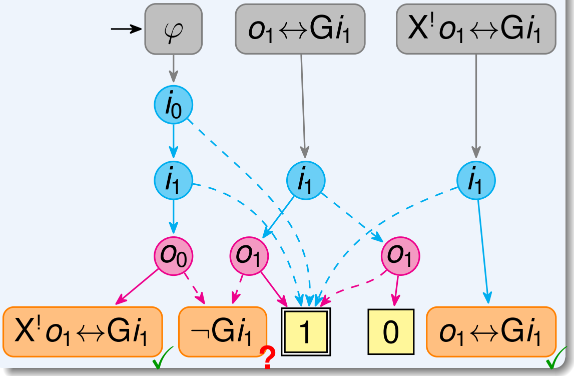
## The Game Interpretation



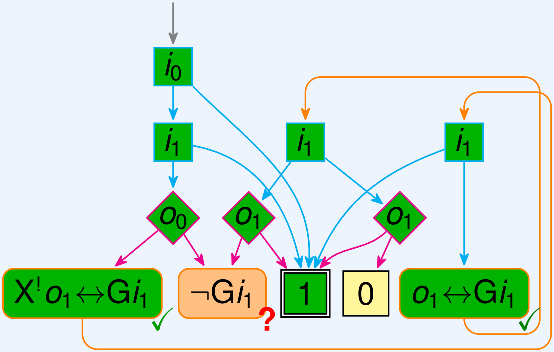
# Building the MTDFA & Solving the Game On-The-Fly

$$\varphi = (i_0 \land \mathsf{G} i_1) \leftrightarrow (o_0 \land \mathsf{X}^! \mathsf{X}^! o_1)$$

## The MTDFA



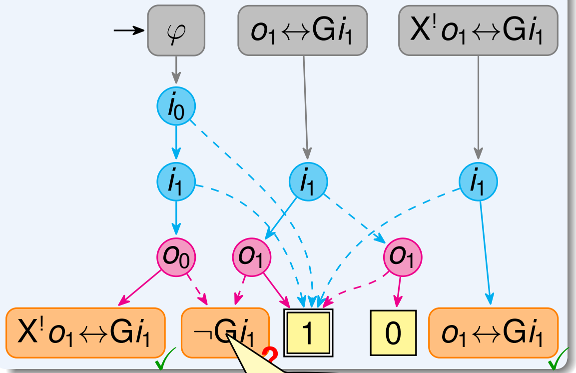
## The Game Interpretation



# Building the MTDFA & Solving the Game On-The-Fly

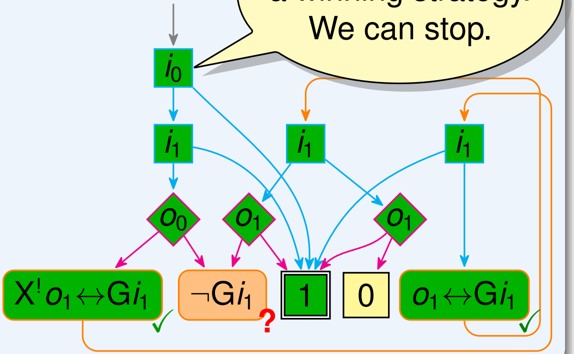
$$\varphi = (i_0 \land \mathsf{G} i_1) \leftrightarrow (o_0 \land \mathsf{X}^! \mathsf{X}^! o_1)$$

## The MTDFA



Never developed.

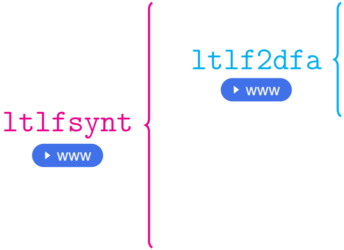
## The Game Inte



Output player has a winning strategy. We can stop.

