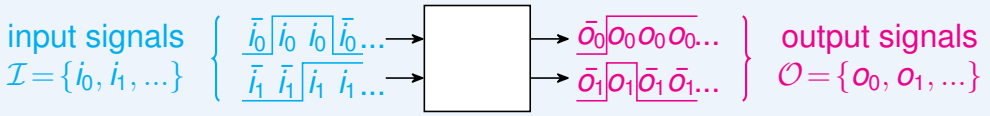
Reactive Synthesis using MTBDDs

Alexandre Duret-Lutz Dror Fried
Giuseppe De Giacomo Lucas M. Tabajara
Marcin Jurdzinski Moshe Y. Vardi Nadav Alon
Nir Piterman Shufang Zhu Supratik Chakraborty

Outline

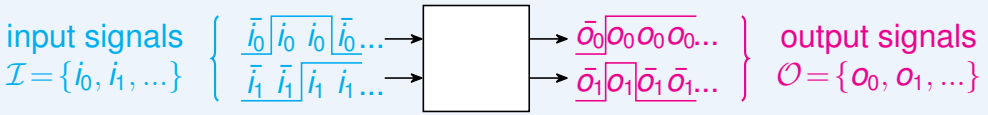
Reactive Synthesis in a Nutshell

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



Reactive Synthesis in a Nutshell

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



The reactive synthesis problems

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

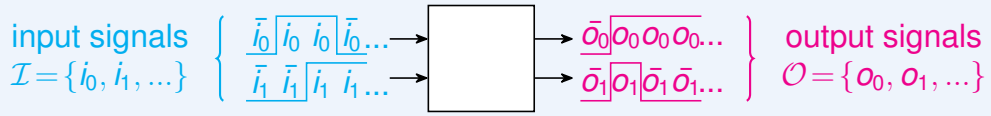
Realizability: decide if a controller exists;

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

our focus

Reactive Synthesis in a Nutshell

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



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 φ

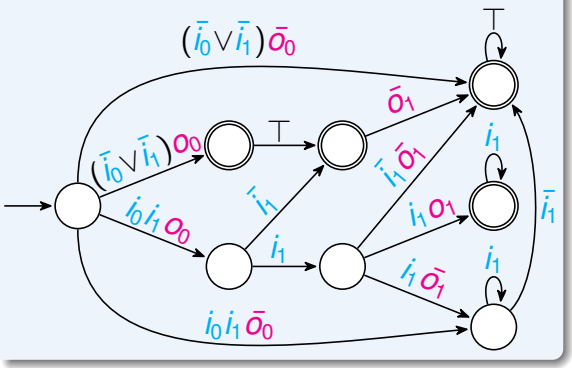
$(i_0 \wedge \mathsf{G} i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^! \mathsf{X}^! o_1)$

Text-Book Approach

1. LTL$_f$ specification φ

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

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

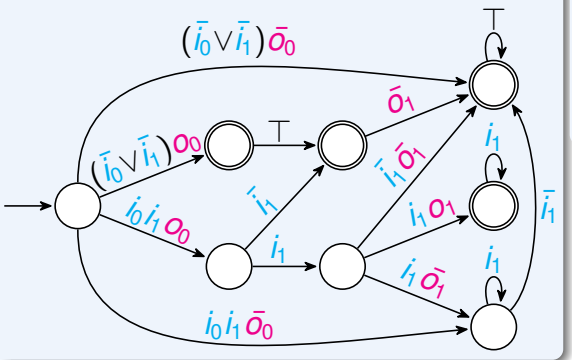


Text-Book Approach

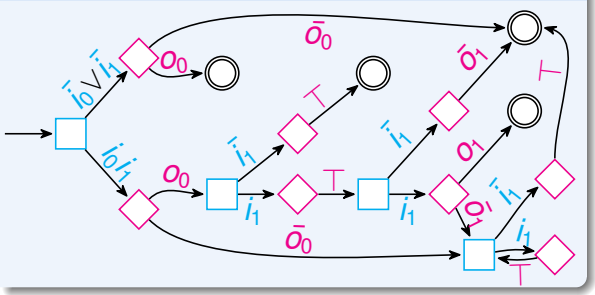
1. LTL$_f$ specification φ

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

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



3. Make it a Reachability Game

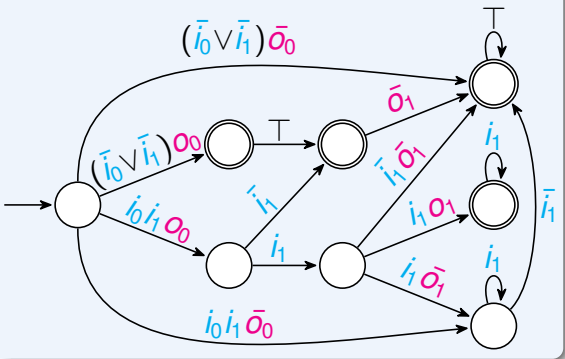


Text-Book Approach

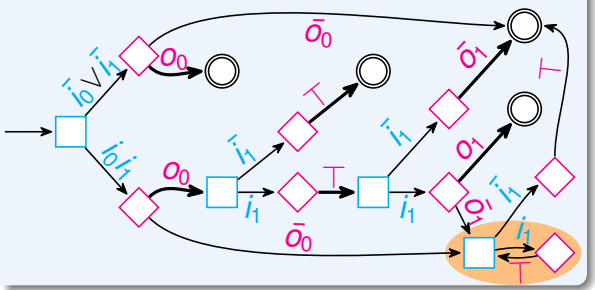
1. LTL$_f$ specification φ

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

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



3. Make it a Reachability Game



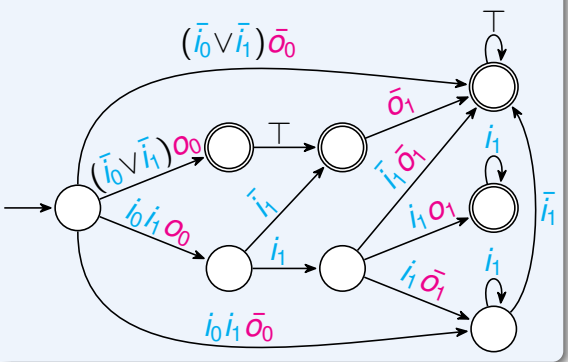
4. Solve it

Text-Book Approach

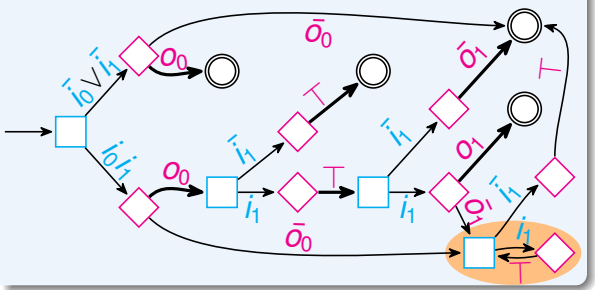
1. LTL$_f$ specification φ

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

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

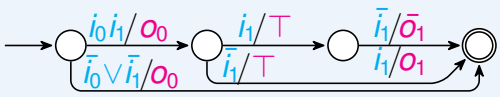


3. Make it a Reachability Game



4. Solve it

5. Extract a Controller if Desired



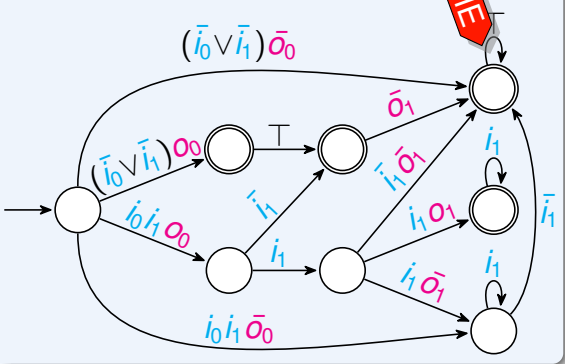
Text-Book Approach

1. LTL$_f$ specification φ

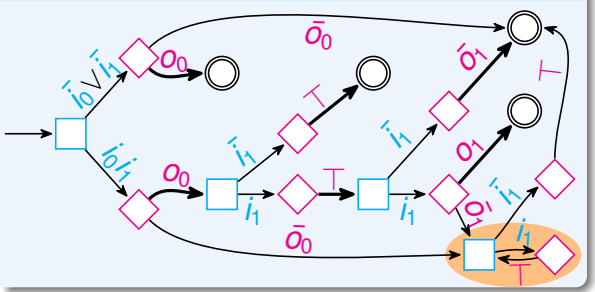
$(i_0 \wedge \mathsf{G} i_1) \leftrightarrow (o_0 \wedge \mathsf{X}^! \mathsf{X}^! o_1)$

2-EXPTIME

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

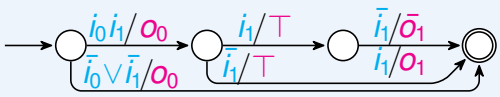


3. Make it a Reachability Game



4. Solve it

5. Extract a Controller if Desired

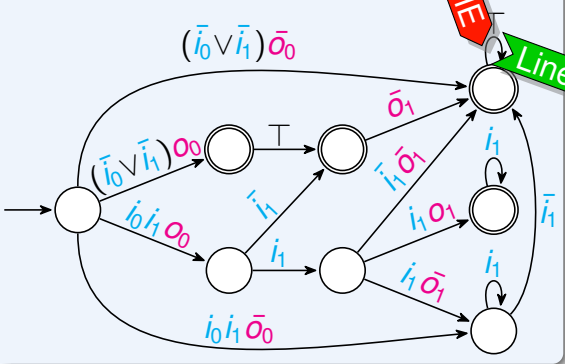


Text-Book Approach

1. LTL$_f$ specification φ

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

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

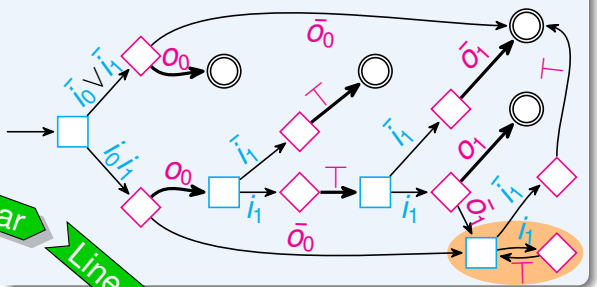


2-EXPTIME

Linear

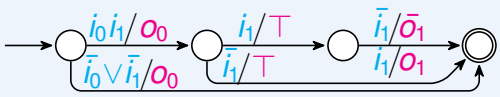
Linear

3. Make it a Reachability Game



4. Solve it

5. Extract a Controller if Desired

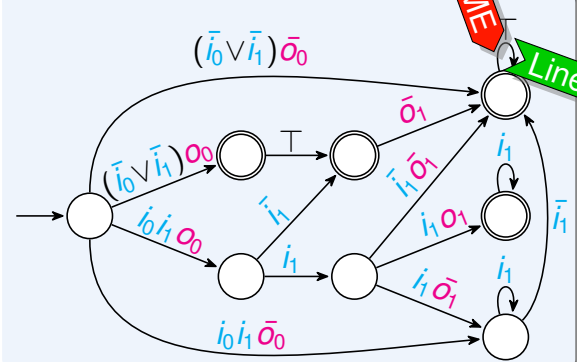


Text-Book Approach

1. LTL$_f$ specification φ

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

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



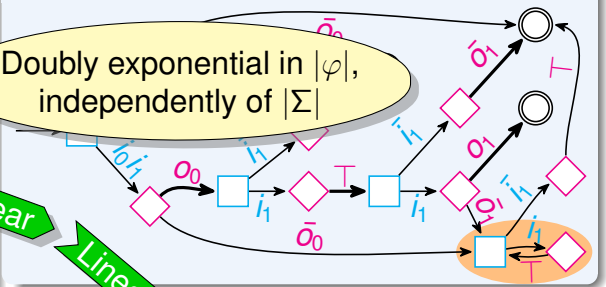
2-EXPTIME

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

Linear

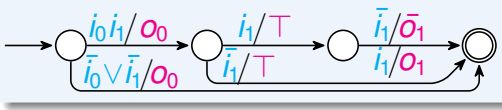
Linear

3. Make it a Reachability Game



4. Solve it

5. Extract a Controller if Desired

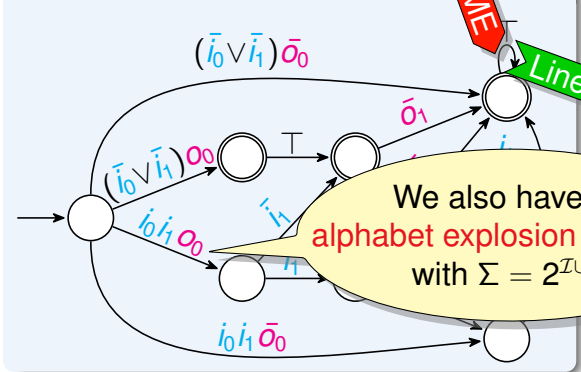


Text-Book Approach

1. LTL$_f$ specification φ

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

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



2-EXPTIME

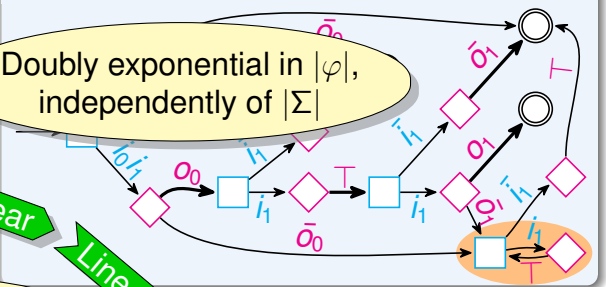
Doubly exponential in $|\varphi|$, independently of $|\Sigma|$

Linear

Linear

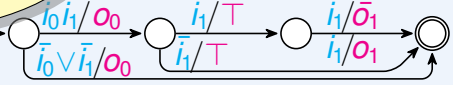
We also have an alphabet explosion problem, with $\Sigma = 2^{\mathcal{I} \cup \mathcal{O}}$

3. Make it a Reachability Game



Solve it

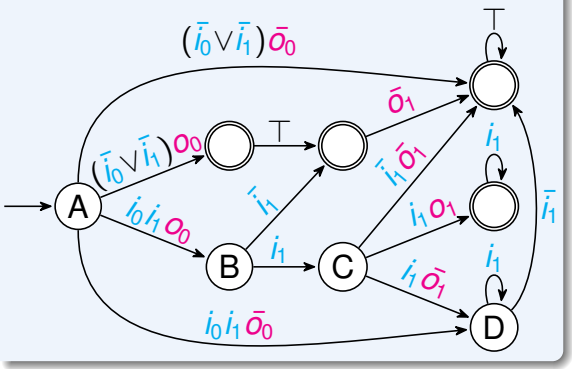
Extract a Controller if Desired



Stopping the DFA Construction on Final States

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

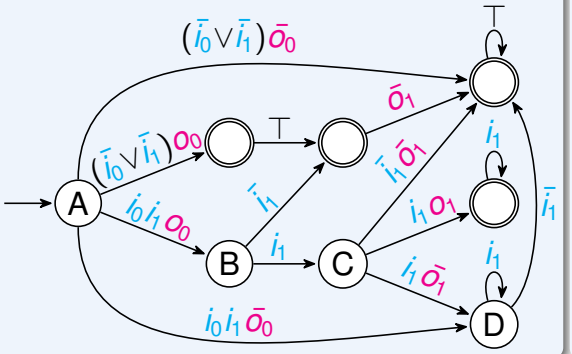
Overkill



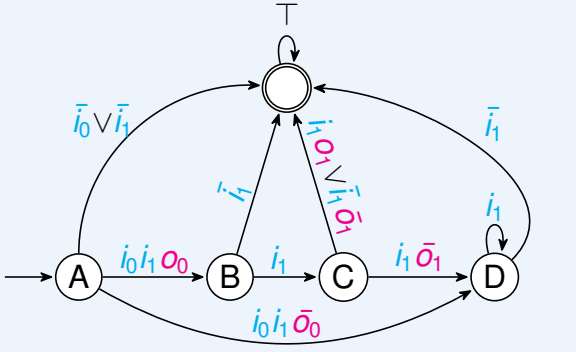
Stopping the DFA Construction on Final States

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

Overkill

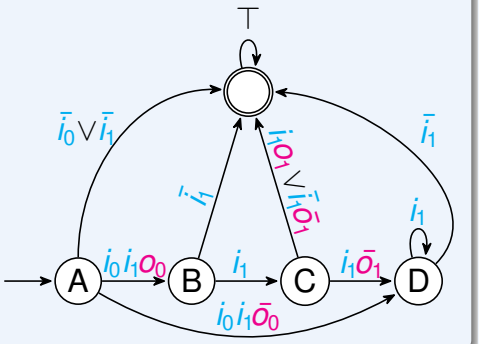


Better



Fighting Alphabet Explosion with MTBDDs

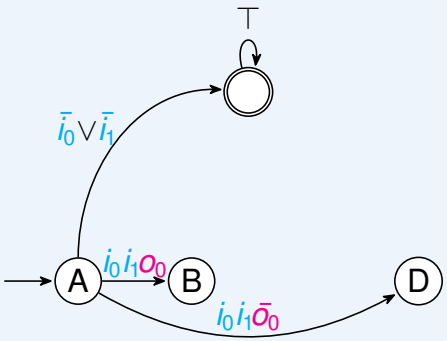
Explicit representation



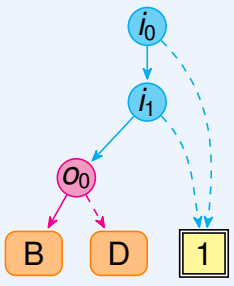
Semi-symbolic representation: MTBDD

Fighting Alphabet Explosion with MTBDDs

Explicit representation



Semi-symbolic representation: MTBDD

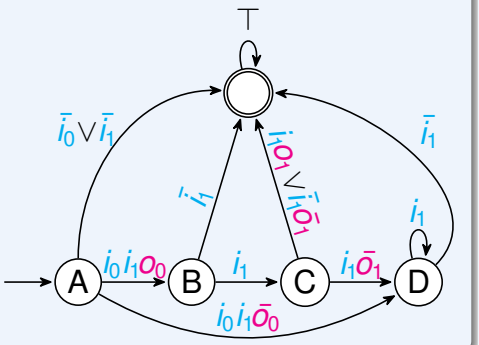


MTBDD representing outgoing edges of (A) (unique for a given variable order)

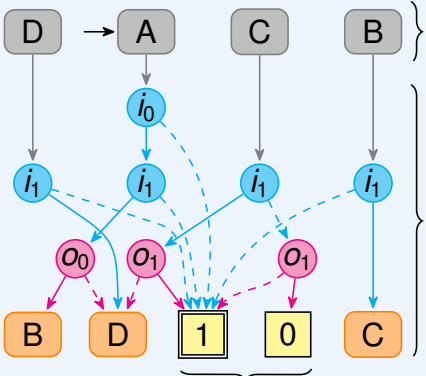
true BDD node as accepting sink

Fighting Alphabet Explosion with MTBDDs

Explicit representation



Semi-symbolic representation: **MTDFA**



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 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.

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 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.

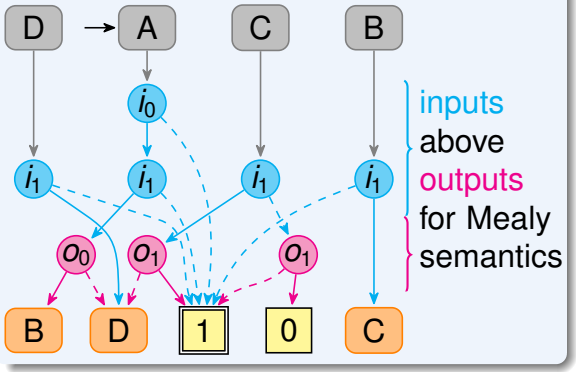
What we Suggest

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

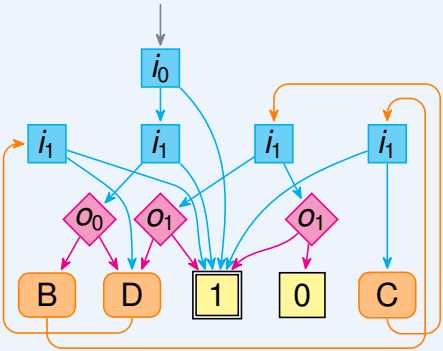
Seeing the MTDFA as a Game Arena

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

The MTDFA



The Game Interpretation

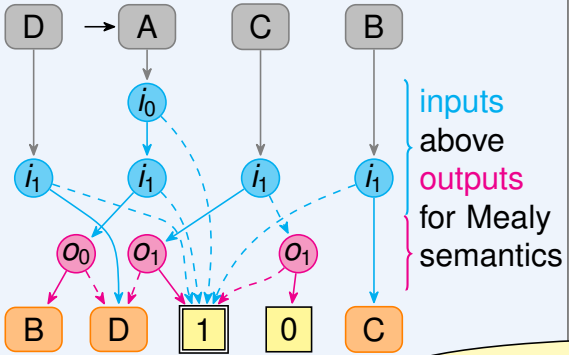


Seeing the MTDFA as a Game Arena

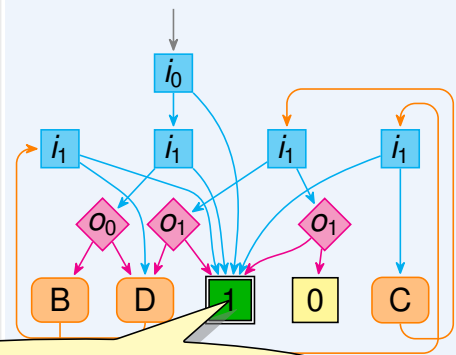
Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

Actually stores reversed edges.

The MTDFA



The Game Interpretation



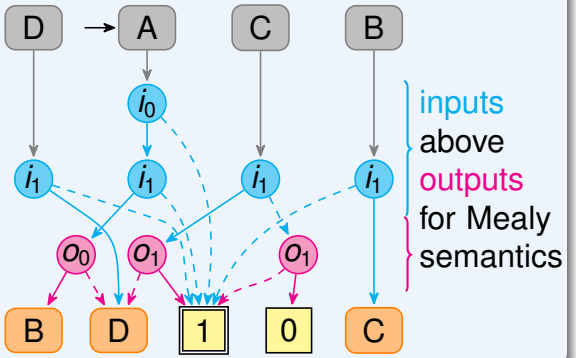
Solve by backpropagation from 1.

Seeing the MTDFA as a Game Arena

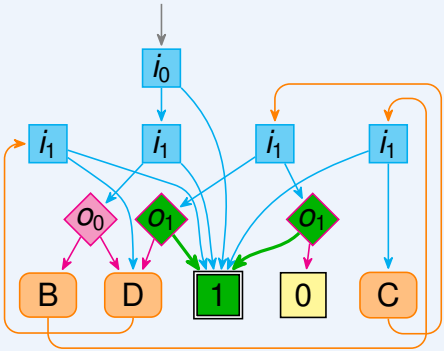
Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

Actually stores reversed edges.

The MTDFA



The Game Interpretation

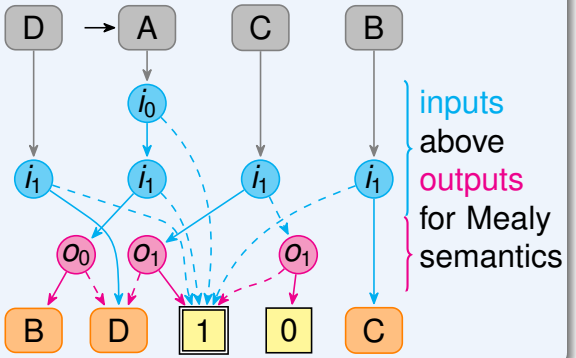


Seeing the MTDFA as a Game Arena

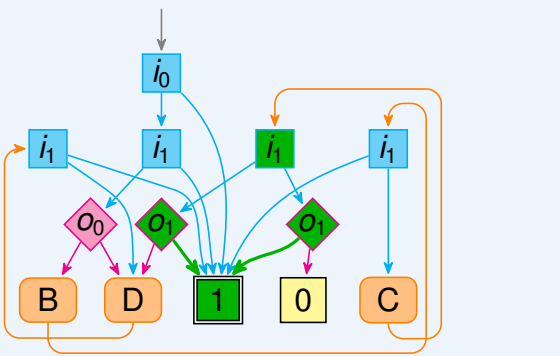
Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

Actually stores reversed edges.

The MTDFA



The Game Interpretation

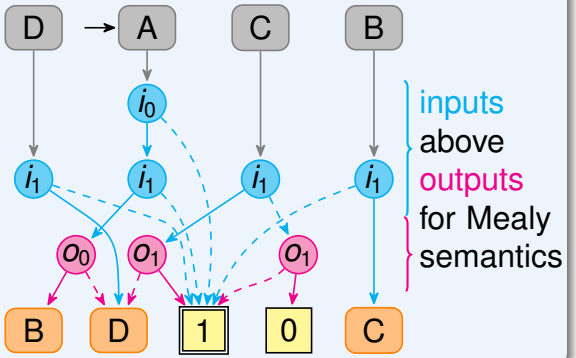


Seeing the MTDFA as a Game Arena

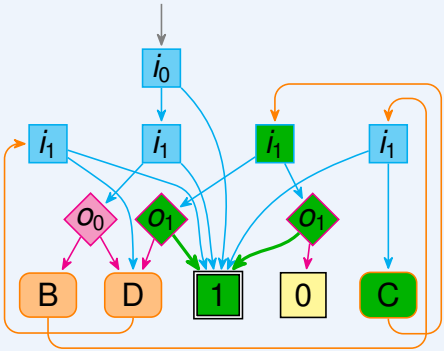
Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

Actually stores reversed edges.

The MTDFA



The Game Interpretation

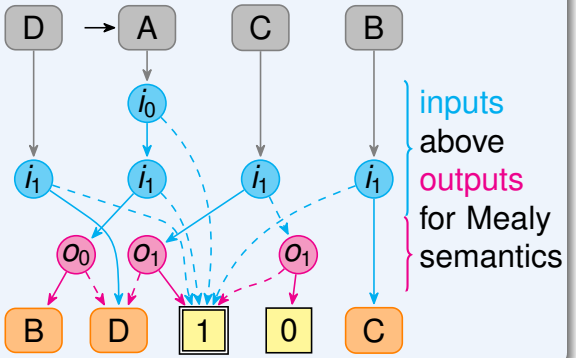


Seeing the MTDFA as a Game Arena

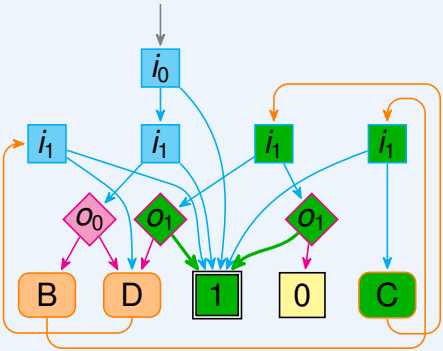
Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

Actually stores reversed edges.

The MTDFA



The Game Interpretation

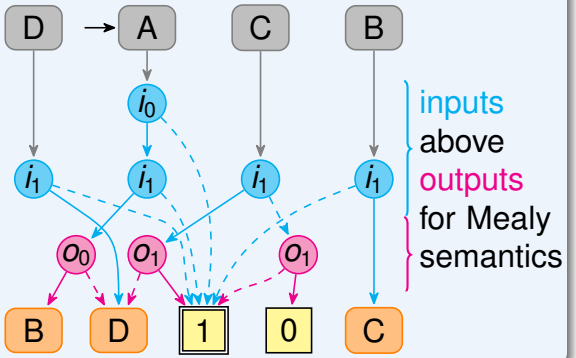


Seeing the MTDFA as a Game Arena

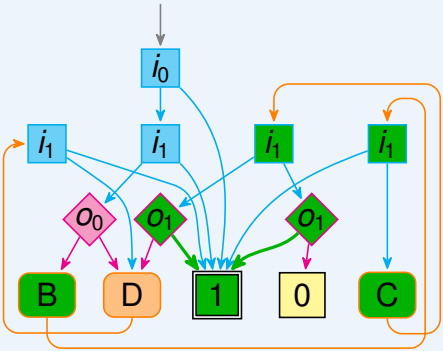
Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

Actually stores reversed edges.

The MTDFA



The Game Interpretation

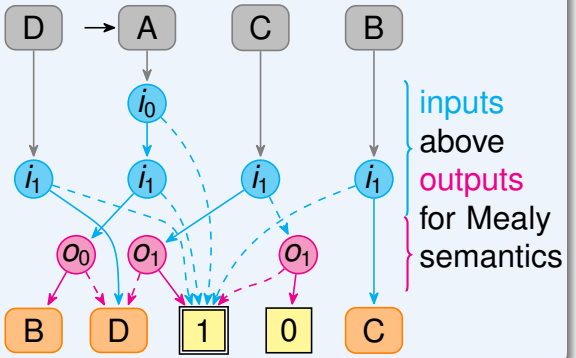


Seeing the MTDFA as a Game Arena

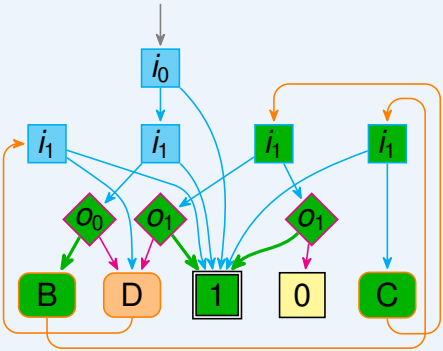
Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

Actually stores reversed edges.

The MTDFA



The Game Interpretation

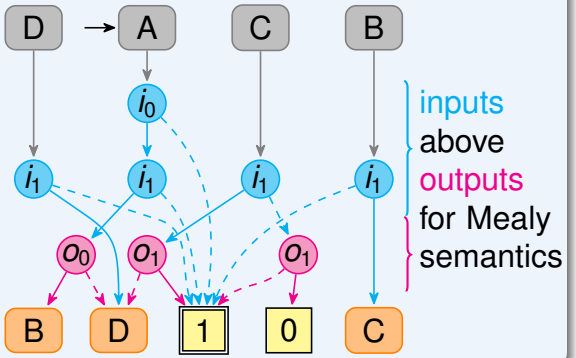


Seeing the MTDFA as a Game Arena

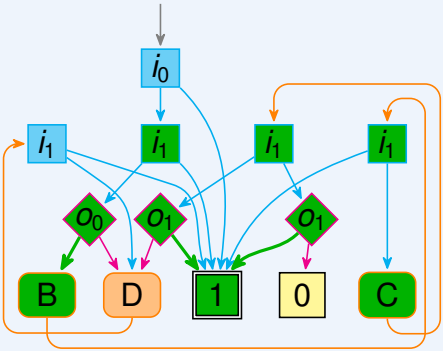
Turn input/output nodes into universal/existential vertices.
Order input/output variables according to the desired semantics (

Actually stores reversed edges.

The MTDFA



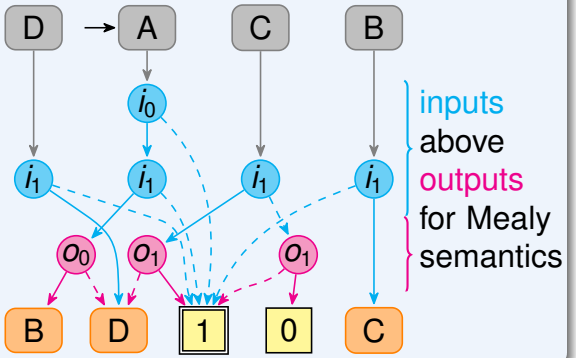
The Game Interpretation



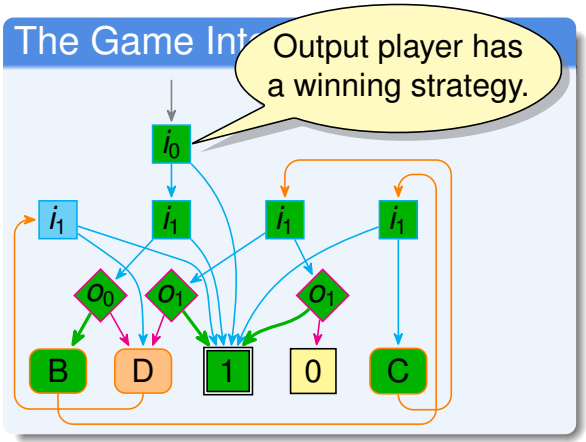
Seeing the MTDFA as a Game Arena

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

The MTDFA



The Game Int



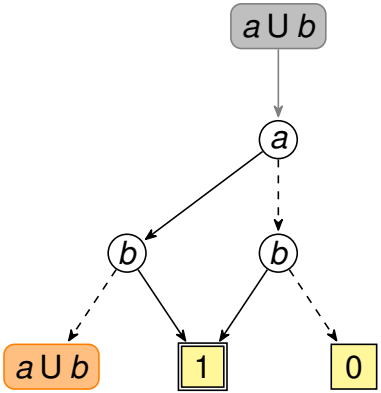
LTL_f Synthesis with MTDFA

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 $a \,\mathsf{U}\, b$

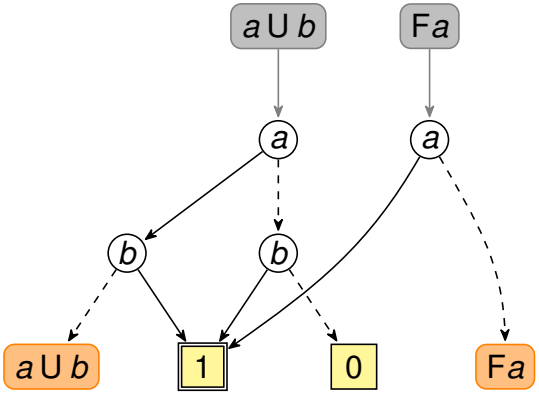


This is a deterministic representation of the *next normal form* (XNF):

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

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 $\mathsf{F}a$.

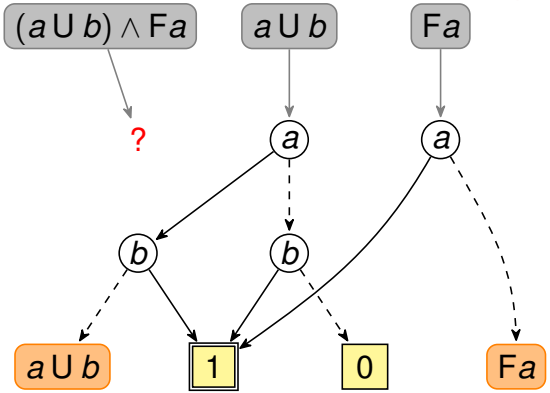


This is a deterministic representation of the *next normal form* (XNF):

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

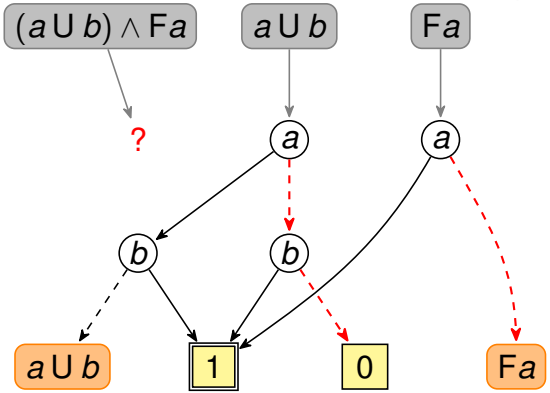
From LTL_f to MTBDD (not yet MTDFA)

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)

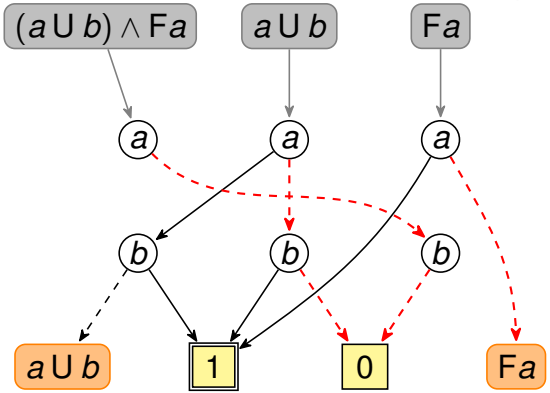
Use LTL$_f$ formulas as terminals.
Assume we know an MTBDD for the successors of $a \,\mathsf{U}\, b$, and another for $\mathsf{F}a$.
We want to compute an MTBDD for $(a \,\mathsf{U}\, b) \wedge \mathsf{F}a$:



Classical BDD `apply` procedure, but combine terminals with "$\wedge$".

From LTL_f to MTBDD (not yet MTDFA)

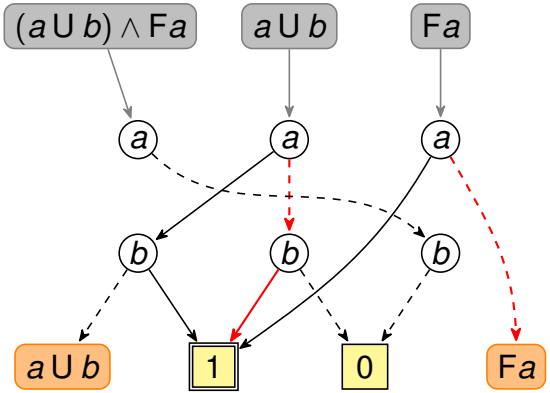
Use LTL_f formulas as terminals.
Assume we know an MTBDD for the successors of $a \,\mathsf{U}\, b$, and another for $\mathsf{F}a$.
We want to compute an MTBDD for $(a \,\mathsf{U}\, b) \wedge \mathsf{F}a$:



Classical BDD `apply` procedure, but combine terminals with "$\wedge$".

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

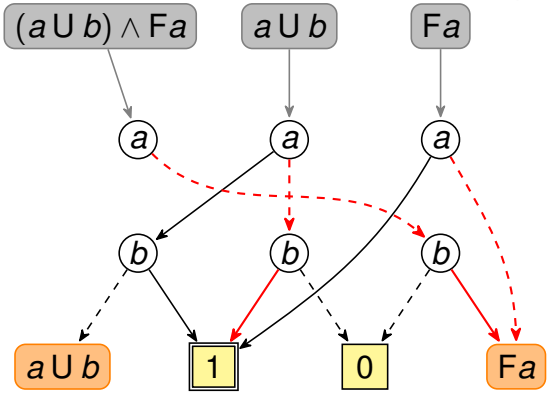
Use LTL$_f$ formulas as terminals.
Assume we know an MTBDD for the successors of $a \,\mathsf{U}\, b$, and another for $\mathsf{F}a$.
We want to compute an MTBDD for $(a \,\mathsf{U}\, b) \wedge \mathsf{F}a$:



Classical BDD `apply` procedure, but combine terminals with "$\wedge$".