# LTL$_f$ Synthesis with MTDFA

Implemented in two new tools distributed with Spot 2.14 ▸ www



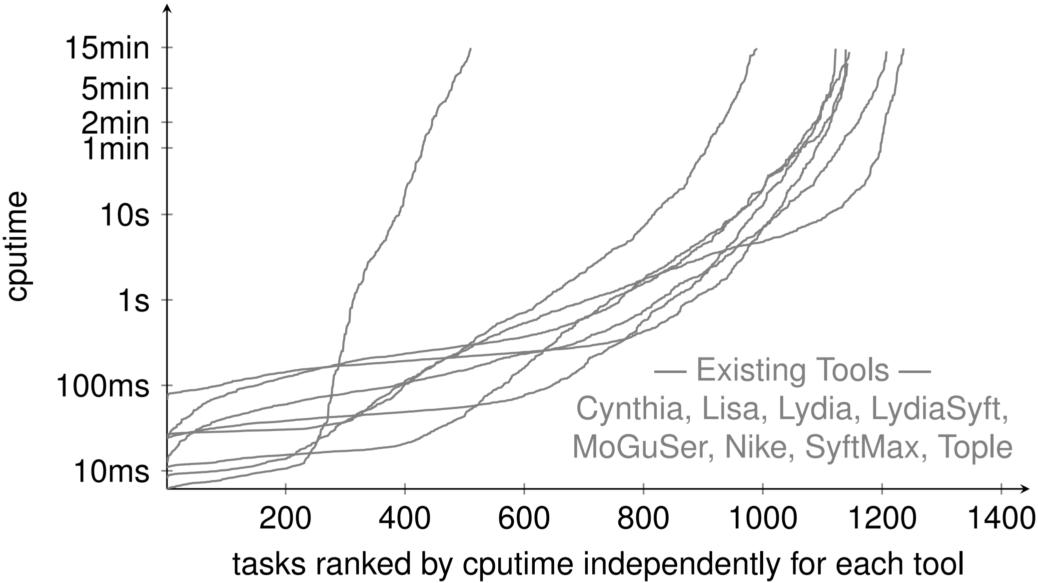
`ltlf2dfa` ▸ www

`ltlfsynt` ▸ www

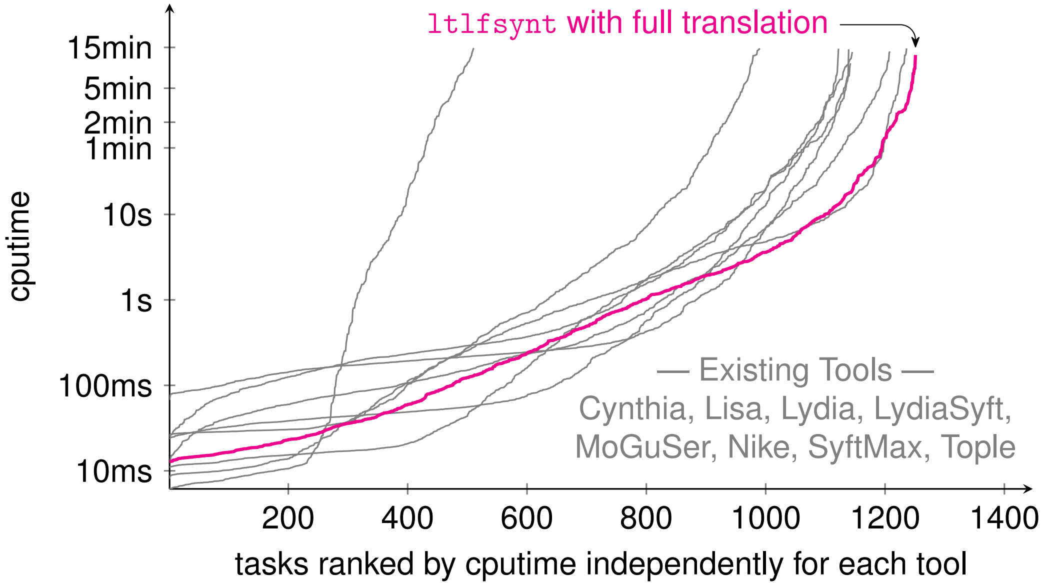
**What we Suggest**

1. Direct translation from LTL$_f$ to MTBDD, building the MTDFA one state at a time. ✓
2. Solving the game on the MTDFA directly. ✓
3. Doing those on-the-fly. ✓