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

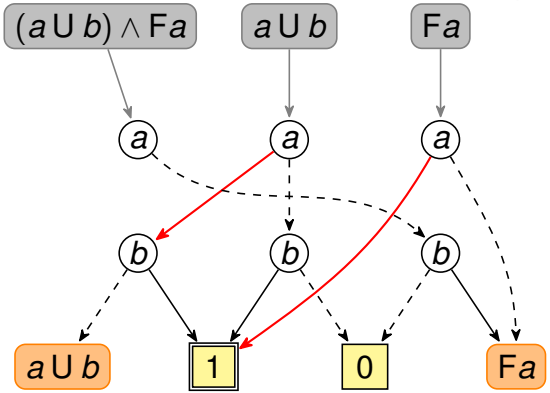
Use LTL$_f$ formulas as terminals.
Assume we know an MTBDD for the successors of $a \,\mathsf{U}\, b$, and another for $\mathsf{F}a$.
We want to compute an MTBDD for $(a \,\mathsf{U}\, b) \wedge \mathsf{F}a$:



Classical BDD `apply` procedure, but combine terminals with "$\wedge$".

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

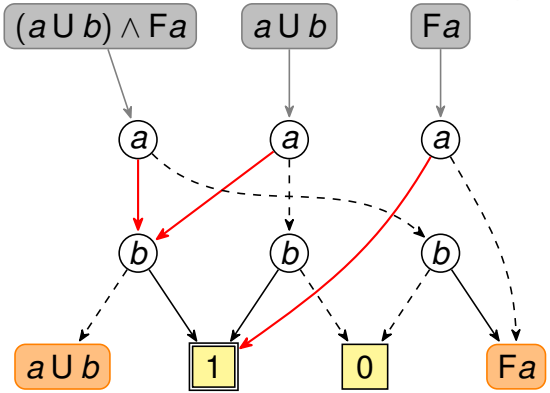
Use LTL$_f$ formulas as terminals.
Assume we know an MTBDD for the successors of $a \,\mathsf{U}\, b$, and another for $\mathsf{F}a$.
We want to compute an MTBDD for $(a \,\mathsf{U}\, b) \wedge \mathsf{F}a$:



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

From LTL_f to MTBDD (not yet MTDFA)

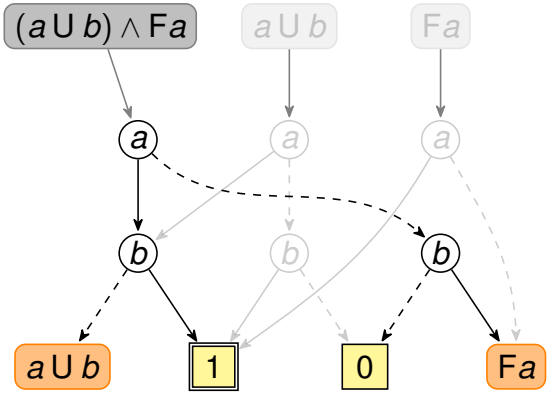
Use LTL_f formulas as terminals.
Assume we know an MTBDD for the successors of $a \,\mathsf{U}\, b$, and another for $\mathsf{F}a$.
We want to compute an MTBDD for $(a \,\mathsf{U}\, b) \wedge \mathsf{F}a$:



Classical BDD `apply` procedure, but combine terminals with "$\wedge$".
Leaves 0 and 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 $(a \,\mathsf{U}\, b) \wedge \mathsf{F}a$:

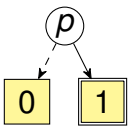


MTBDDs for subformulas are cached (i.e., not thrown away) in case they are needed later during the construction.

From LTL$_f$ to MTBDD: Formal Definition

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

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

$\mathrm{tr}(p) =$ 

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

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

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

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

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

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

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

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

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

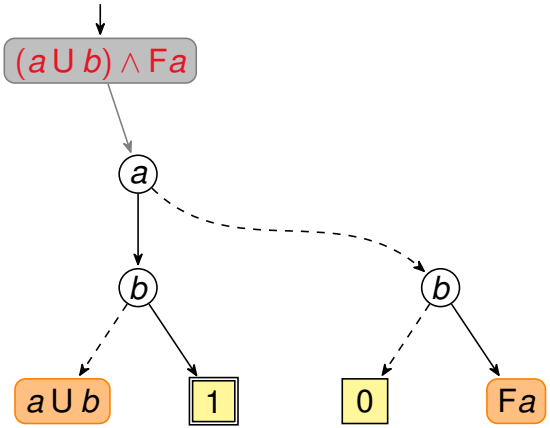
With the convention that $\alpha \wedge \beta = \alpha \wedge \beta$, $\alpha \vee \beta = \alpha \vee \beta$, ...

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: $\mathrm{tr}((a \,\mathsf{U}\, b) \wedge \mathsf{F}a)$.

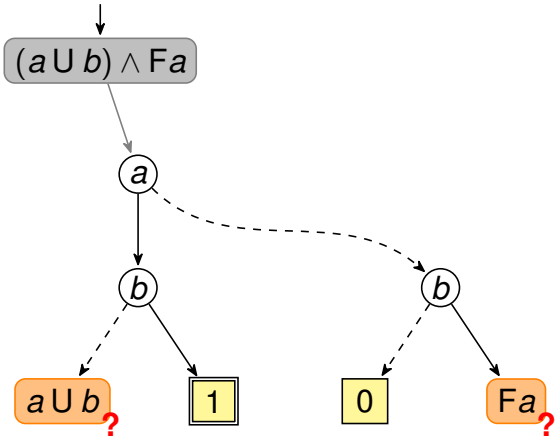
From LTL$_f$ to MTDFA: Build States One at a Time



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

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

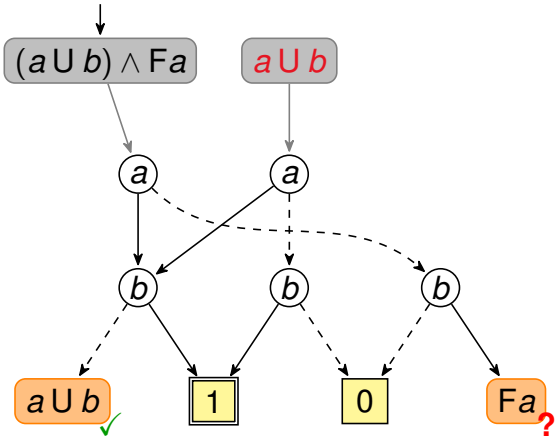
From LTL$_f$ to MTDFA: Build States One at a Time



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:

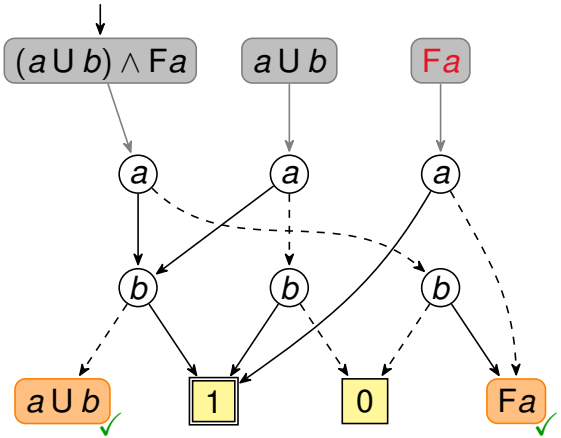
From LTL$_f$ to MTDFA: Build States One at a Time



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)

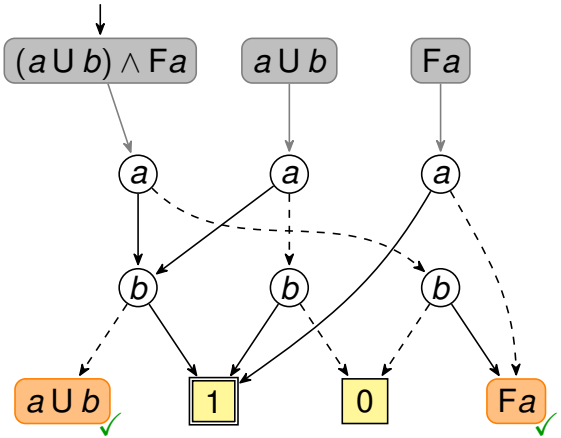
From LTL$_f$ to MTDFA: Build States One at a Time



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

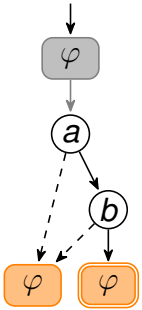


To translate $(a \,\mathsf{U}\, b) \wedge \mathsf{F}a$:

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)
3. Done.

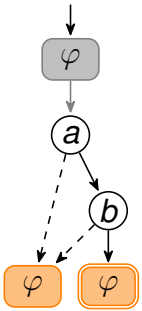
Quick Note on Accepting Terminals

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

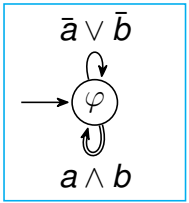


Quick Note on Accepting Terminals

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

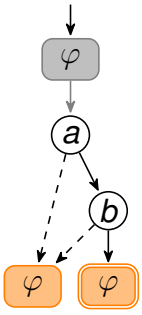


If you interpret the MTBDD roots as states, you get a transition-based DFA:

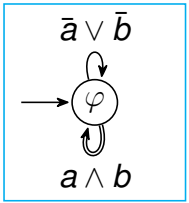


Quick Note on Accepting Terminals

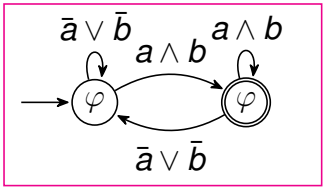
Our use of accepting terminals differs from Mona's implementation of MTDFAs.



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

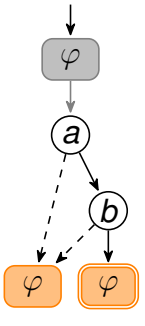


If you interpret the MTBDD terminals as states, you get a state-based DFA:

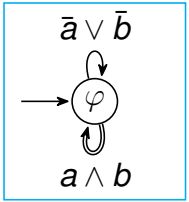


Quick Note on Accepting Terminals

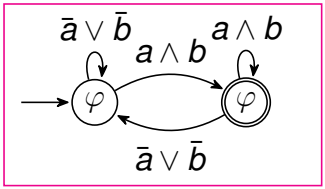
Our use of accepting terminals differs from Mona's implementation of MTDFAs.



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 1.

LTL_f Synthesis with MTDFA

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. next slide

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

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

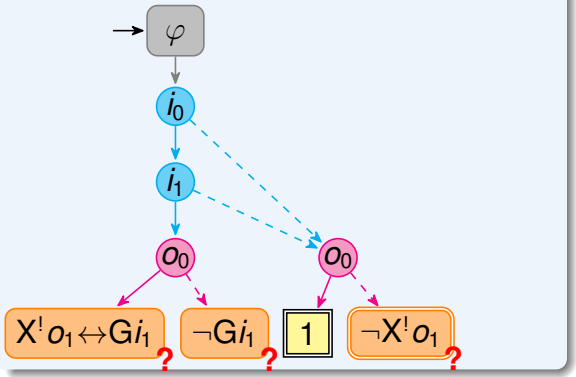
The MTDFA

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

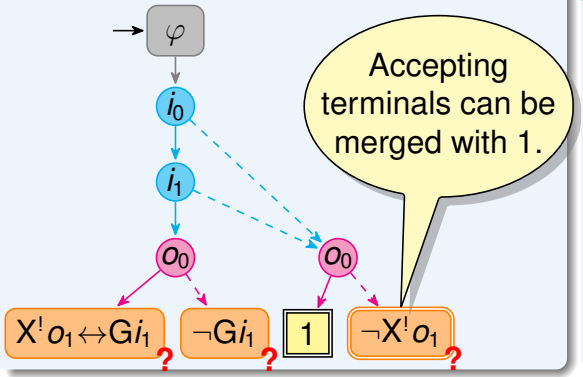


The Game Interpretation

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

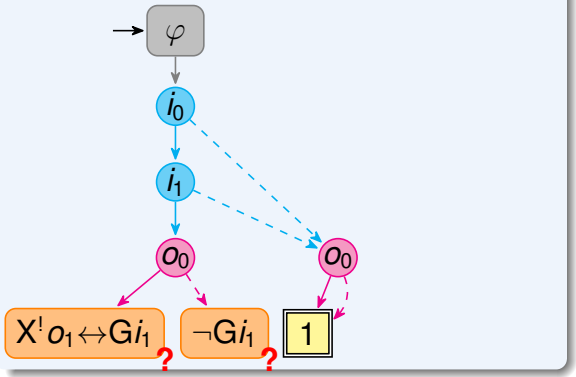


The Game Interpretation

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

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

The MTDFA

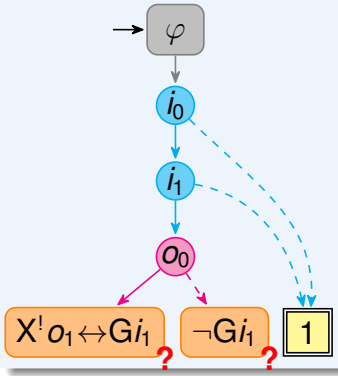


The Game Interpretation

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

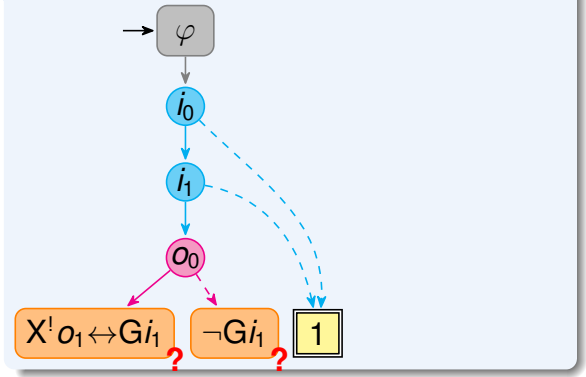


The Game Interpretation

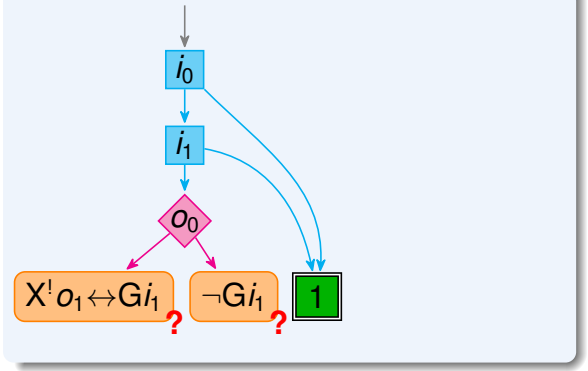
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



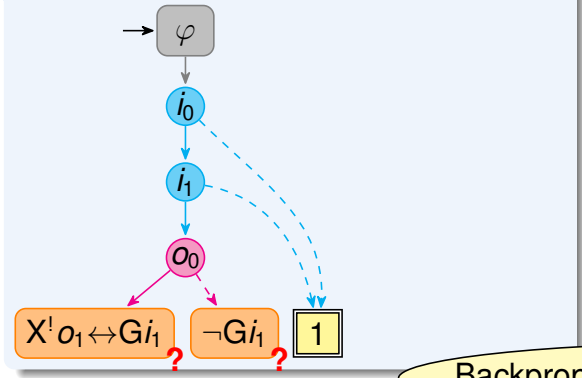
The Game Interpretation



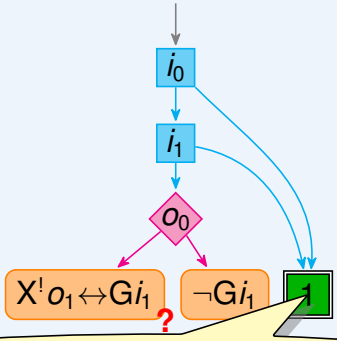
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



The Game Interpretation

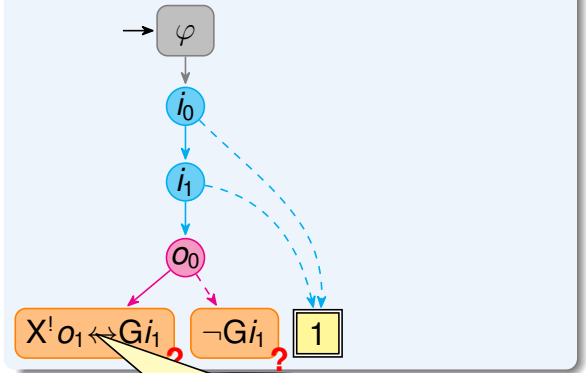


Backpropagation runs during construction.

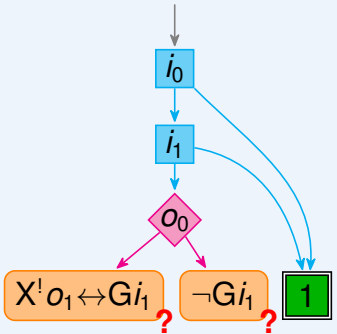
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



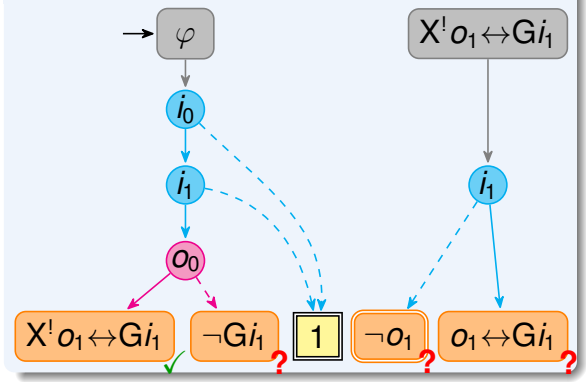
The Game Interpretation



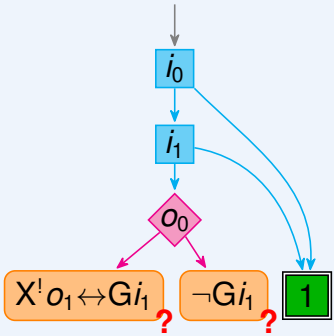
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



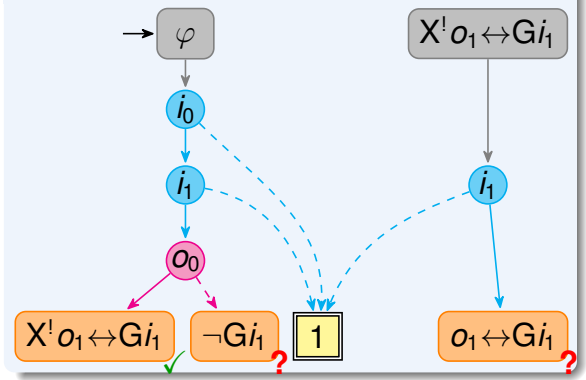
The Game Interpretation



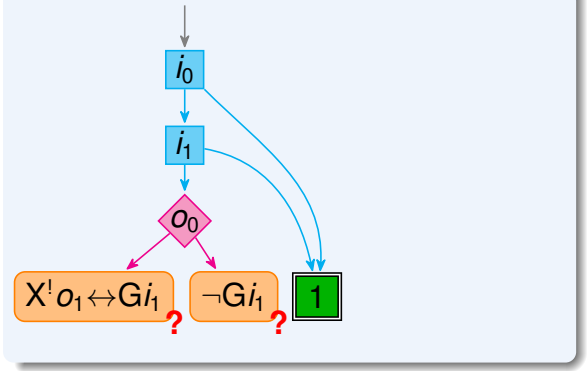
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



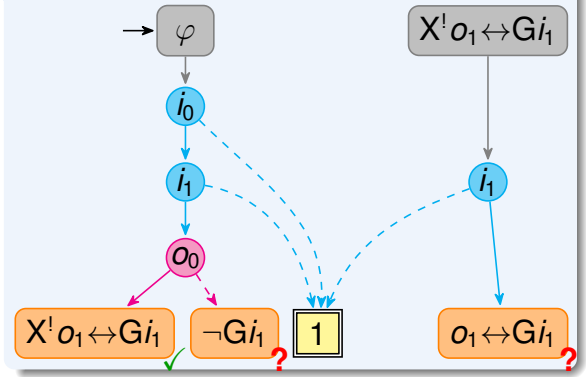
The Game Interpretation



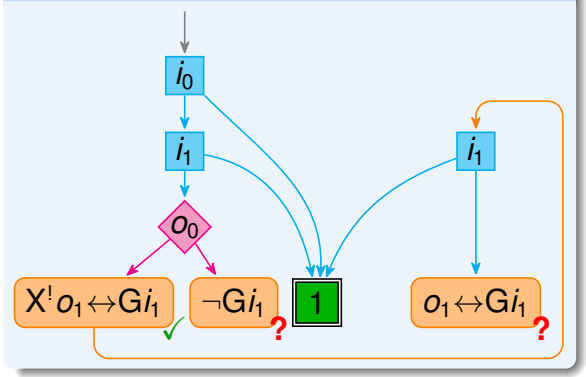
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



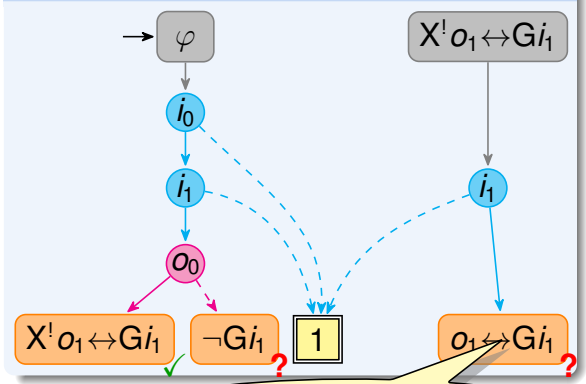
The Game Interpretation



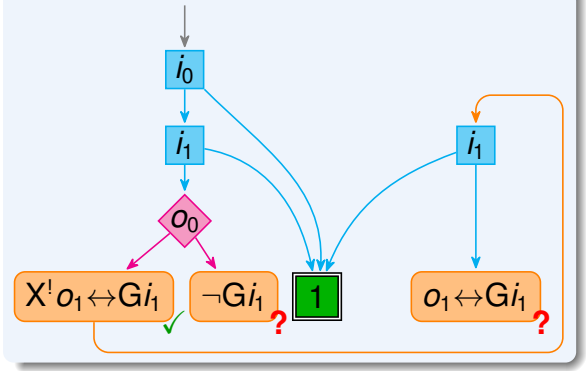
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



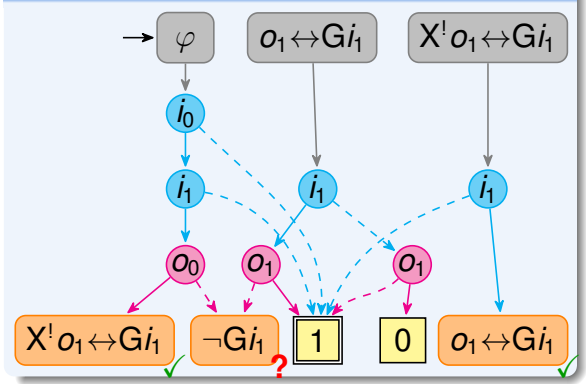
The Game Interpretation



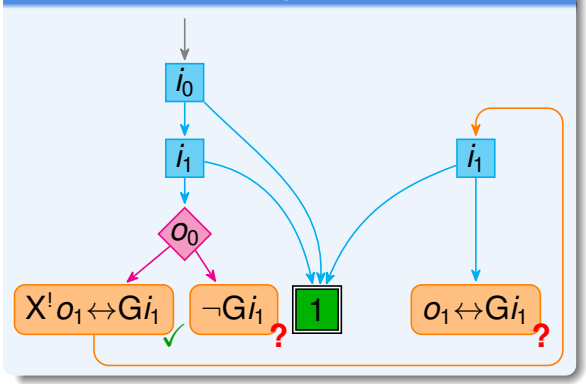
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



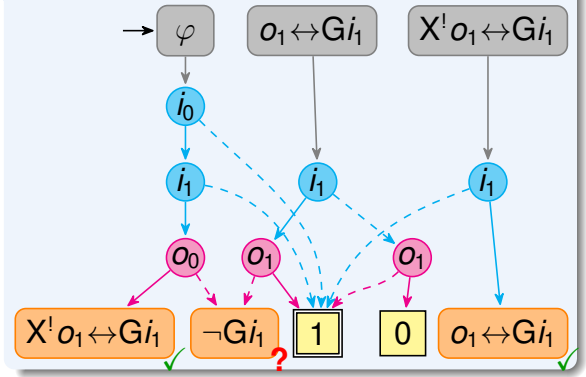
The Game Interpretation



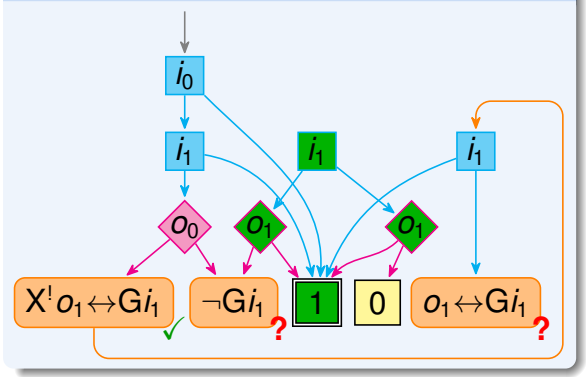
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



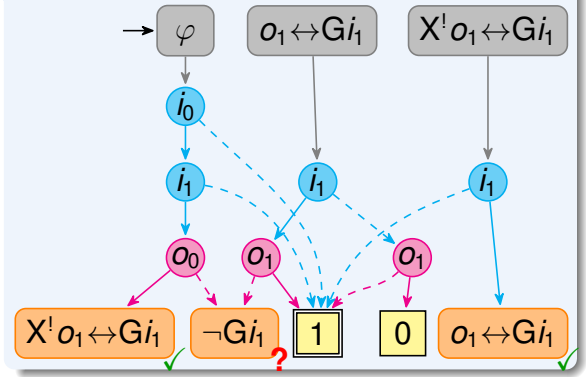
The Game Interpretation



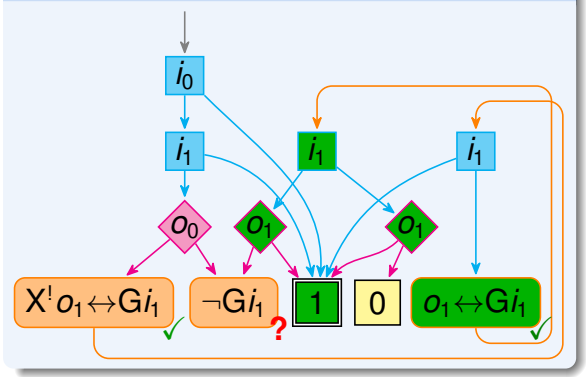
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



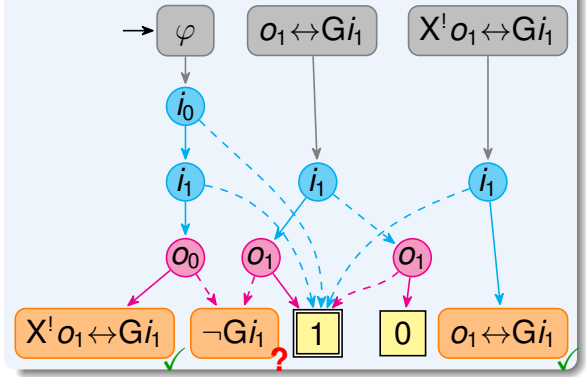
The Game Interpretation



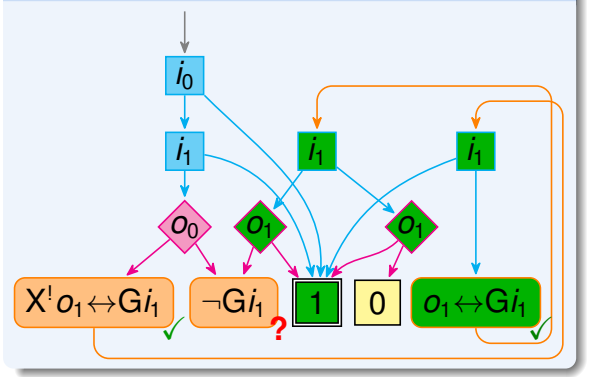
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



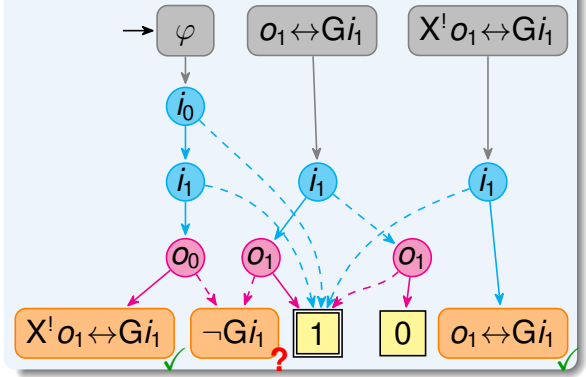
The Game Interpretation



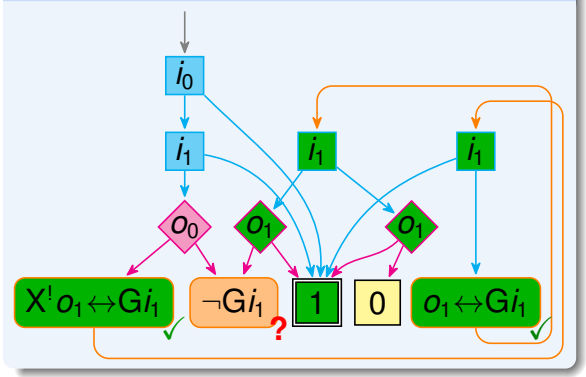
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



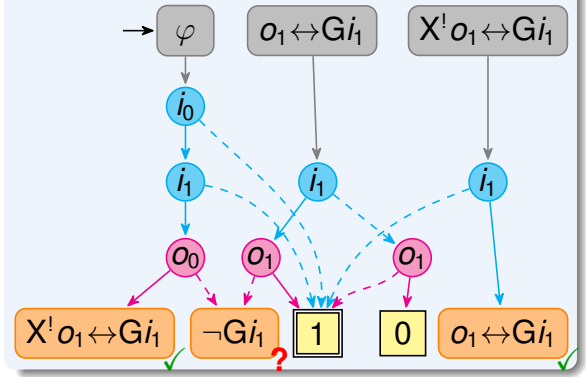
The Game Interpretation



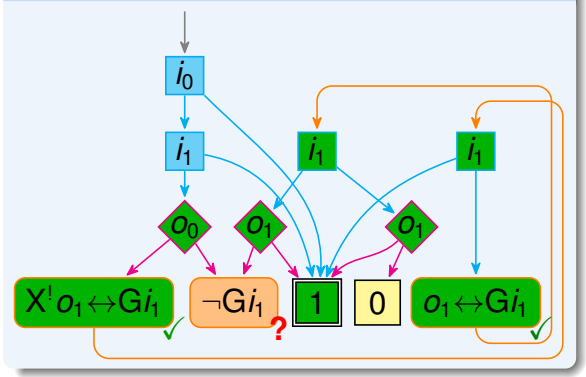
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



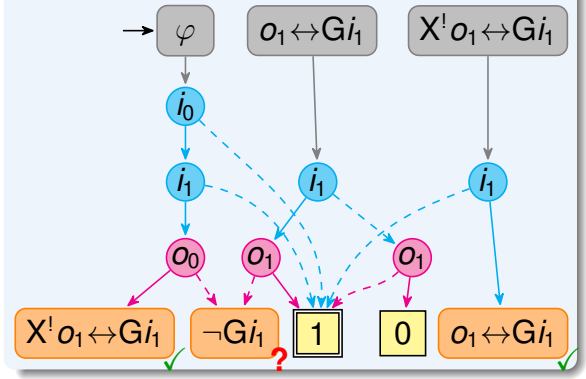
The Game Interpretation



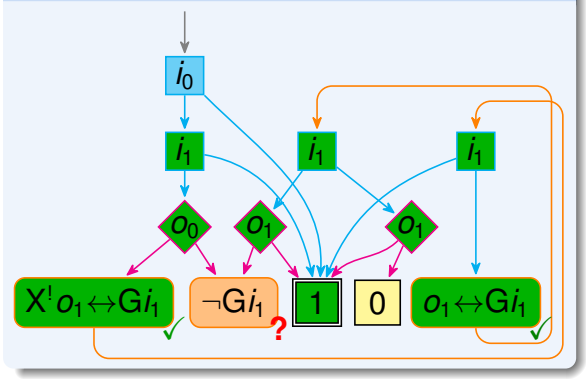
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA



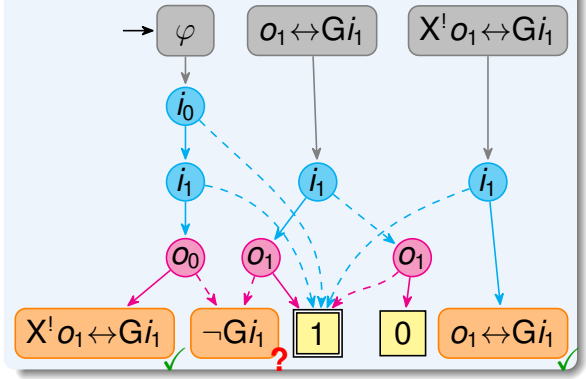
The Game Interpretation



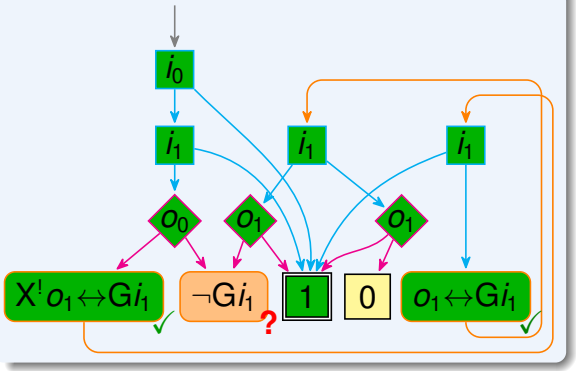
Building the MTDFA & Solving the Game On-The-Fly

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

The MTDFA

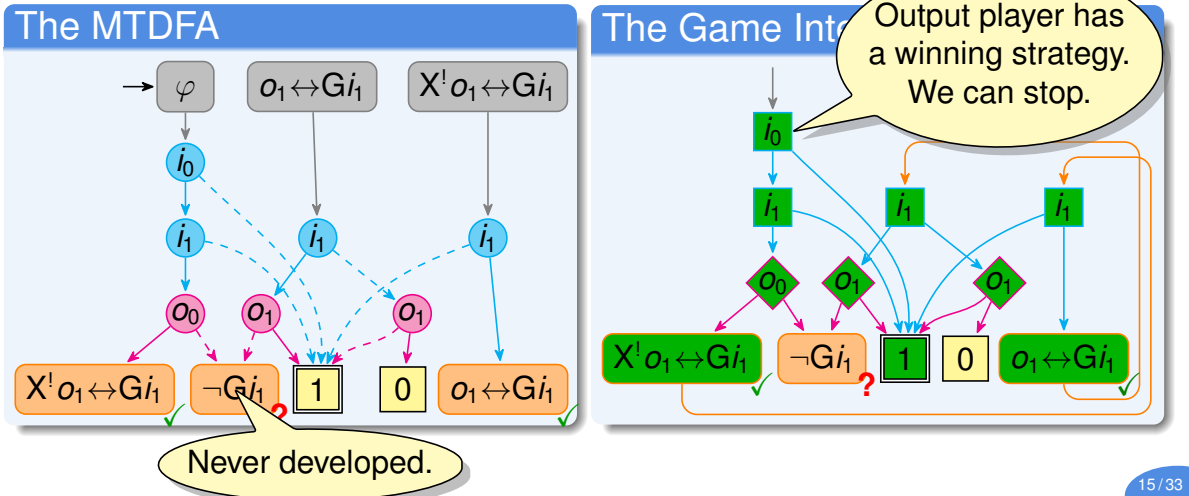


The Game Interpretation



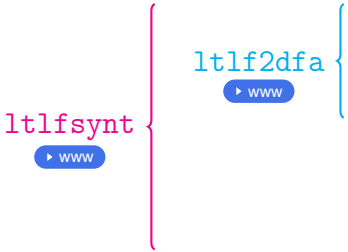
Building the MTDFA & Solving the Game On-The-Fly

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



LTL_f Synthesis with MTDFA

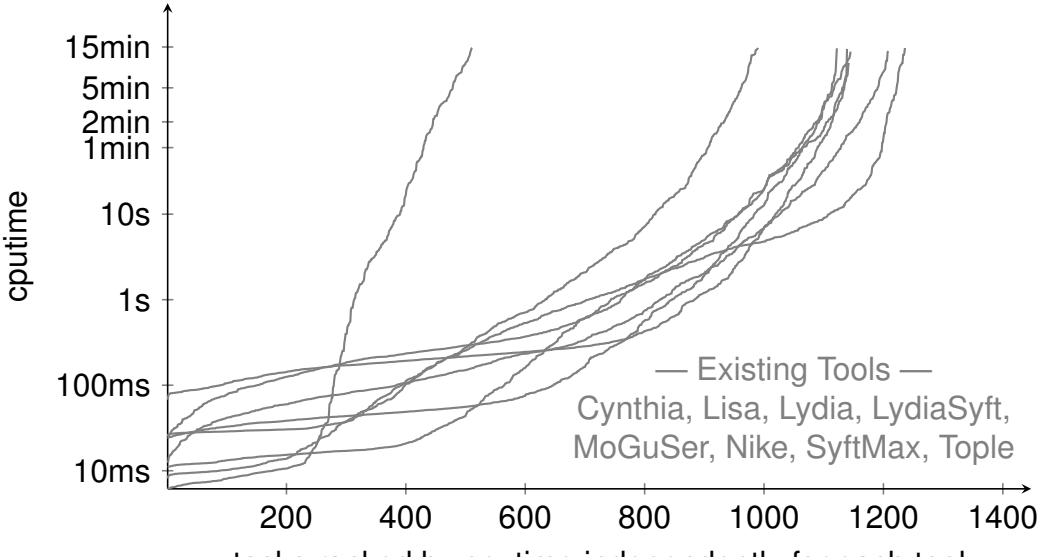
Implemented in two new tools distributed with Spot 2.14 ▸ www