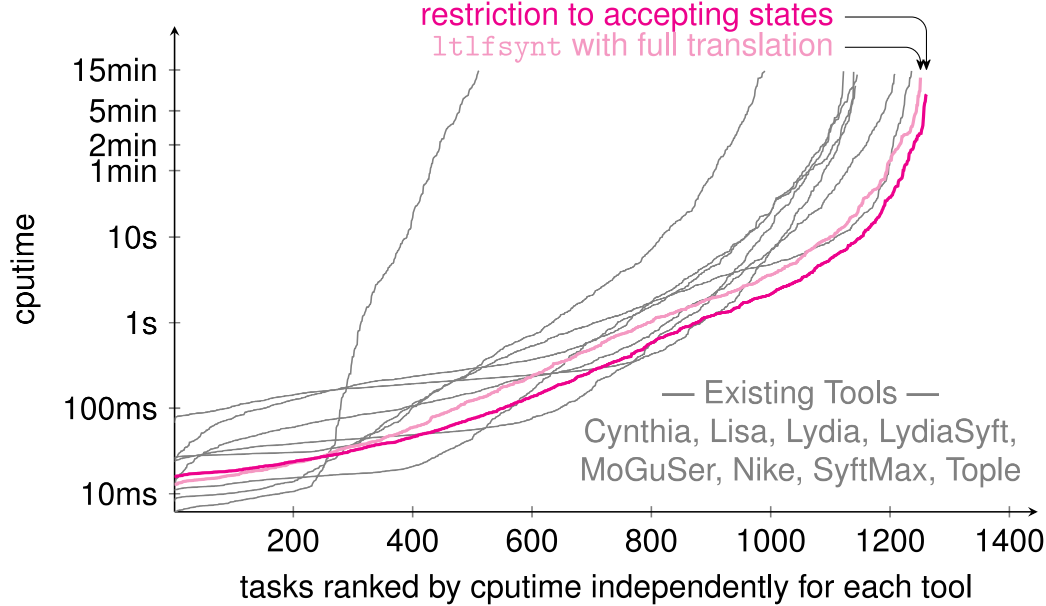
# A Benchmark (Specifications from SyntComp)



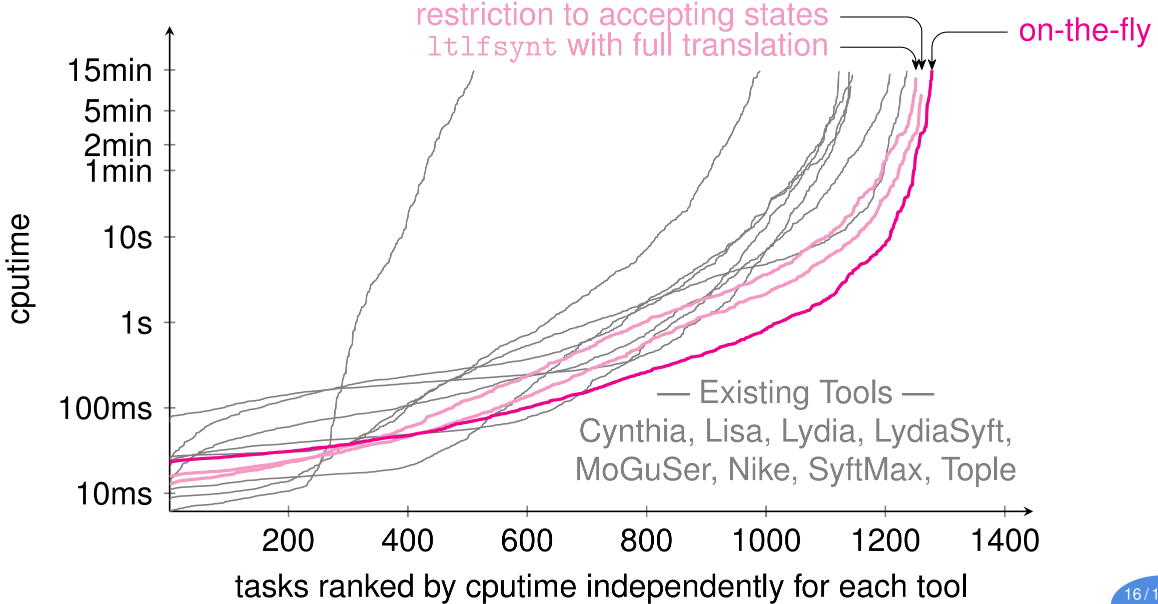
# A Benchmark (Specifications from SyntComp)