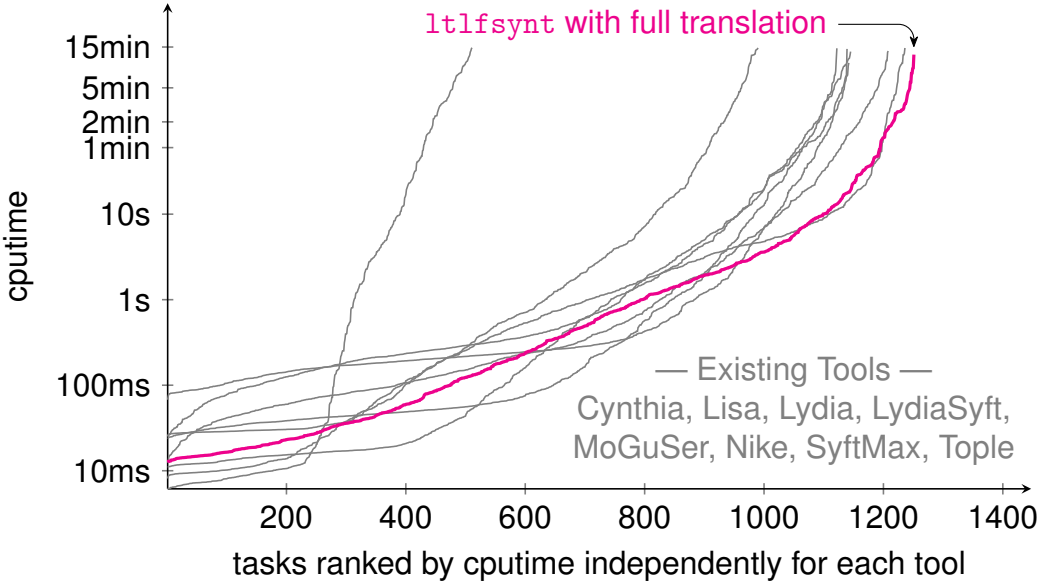
What we Suggest

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. ✓

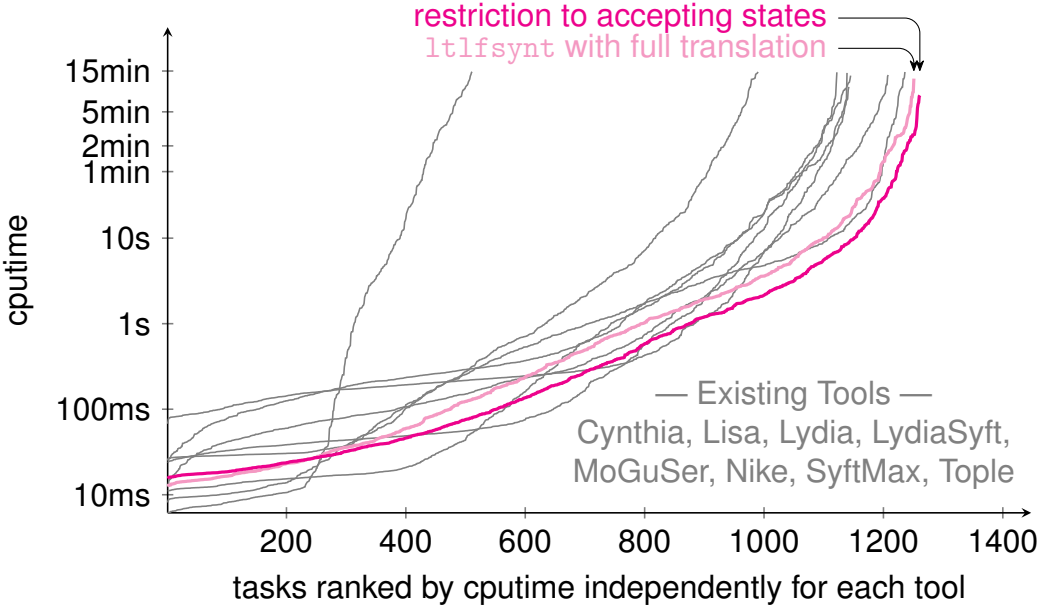
A Benchmark (Specifications from SyntComp)



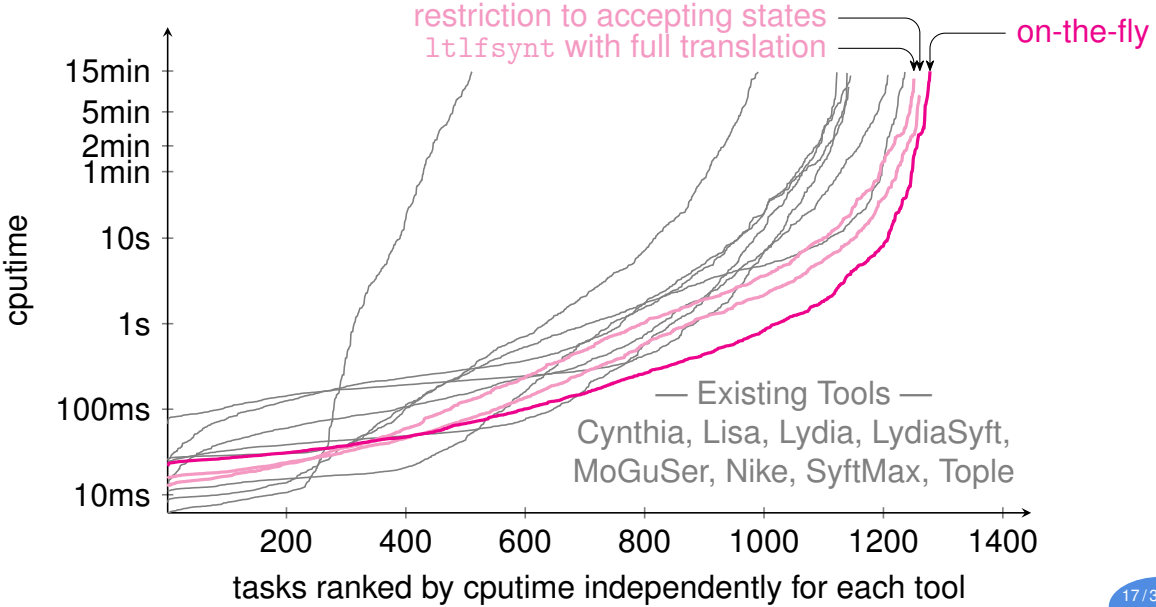
A Benchmark (Specifications from SyntComp)



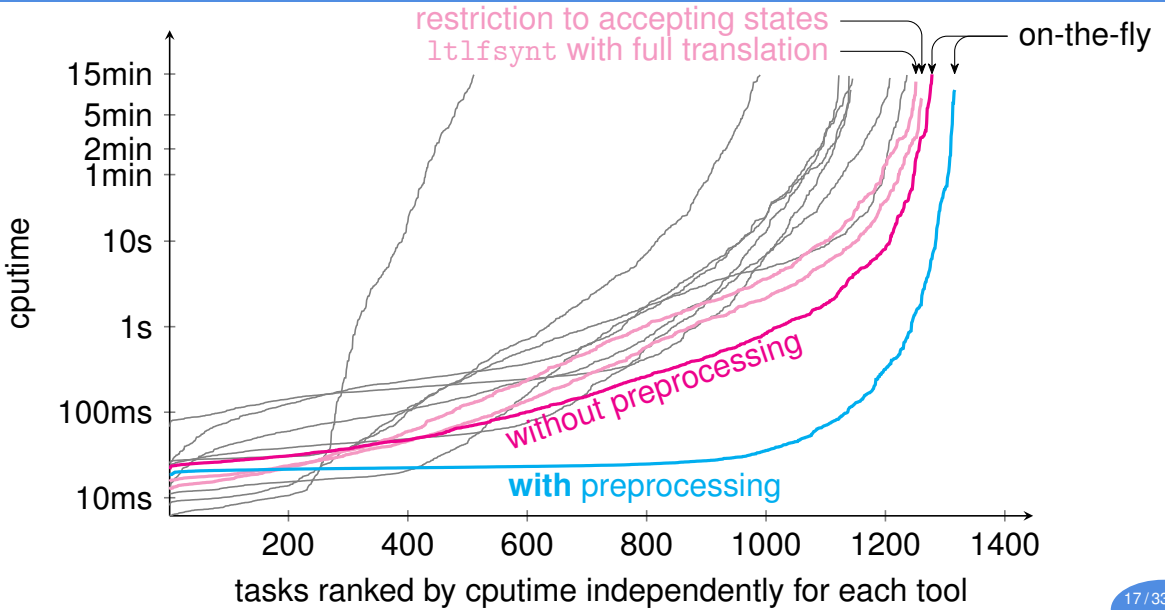
A Benchmark (Specifications from SyntComp)



A Benchmark (Specifications from SyntComp)



A Benchmark (Specifications from SyntComp)



Conclusion (for first part)

Efficient LTL$_f$ tools to build upon

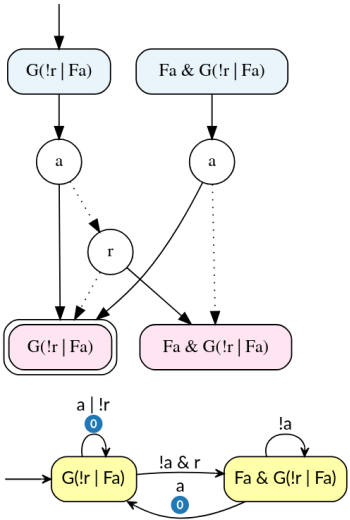
Distributed with Spot 2.14:

- `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).

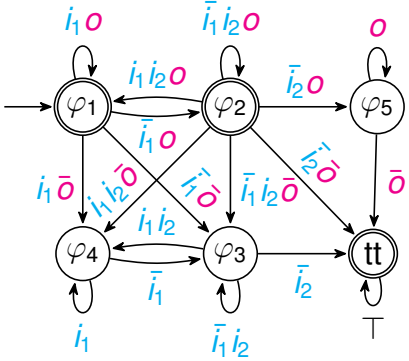
```
In [20]: a3 = spot.ltlf_to_mtdfa("G(!r | Fa)")
         display(a3, a3.as_twa())
```

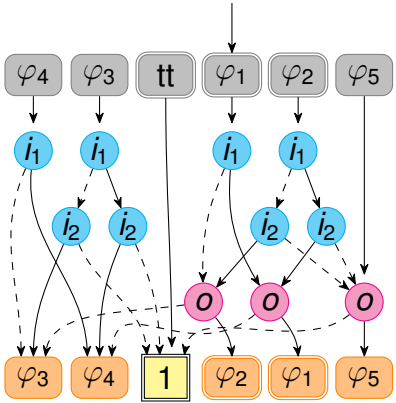


Outline

Duret-Lutz et al. *Fast obligation translation and synthesis*. **CAV'26**. To appear.

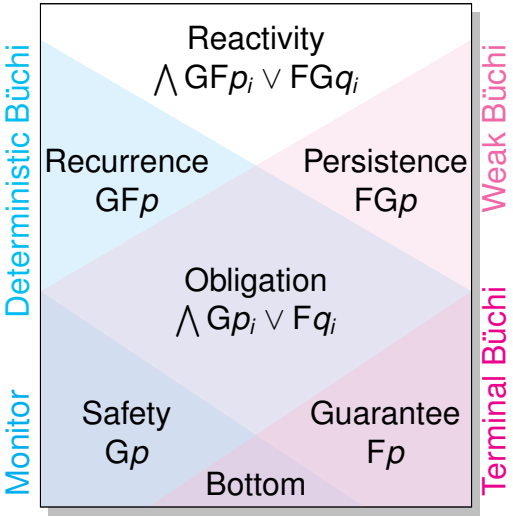
MTBDD-based DBA with State-Based Acceptance



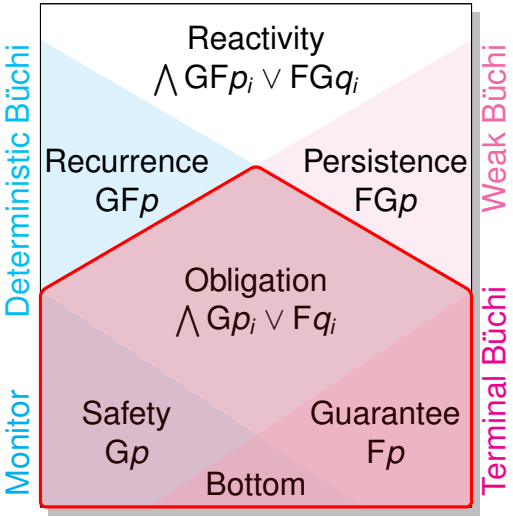


$$\varphi_1 = \mathsf{G}(i_1 \vee \mathsf{X} i_2) \leftrightarrow \mathsf{G} o$$
$$\varphi_2 = (i_2 \wedge \mathsf{G}(i_1 \vee \mathsf{X} i_2)) \leftrightarrow \mathsf{G} o$$
$$\varphi_3 = \neg(i_2 \wedge \mathsf{G}(i_1 \vee \mathsf{X} i_2))$$
$$\varphi_4 = \neg \mathsf{G}(i_1 \vee \mathsf{X} i_2)$$
$$\varphi_5 = \neg \mathsf{G} o$$

The Manna-Pnueli Hierarchy



The Manna-Pnueli Hierarchy



Obligations = Weak Deterministic Büchi Automata

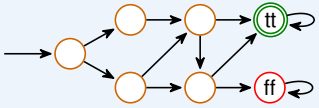
Bottom

$$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$$

Obligations = Weak Deterministic Büchi Automata

Bottom

$$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$$

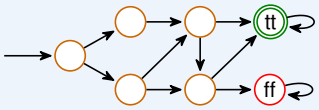


DBA is a DAG, until (tt) or (ff)

Obligations = Weak Deterministic Büchi Automata

Bottom

$$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$$



DBA is a DAG, until (tt) or (ff)

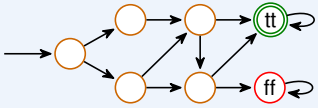
Safety

$$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \mathsf{X}\varphi_S \mid \mathsf{G}\varphi_S \mid \varphi_S \,\mathsf{R}\, \varphi_S$$

Obligations = Weak Deterministic Büchi Automata

Bottom

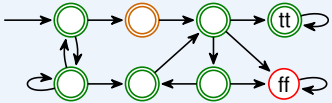
$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$



DBA is a DAG, until (tt) or (ff)

Safety

$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \mathsf{X}\varphi_S \mid \mathsf{G}\varphi_S \mid \varphi_S \,\mathsf{R}\, \varphi_S$

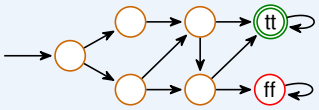


all states but (ff) are accepting

Obligations = Weak Deterministic Büchi Automata

Bottom

$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$



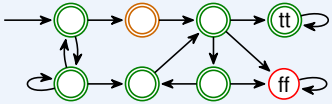
DBA is a DAG, until (tt) or (ff)

Guarantee

$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \mathsf{X}\varphi_G \mid \mathsf{F}\varphi_G \mid \varphi_G \,\mathsf{U}\, \varphi_G$

Safety

$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \mathsf{X}\varphi_S \mid \mathsf{G}\varphi_S \mid \varphi_S \,\mathsf{R}\, \varphi_S$

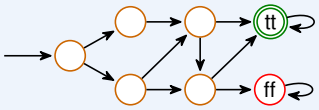


all states but (ff) are accepting

Obligations = Weak Deterministic Büchi Automata

Bottom

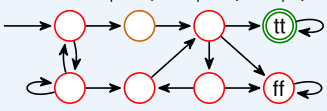
$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$



DBA is a DAG, until tt or ff

Guarantee

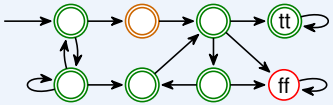
$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \mathsf{X}\varphi_G \mid \mathsf{F}\varphi_G \mid \varphi_G \,\mathsf{U}\, \varphi_G$



all states but tt are rejecting

Safety

$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \mathsf{X}\varphi_S \mid \mathsf{G}\varphi_S \mid \varphi_S \,\mathsf{R}\, \varphi_S$

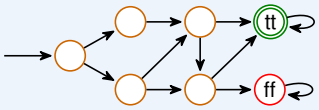


all states but ff are accepting

Obligations = Weak Deterministic Büchi Automata

Bottom

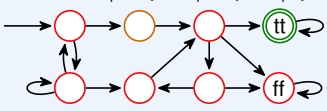
$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$



DBA is a DAG, until (tt) or (ff)

Guarantee

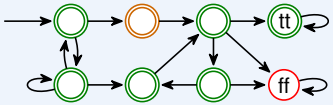
$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \neg\varphi_S \mid \mathsf{X}\varphi_G \mid \mathsf{F}\varphi_G \mid \varphi_G \,\mathsf{U}\, \varphi_G$



all states but (tt) are rejecting

Safety

$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \neg\varphi_G \mid \mathsf{X}\varphi_S \mid \mathsf{G}\varphi_S \mid \varphi_S \,\mathsf{R}\, \varphi_S$

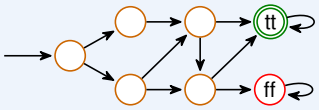


all states but (ff) are accepting

Obligations = Weak Deterministic Büchi Automata

Bottom

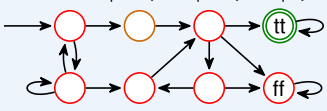
$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$



DBA is a DAG, until (tt) or (ff)

Guarantee

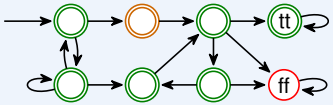
$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \neg\varphi_S \mid \mathsf{X}\varphi_G \mid \mathsf{F}\varphi_G \mid \varphi_G \,\mathsf{U}\, \varphi_G$



all states but (tt) are rejecting

Safety

$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \neg\varphi_G \mid \mathsf{X}\varphi_S \mid \mathsf{G}\varphi_S \mid \varphi_S \,\mathsf{R}\, \varphi_S$



all states but (ff) are accepting

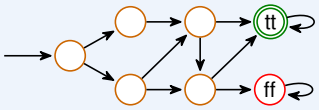
Obligation

$\varphi_O ::= \varphi_G \mid \varphi_S \mid \neg\varphi_O \mid \varphi_O \wedge \varphi_O \mid \varphi_O \vee \varphi_O \mid \mathsf{X}\varphi_O$

Obligations = Weak Deterministic Büchi Automata

Bottom

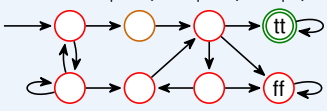
$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$



DBA is a DAG, until (tt) or (ff)

Guarantee

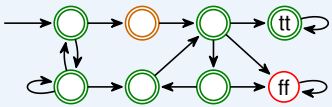
$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \neg\varphi_S \mid \mathsf{X}\varphi_G \mid \mathsf{F}\varphi_G \mid \varphi_G \,\mathsf{U}\, \varphi_G$



all states but (tt) are rejecting

Safety

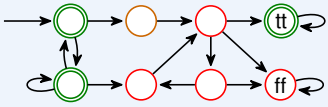
$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \neg\varphi_G \mid \mathsf{X}\varphi_S \mid \mathsf{G}\varphi_S \mid \varphi_S \,\mathsf{R}\, \varphi_S$



all states but (ff) are accepting

Obligation

$\varphi_O ::= \varphi_G \mid \varphi_S \mid \neg\varphi_O \mid \varphi_O \wedge \varphi_O \mid \varphi_O \vee \varphi_O \mid \mathsf{X}\varphi_O$

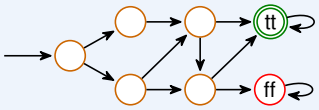


each SCC contains only accepting or rejecting states

Obligations = Weak Deterministic Büchi Automata

Bottom

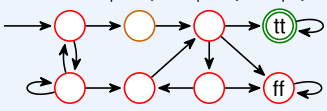
$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$



DBA is a DAG, until (tt) or (ff)

Guarantee

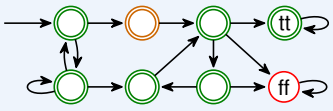
$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \neg\varphi_S \mid \mathsf{X}\varphi_G \mid \mathsf{F}\varphi_G \mid \varphi_G \,\mathsf{U}\, \varphi_G$



all states but (tt) are rejecting

Safety

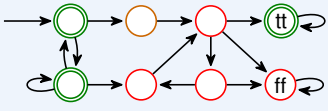
$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \neg\varphi_G \mid \mathsf{X}\varphi_S \mid \mathsf{G}\varphi_S \mid \varphi_S \,\mathsf{R}\, \varphi_S$



all states but (ff) are accepting

Obligation

$\varphi_O ::= \varphi_G \mid \varphi_S \mid \neg\varphi_O \mid \varphi_O \wedge \varphi_O \mid \varphi_O \vee \varphi_O \mid \mathsf{X}\varphi_O \mid \varphi_O \,\mathsf{U}\, \varphi_G \mid \varphi_O \,\mathsf{R}\, \varphi_S$



each SCC contains only accepting or rejecting states

Chang, Manna, and Pnueli. *Characterization of temporal property classes.* **ICALP'92.** doi

Deciding acceptance of an obligation-labeled state

$\lambda(\mathsf{ff}) = \bot$ $\quad$ $\lambda(\mathsf{G}\alpha) = \top$ $\quad$ $\lambda(\mathsf{F}\alpha) = \bot$

$\lambda(\mathsf{tt}) = \top$ $\quad$ $\lambda(\alpha\ \mathsf{R}\ \beta) = \top$ $\quad$ $\lambda(\alpha\ \mathsf{U}\ \beta) = \bot$

$\lambda(\neg\alpha) = \neg\lambda(\alpha)$ $\quad$ $\lambda(\alpha \wedge \beta) = \lambda(\alpha) \wedge \lambda(\beta)$ $\quad$ $\lambda(\alpha \vee \beta) = \lambda(\alpha) \vee \lambda(\beta)$

Deciding acceptance of an obligation-labeled state

$$\lambda(\mathsf{ff}) = \bot \qquad \lambda(\mathsf{G}\alpha) = \top \qquad \lambda(\mathsf{F}\alpha) = \bot$$

$$\lambda(\mathsf{tt}) = \top \qquad \lambda(\alpha\ \mathsf{R}\ \beta) = \top \qquad \lambda(\alpha\ \mathsf{U}\ \beta) = \bot$$

$$\lambda(p) = *$$

$$\lambda(\mathsf{X}\alpha) = *$$

$$\lambda(\neg\alpha) = \neg\lambda(\alpha) \qquad \lambda(\alpha \wedge \beta) = \lambda(\alpha) \wedge \lambda(\beta) \qquad \lambda(\alpha \vee \beta) = \lambda(\alpha) \vee \lambda(\beta)$$

will ignore any "$*$"

Deciding acceptance of an obligation-labeled state

$$\lambda(\mathsf{ff}) = \bot \qquad \lambda(\mathsf{G}\alpha) = \top \qquad \lambda(\mathsf{F}\alpha) = \bot$$
$$\lambda(\mathsf{tt}) = \top \qquad \lambda(\alpha\ \mathsf{R}\ \beta) = \top \qquad \lambda(\alpha\ \mathsf{U}\ \beta) = \bot$$
$$\lambda(p) = *$$
$$\lambda(\mathsf{X}\alpha) = *$$
$$\lambda(\neg\alpha) = \neg\lambda(\alpha) \qquad \lambda(\alpha \wedge \beta) = \lambda(\alpha) \wedge \lambda(\beta) \qquad \lambda(\alpha \vee \beta) = \lambda(\alpha) \vee \lambda(\beta)$$

will ignore any "$*$"

Example:

$$\lambda(\varphi_2) = \lambda((\underbrace{\underbrace{i_2}_{*} \wedge \underbrace{\mathsf{G}(i_1 \vee \mathsf{X}i_2)}_{\top}}_{\top} \leftrightarrow \underbrace{\mathsf{G}o}_{\top}) = \top$$

Deciding acceptance of an obligation-labeled state

$$\lambda(\mathsf{ff}) = \bot \qquad \lambda(\mathsf{G}\alpha) = \top \qquad \lambda(\mathsf{F}\alpha) = \bot$$
$$\lambda(\mathsf{tt}) = \top \qquad \lambda(\alpha\ \mathsf{R}\ \beta) = \top \qquad \lambda(\alpha\ \mathsf{U}\ \beta) = \bot$$
$$\lambda(p) = *$$
$$\lambda(\mathsf{X}\alpha) = *$$
$$\lambda(\neg\alpha) = \neg\lambda(\alpha) \qquad \lambda(\alpha \wedge \beta) = \lambda(\alpha) \wedge \lambda(\beta) \qquad \lambda(\alpha \vee \beta) = \lambda(\alpha) \vee \lambda(\beta)$$

will ignore any "$*$"

Example:

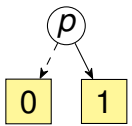
$$\lambda(\varphi_2) = \lambda((\underbrace{i_2}_{*} \wedge \underbrace{\mathsf{G}(i_1 \vee \mathsf{X}i_2)}_{\top}) \leftrightarrow \underbrace{\mathsf{G}o}_{\top}) = \top$$

follows from the above rules

MTBDD translation is straightforward

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

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

$\mathrm{tr}(p) =$ 

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

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

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

LTL operator

MTBDD operator

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

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

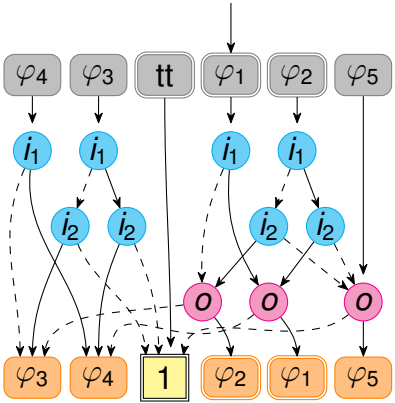
$\mathrm{tr}(\alpha\ \mathsf{R}\ \beta) = \mathrm{tr}(\beta) \wedge (\mathrm{tr}(\alpha) \vee \alpha\ \mathsf{R}\ \beta)$

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

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

Acceptance of a state labeled by α is $\lambda(\alpha)$.

MTBDD-based DBA with State-Based Acceptance



$\varphi_1 = \mathsf{G}(i_1 \vee \mathsf{X} i_2) \leftrightarrow \mathsf{G} o$

$\varphi_2 = (i_2 \wedge \mathsf{G}(i_1 \vee \mathsf{X} i_2)) \leftrightarrow \mathsf{G} o$

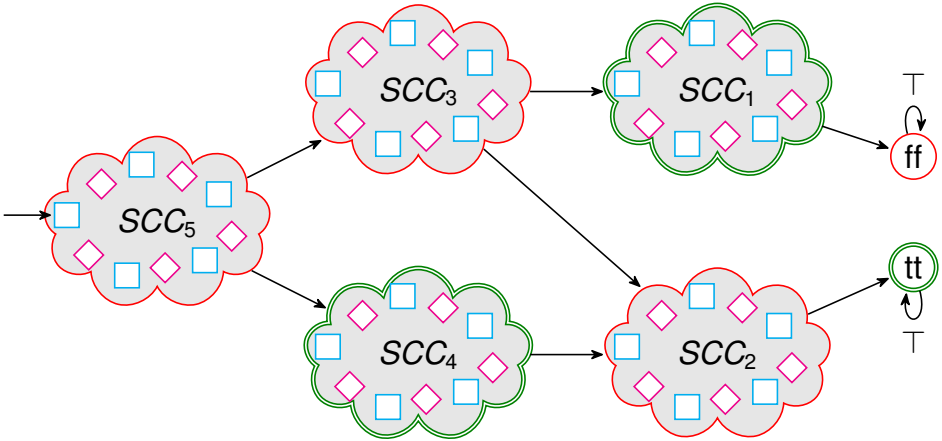
$\varphi_3 = \neg(i_2 \wedge \mathsf{G}(i_1 \vee \mathsf{X} i_2))$

$\varphi_4 = \neg\mathsf{G}(i_1 \vee \mathsf{X} i_2)$

$\varphi_5 = \neg\mathsf{G} o$

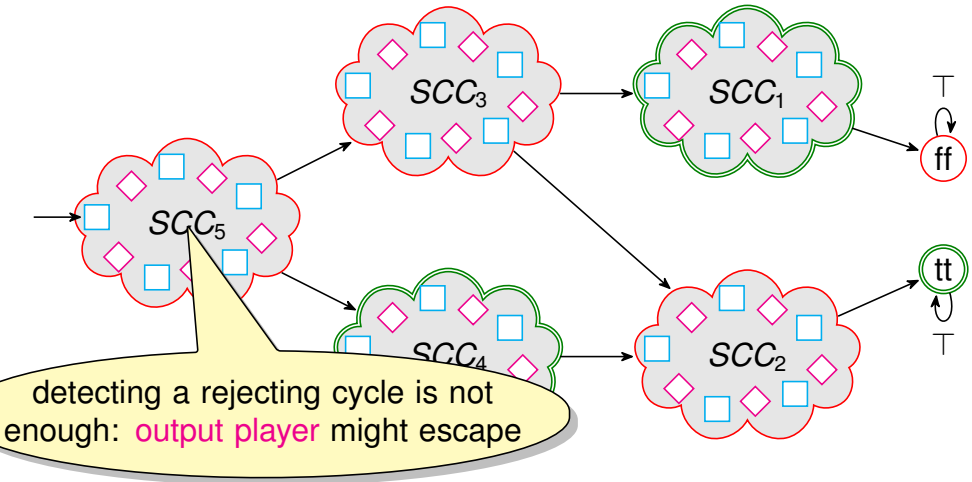
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



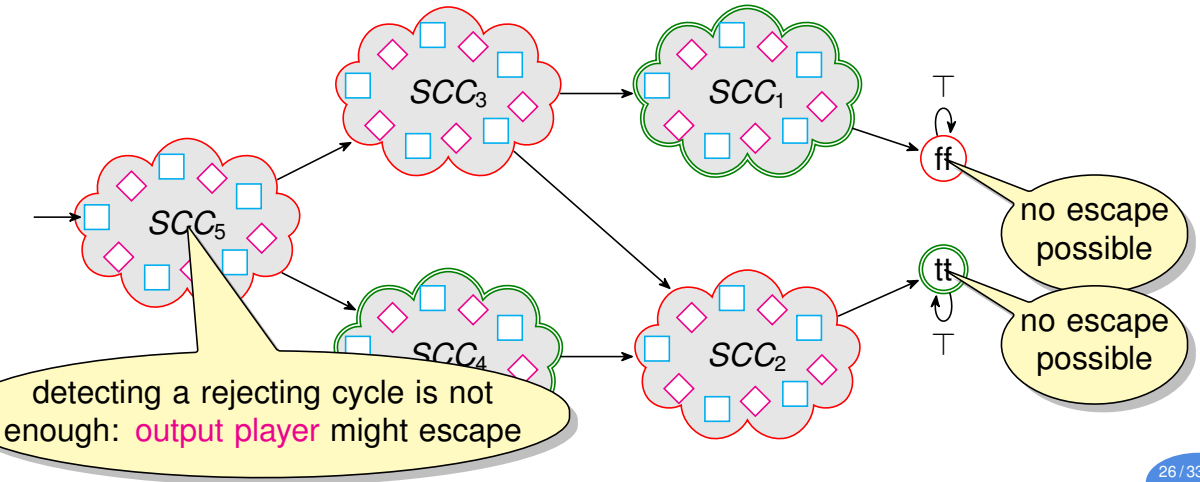
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



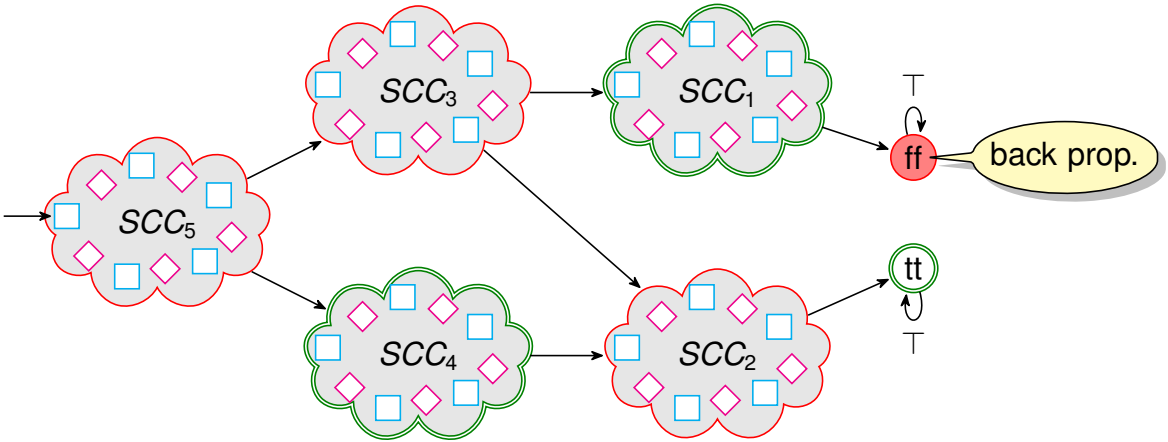
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



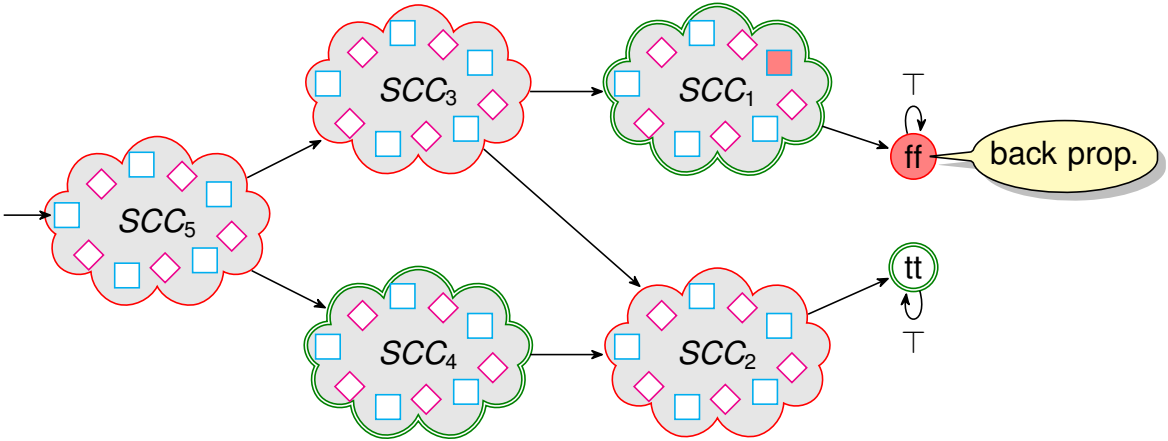
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



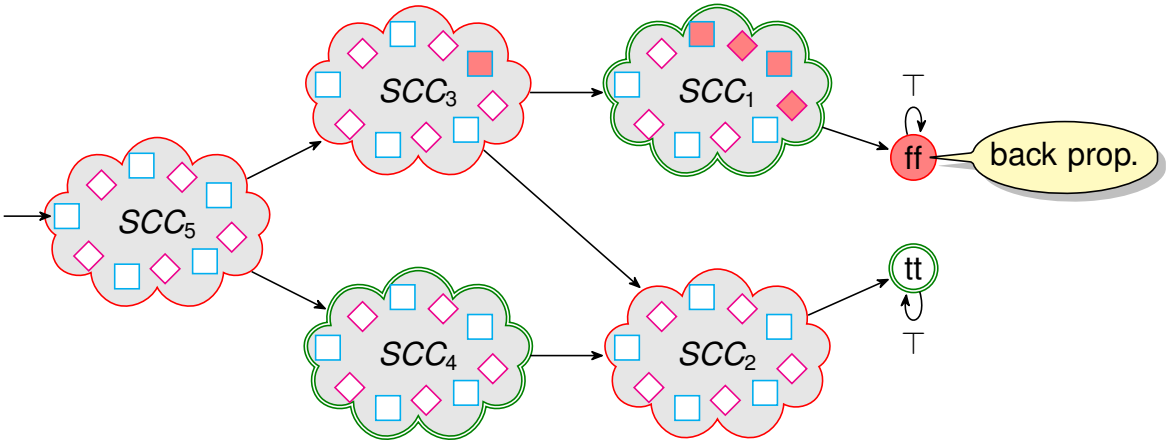
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



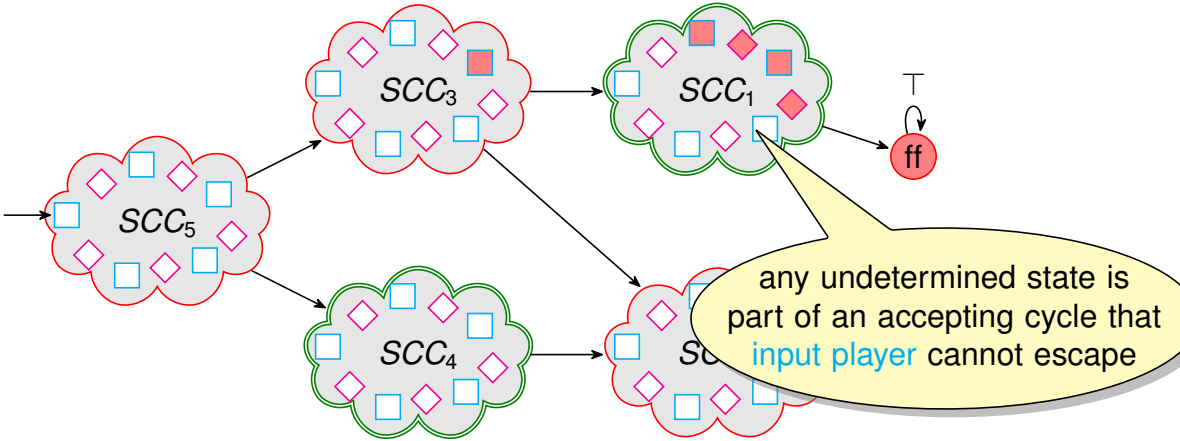
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



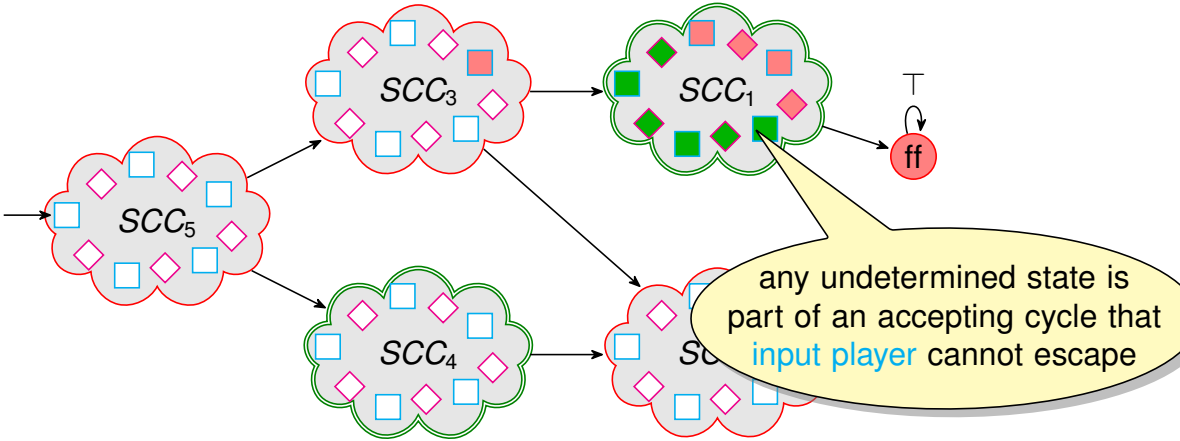
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



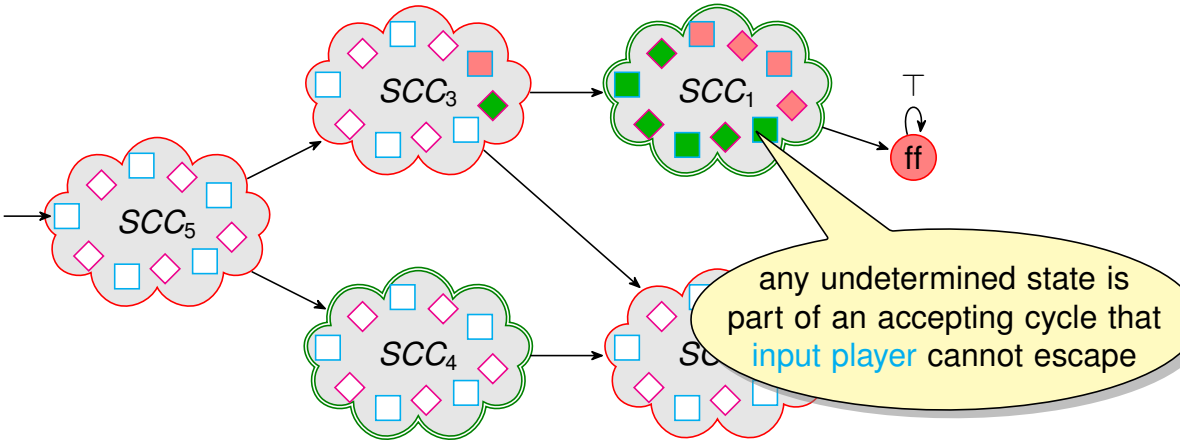
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