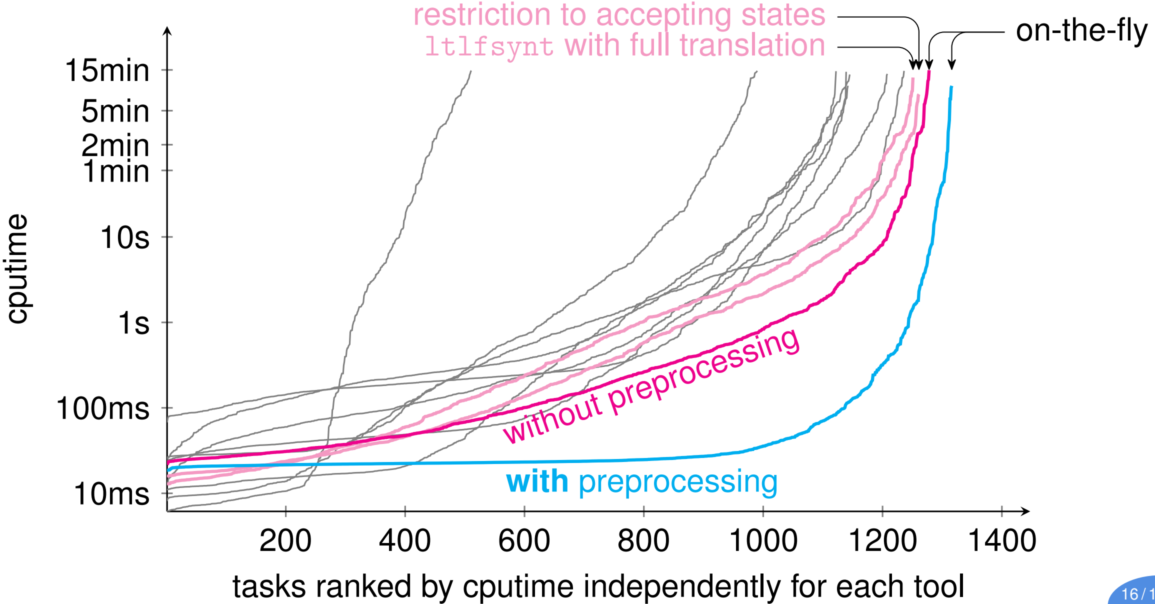
# A Benchmark (Specifications from SyntComp)



# A Benchmark (Specifications from SyntComp)



# A Benchmark (Specifications from SyntComp)



# Conclusion

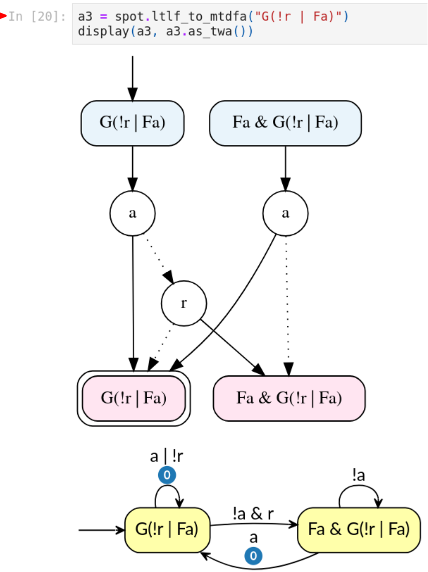
## Efficient LTL$_f$ tools to build upon