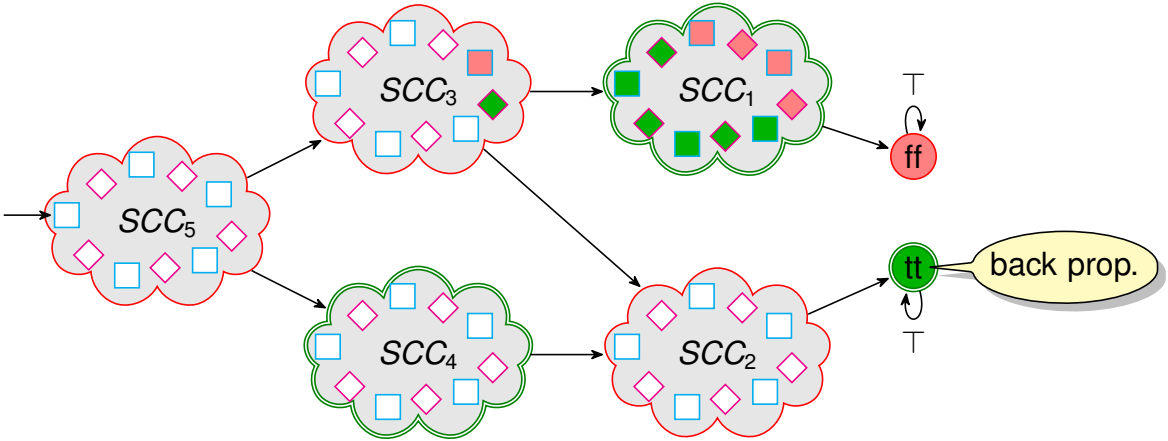
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



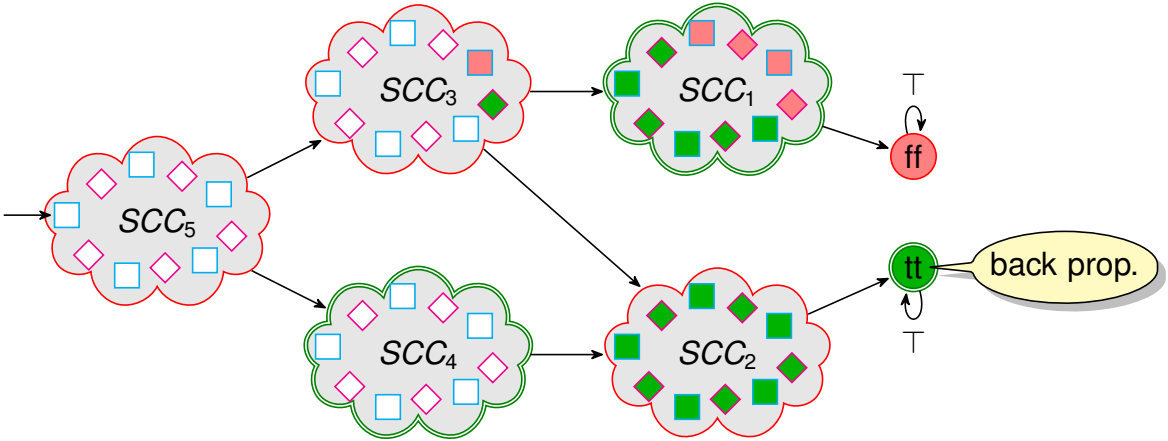
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



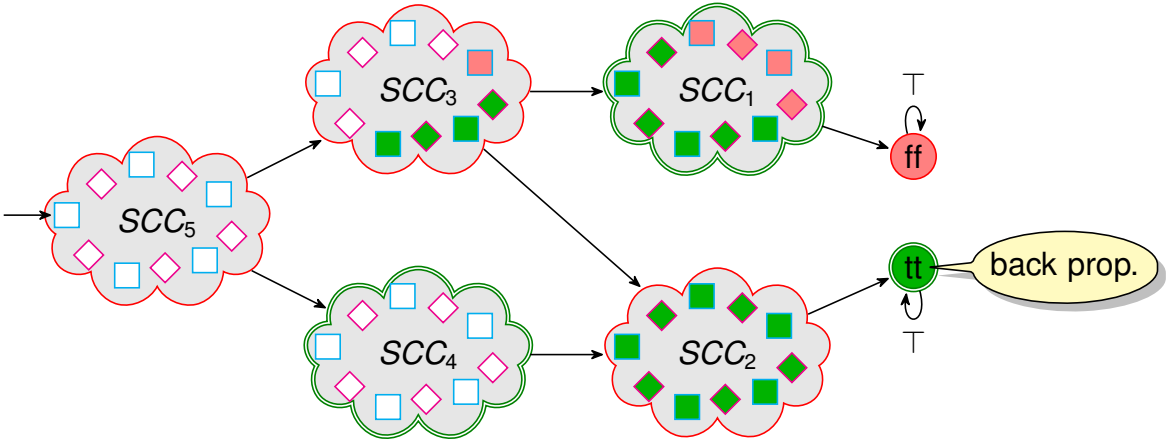
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



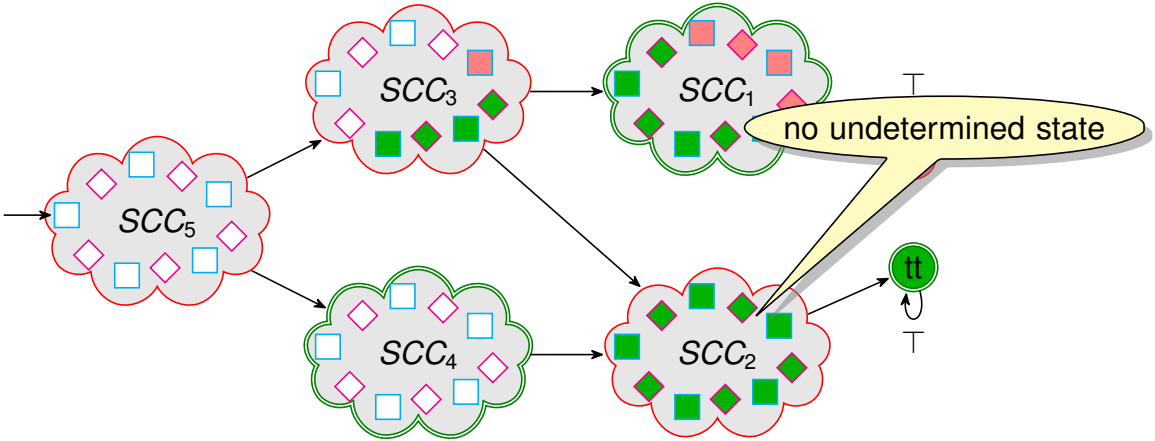
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



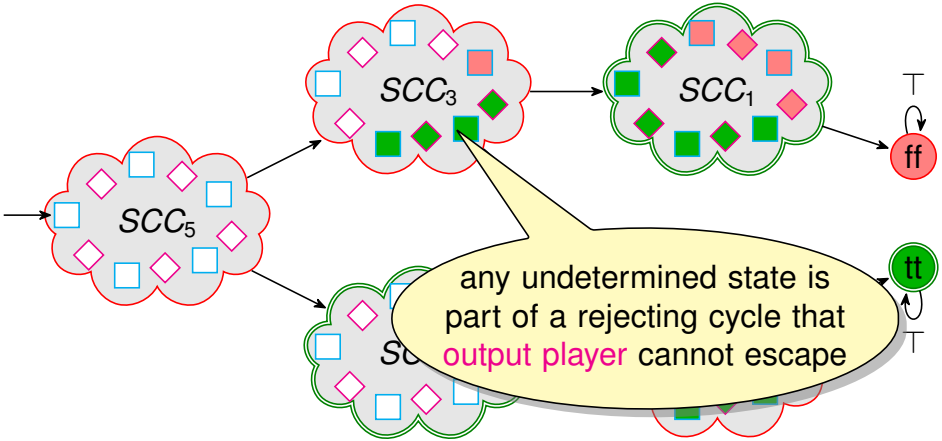
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



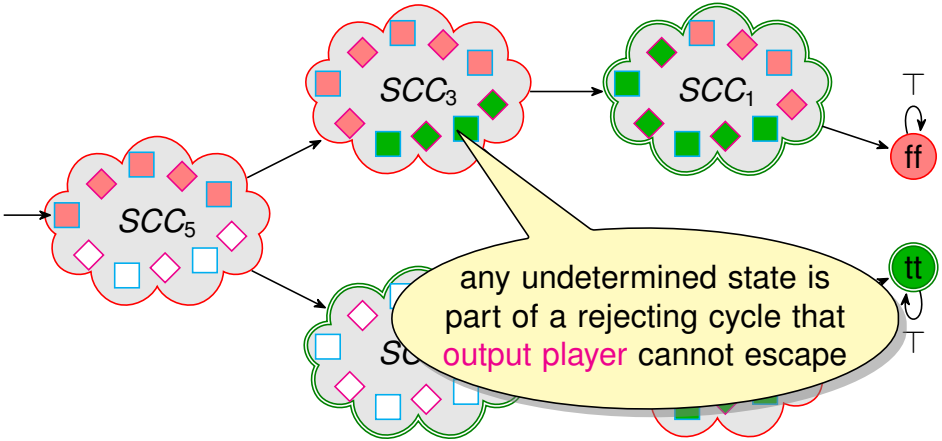
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



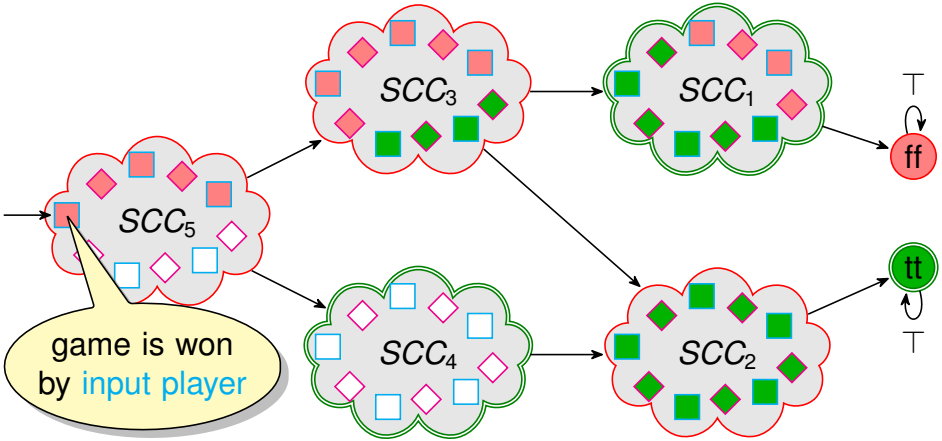
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



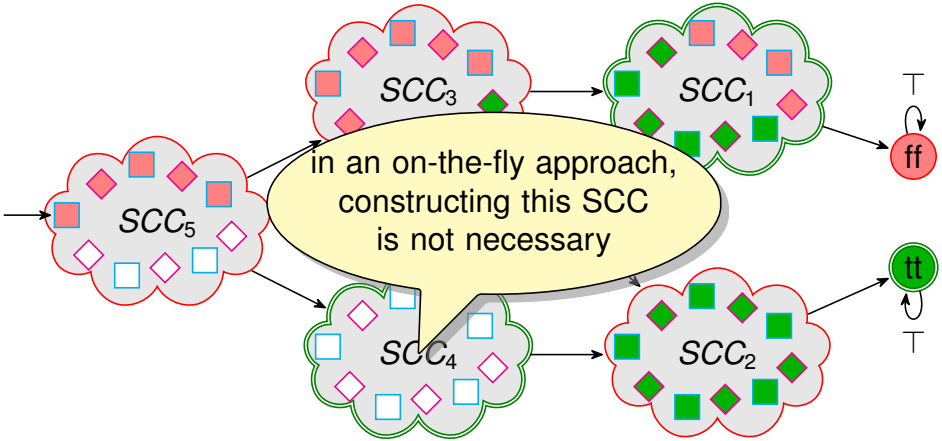
Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.

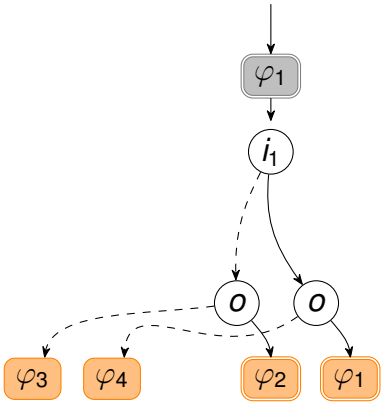


Solving Weak Games Bottom-Up

The input player wants the play to get stuck in a rejecting SCC.
The output player wants the play to get stuck in an accepting SCC.



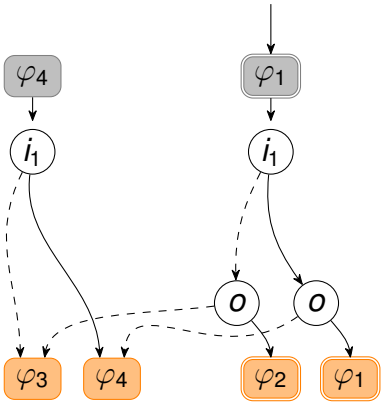
On-the-Fly MTBDD-based Obligation Synthesis



$\{\varphi_1\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis

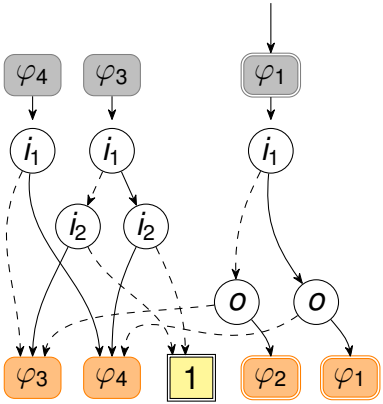


$\{\varphi_1\}$

$\{\varphi_4\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis

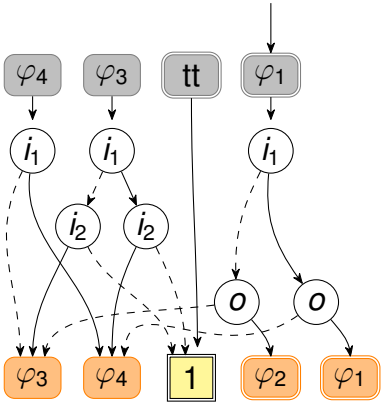


$\{\varphi_1\}$

$\{\varphi_4, \varphi_3\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis



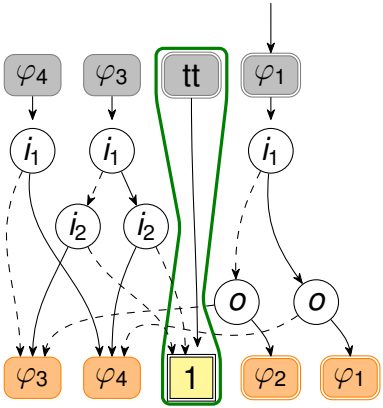
$\{\varphi_1\}$

$\{\varphi_4, \varphi_3\}$

$\{\text{tt}\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis



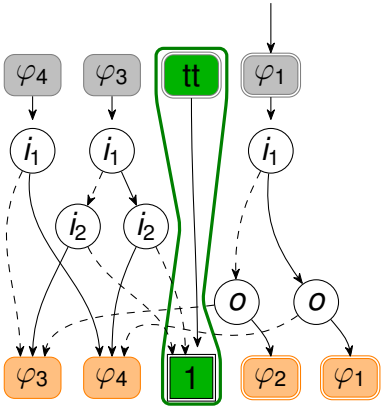
$\{\varphi_1\}$

$\{\varphi_4, \varphi_3\}$

$\{\mathrm{tt}\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis

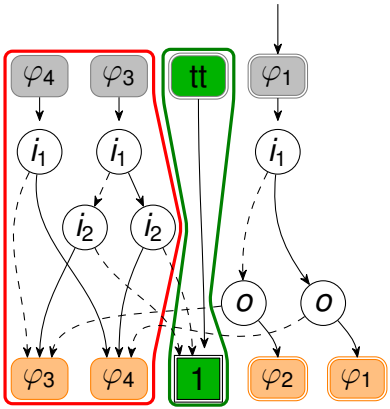


$\{\varphi_1\}$

$\{\varphi_4, \varphi_3\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis

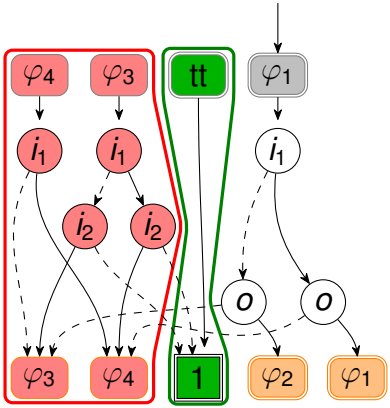


$\{\varphi_1\}$

$\{\varphi_4, \varphi_3\}$

DFS stack organized as partial SCCs

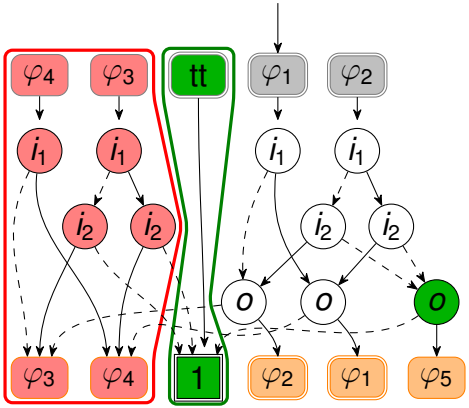
On-the-Fly MTBDD-based Obligation Synthesis



$\{\varphi_1\}$

DFS stack organized as partial SCCs

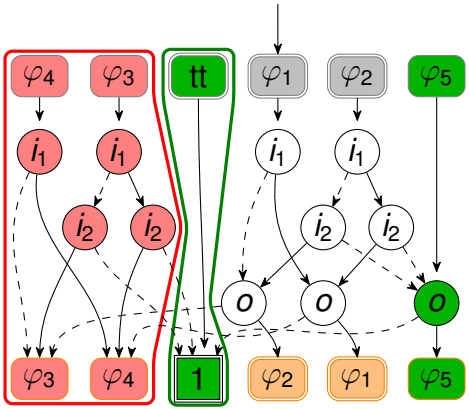
On-the-Fly MTBDD-based Obligation Synthesis