Distributed with Spot 2.14:

- `ltlf2dfa`
- `ltlfsynt` (🏆 won SyntComp’25)
- C++ & Python APIs available

## Ideas to take away

- MTBDDs are great for deterministic automata with propositional alphabets.
- Such automata can be interpreted as games at the level of MTBDD nodes (deciding one proposition at a time).



```
In [20]: a3 = spot.ltlf_to_mtdfa("G(!r | Fa)")
         display(a3, a3.as_twa())
```

# Warp Zone

1. Title

2. Reactive Synthesis 3. Text-Book Approach 4. Stopping on Final States 5. MTBDD/MTDFA 6. Outline

7. MTDFA as game

9. $\mathrm{LTL}_f \longrightarrow$MTBDD example 10. $\mathrm{LTL}_f \longrightarrow$MTBDD formal 11. $\mathrm{LTL}_f \longrightarrow$MTDFA 12. Accepting Terminals

14. On-the-Fly

16. Benchmark

17. Conclusion

19. Preprocessings 20. Propositional Equivalence

# Preprocessings

## Simplify specification using polarity of propositions

- If an output proposition is always positive/negative in the specification, replace it by $\top/\bot$.
- If an input proposition is always positive/negative in the specification, replace it by $\bot/\top$.

Example: $\mathsf{G}(i \rightarrow o)$ becomes $\mathsf{G}(\top \rightarrow \top) \equiv \top$.

## Use cheap rewritings to reduce number of MTBDD operations

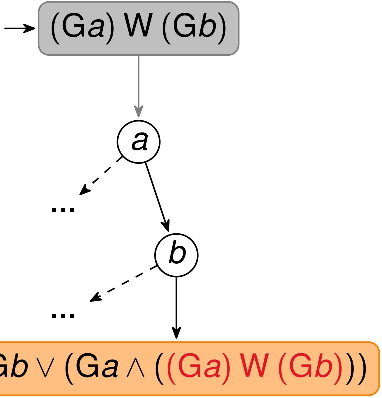
$$\mathsf{X}\alpha \land \mathsf{X}\beta \rightsquigarrow \mathsf{X}(\alpha \land \beta), \qquad (\alpha \rightarrow \beta) \land (\alpha \rightarrow \gamma) \rightsquigarrow \alpha \rightarrow (\beta \land \gamma), \qquad \ldots$$

## Split specification into output disjoint specifications when possible

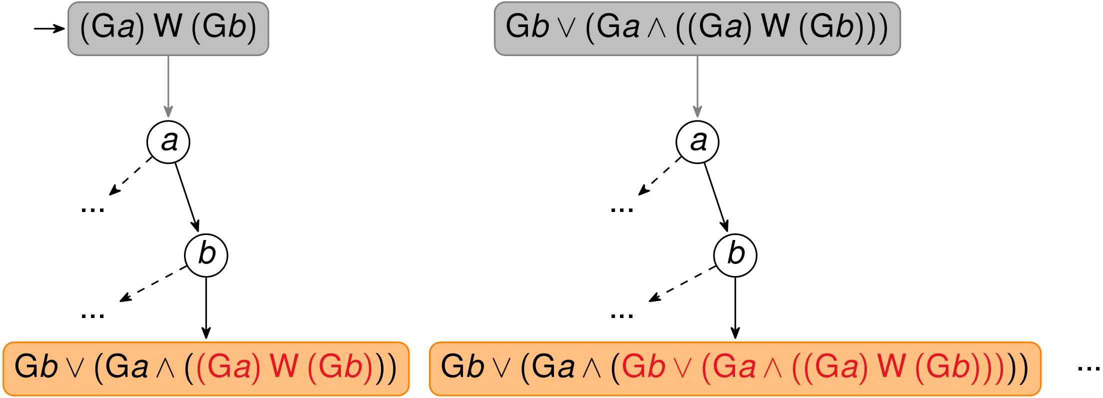
If $\Psi_1$ and $\Psi_2$ are output-disjoint, and admit controllers that agree on accepting lengths, then $\Psi_1 \land \Psi_2$ can be solved as two independent problems.

▸ issue 610

# Propositional Equivalence



# Propositional Equivalence



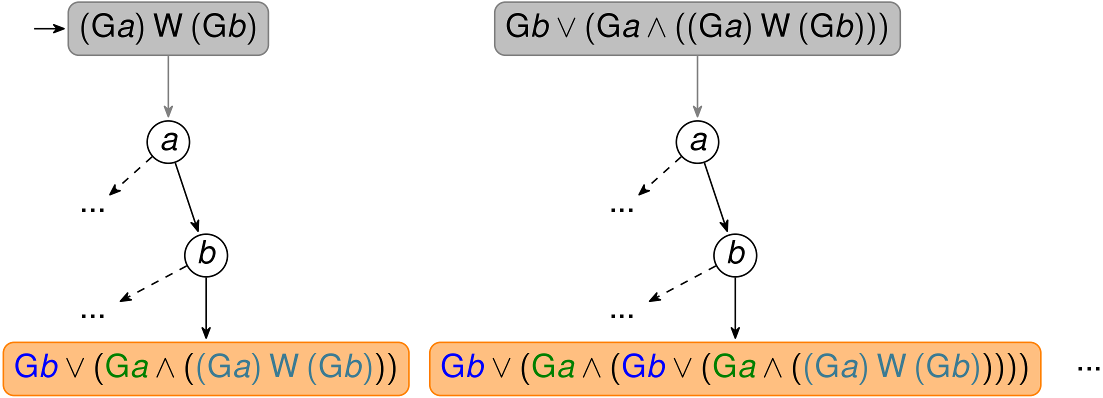
$(\mathrm{G}a)\ \mathrm{W}\ (\mathrm{G}b)$

$\mathrm{G}b \vee (\mathrm{G}a \wedge ((\mathrm{G}a)\ \mathrm{W}\ (\mathrm{G}b)))$

$\mathrm{G}b \vee (\mathrm{G}a \wedge ((\mathrm{G}a)\ \mathrm{W}\ (\mathrm{G}b)))$

$\mathrm{G}b \vee (\mathrm{G}a \wedge (\mathrm{G}b \vee (\mathrm{G}a \wedge ((\mathrm{G}a)\ \mathrm{W}\ (\mathrm{G}b)))))$

# Propositional Equivalence



$(\mathrm{G}a)\ \mathrm{W}\ (\mathrm{G}b)$

$\mathrm{G}b \vee (\mathrm{G}a \wedge ((\mathrm{G}a)\ \mathrm{W}\ (\mathrm{G}b)))$

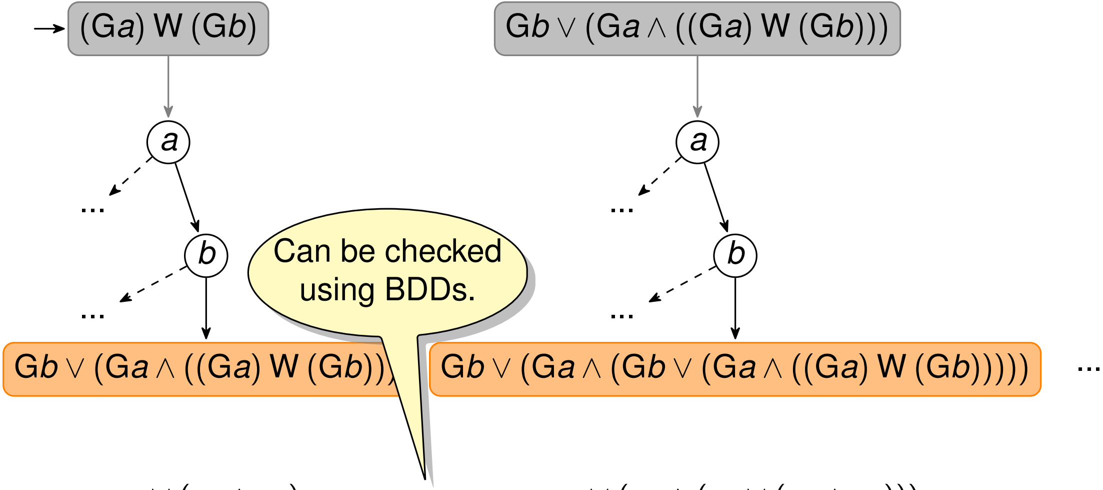
$\mathrm{G}b \vee (\mathrm{G}a \wedge ((\mathrm{G}a)\ \mathrm{W}\ (\mathrm{G}b)))$

$\mathrm{G}b \vee (\mathrm{G}a \wedge (\mathrm{G}b \vee (\mathrm{G}a \wedge ((\mathrm{G}a)\ \mathrm{W}\ (\mathrm{G}b)))))$

$p_3 \vee (p_2 \wedge p_1)$

$p_3 \vee (p_2 \wedge (p_3 \vee (p_2 \wedge p_1)))$

# Propositional Equivalence



$(\mathrm{G}a)\,\mathrm{W}\,(\mathrm{G}b)$

$\mathrm{G}b \vee (\mathrm{G}a \wedge ((\mathrm{G}a)\,\mathrm{W}\,(\mathrm{G}b)))$

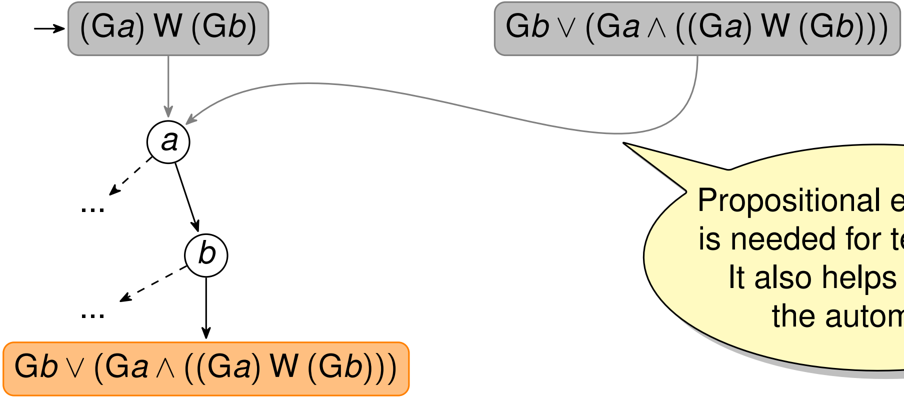
$\mathrm{G}b \vee (\mathrm{G}a \wedge ((\mathrm{G}a)\,\mathrm{W}\,(\mathrm{G}b)))$

$\mathrm{G}b \vee (\mathrm{G}a \wedge (\mathrm{G}b \vee (\mathrm{G}a \wedge ((\mathrm{G}a)\,\mathrm{W}\,(\mathrm{G}b)))))$

Can be checked using BDDs.

$$p_3 \vee (p_2 \wedge p_1) \equiv p_3 \vee (p_2 \wedge (p_3 \vee (p_2 \wedge p_1)))$$

# Propositional Equivalence



Propositional equivalence
is needed for termination.
It also helps reducing
the automaton.