$\{\varphi_1, \varphi_2\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis

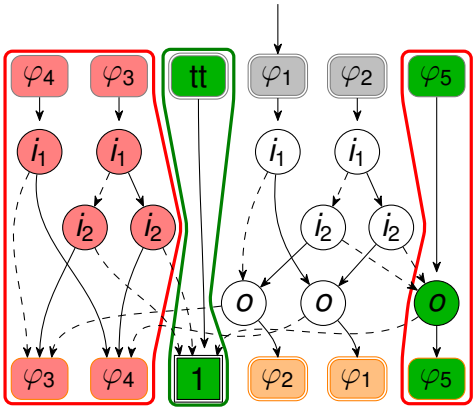


$\{\varphi_1, \varphi_2\}$

$\{\varphi_5\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis

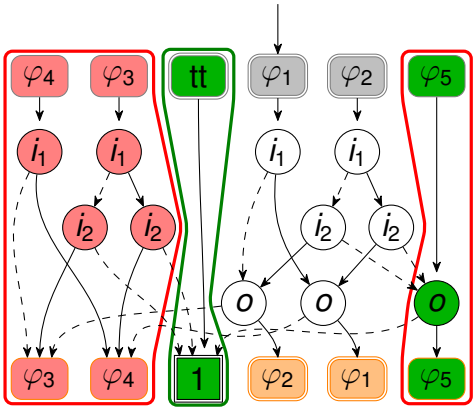


$\{\varphi_1, \varphi_2\}$

$\{\varphi_5\}$

DFS stack organized as partial SCCs

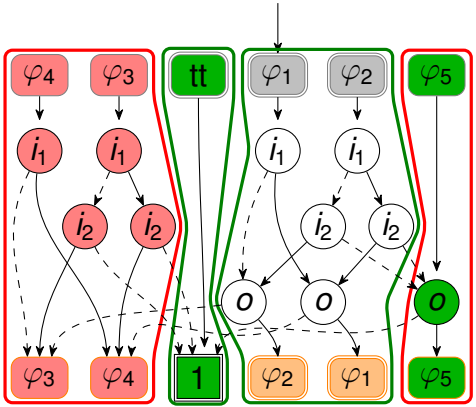
On-the-Fly MTBDD-based Obligation Synthesis



$\{\varphi_1, \varphi_2\}$

DFS stack organized as partial SCCs

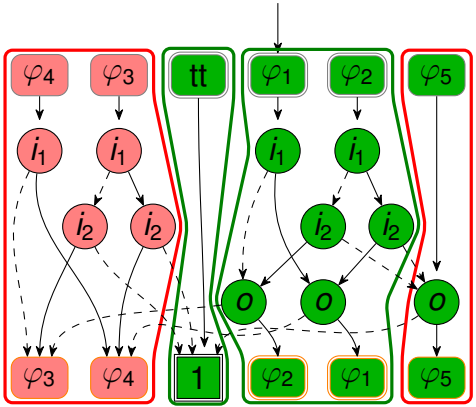
On-the-Fly MTBDD-based Obligation Synthesis



$\{\varphi_1, \varphi_2\}$

DFS stack organized as partial SCCs

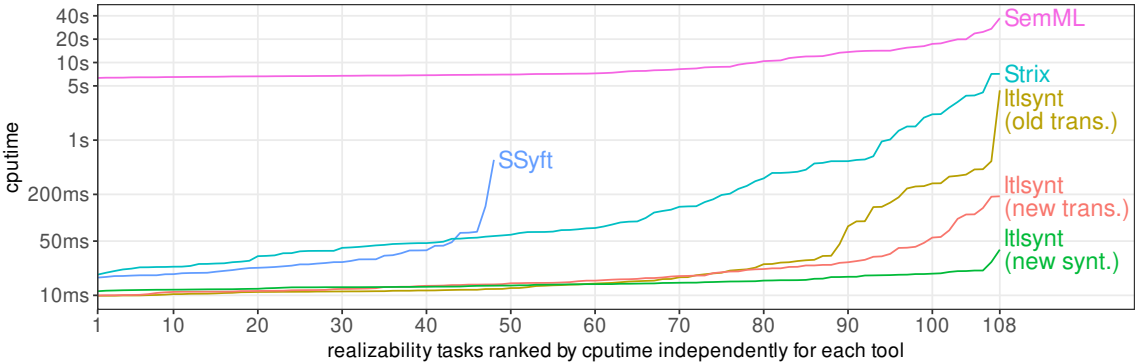
On-the-Fly MTBDD-based Obligation Synthesis



DFS stack organized as partial SCCs

Improving `ltlsynt` on Obligations

The SyntComp repository contains 108 obligations.



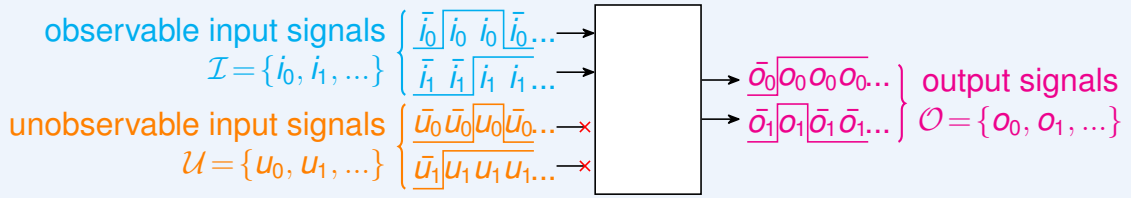
The default in Spot 2.15.

Outline

Alon et al. *On-the-fly LTLf synthesis under partial observability*. **KR'26**. To appear.

Reactive Synthesis with Partial Observation

The reactive controller is **blind** to some of the inputs



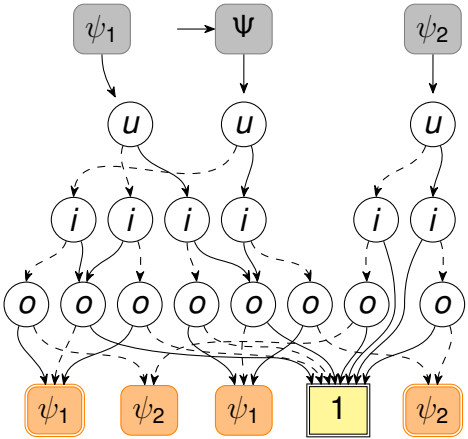
The realizability problem

Given a specification φ over $\mathcal{I} \cup \mathcal{U} \cup \mathcal{O}$, we want to know if $\boxed{\forall \mathcal{U}, \varphi}$ is realizable.

An automaton for $\boxed{\forall \mathcal{U}, \varphi}$ can be computed on-the-fly for LTLf or for obligations!

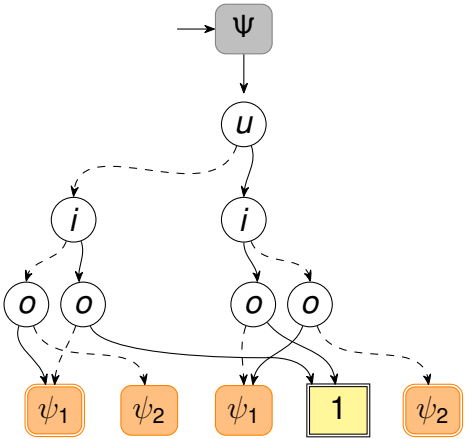
An LTLf Example

$$\Psi = \underbrace{((\mathsf{GF}u) \rightarrow \mathsf{F}(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((\mathsf{GF}\neg u) \rightarrow \mathsf{F}(i \vee o))}_{\psi_2}$$



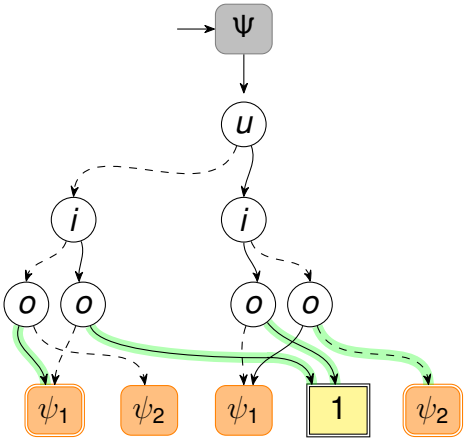
An LTLf Example

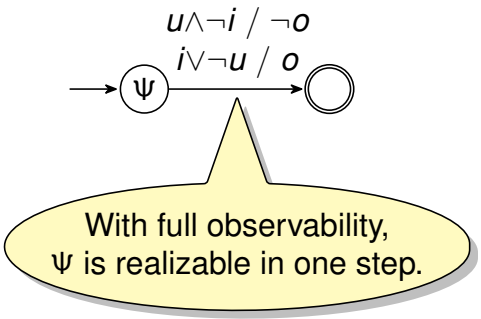
$$\Psi = \underbrace{((\mathsf{GF}u) \rightarrow \mathsf{F}(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((\mathsf{GF}\neg u) \rightarrow \mathsf{F}(i \vee o))}_{\psi_2}$$



An LTLf Example

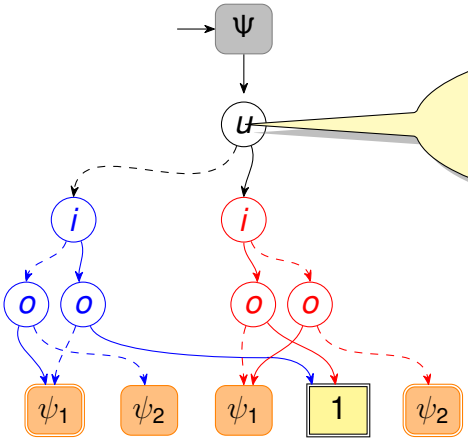
$$\Psi = \underbrace{((\mathsf{GF}u) \rightarrow \mathsf{F}(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((\mathsf{GF}\neg u) \rightarrow \mathsf{F}(i \vee o))}_{\psi_2}$$





An LTLf Example

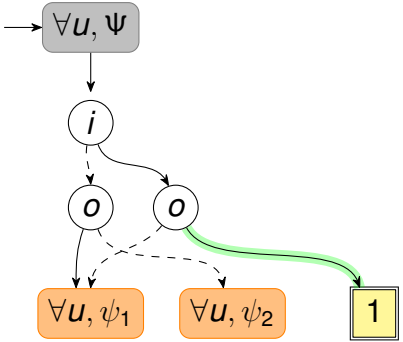
$$\Psi = \underbrace{((\mathsf{GF}u) \rightarrow \mathsf{F}(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((\mathsf{GF}\neg u) \rightarrow \mathsf{F}(i \vee o))}_{\psi_2}$$



To compute $\forall u, \Psi$
we use quantification on MTBDD:
$\forall u, B = B[u \leftarrow \bot] \wedge B[u \leftarrow \top]$

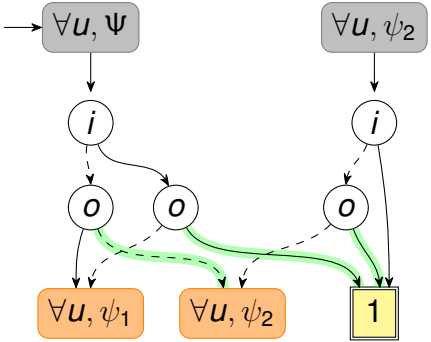
An LTLf Example

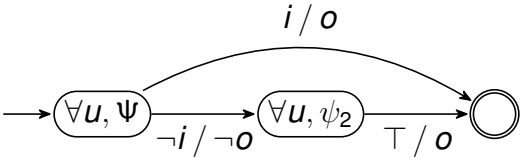
$$\Psi = \underbrace{((\mathsf{GF}u) \rightarrow \mathsf{F}(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((\mathsf{GF}\neg u) \rightarrow \mathsf{F}(i \vee o))}_{\psi_2}$$



An LTLf Example

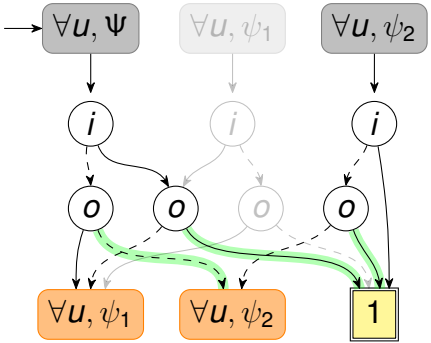
$$\Psi = \underbrace{((\mathsf{GF}u) \rightarrow \mathsf{F}(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((\mathsf{GF}\neg u) \rightarrow \mathsf{F}(i \vee o))}_{\psi_2}$$

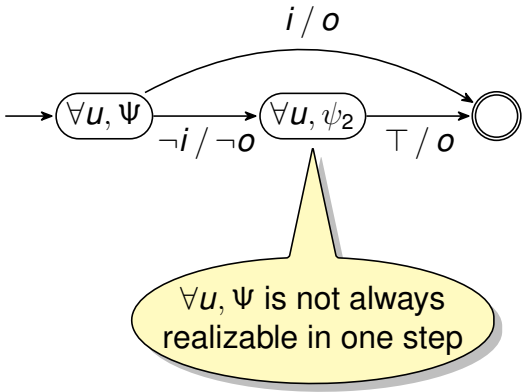




An LTLf Example

$$\Psi = \underbrace{((\mathsf{GF}u) \rightarrow \mathsf{F}(i \leftrightarrow o))}_{\psi_1} \wedge \underbrace{((\mathsf{GF}\neg u) \rightarrow \mathsf{F}(i \vee o))}_{\psi_2}$$





Extensions

- Computing $\forall\mathcal{U}, \varphi$ during translation using MTBDD works for:
  - LTLf$\rightarrow$DFA
  - Obligation$\rightarrow$WDBA
- For these two translations $\exists\mathcal{V}, \varphi$, can be supported similarly.

For full LTL, synthesis under partial observation can be done as follows:

1. compute a NBA for $\exists\mathcal{U}, \neg\varphi$
2. complement NBA to DPA (i.e., a DPA for $\forall\mathcal{U}, \varphi$)
3. solve parity game

All this is implemented in Spot 2.15.

Conclusion

We have discussed 3 MTBDD-based techniques for synthesis problems:

1. LTLf Synthesis
2. Obligation Synthesis (LTL)
3. Synthesis Under Partial Observability

Future work: generalize to full LTL.


Duret-Lutz et al. *Engineering an LTLf synthesis tool*. **CIAA'25**. doi


Duret-Lutz et al. *Fast obligation translation and synthesis*. **CAV'26**. To appear.


Alon et al. *On-the-fly LTLf synthesis under partial observability*. **KR'26**. To appear.

Warp Zone

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

Use cheap rewritings to reduce number of MTBDD operations

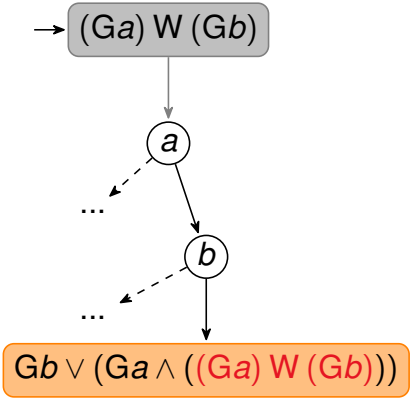
$$\mathsf{X}\alpha \wedge \mathsf{X}\beta \rightsquigarrow \mathsf{X}(\alpha \wedge \beta), \qquad (\alpha \to \beta) \wedge (\alpha \to \gamma) \rightsquigarrow \alpha \to (\beta \wedge \gamma), \qquad \ldots$$

Split specification into output disjoint specifications when possible

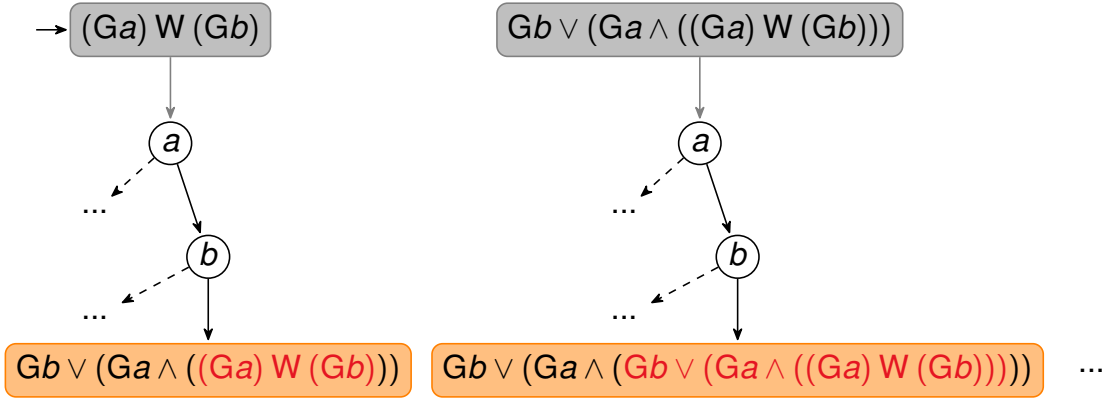
$\Psi_1 \wedge \Psi_2$ can be solved as two independent problems if Ψ_1 and Ψ_2 are output-disjoint, and admit controllers that agree on accepting lengths.

▸ issue 610

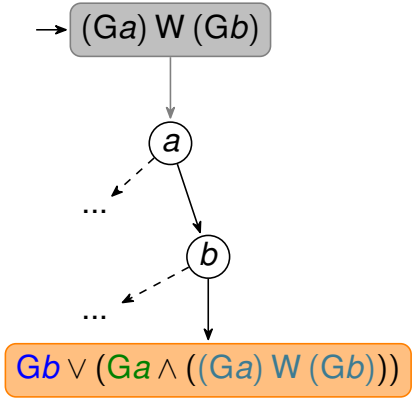
Propositional Equivalence

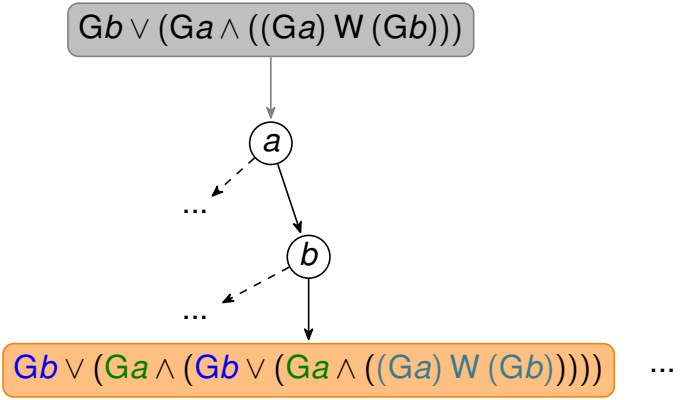


Propositional Equivalence



Propositional Equivalence

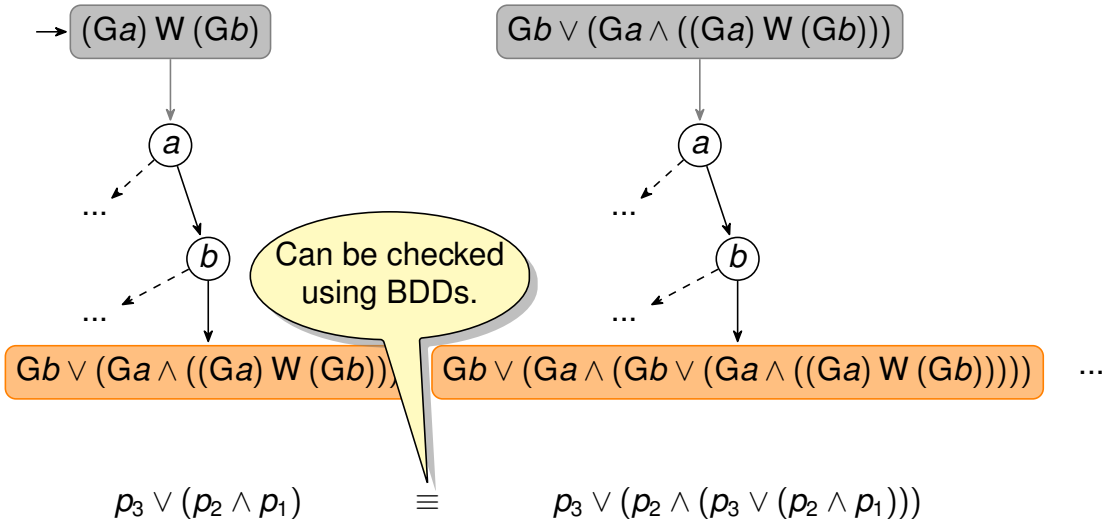




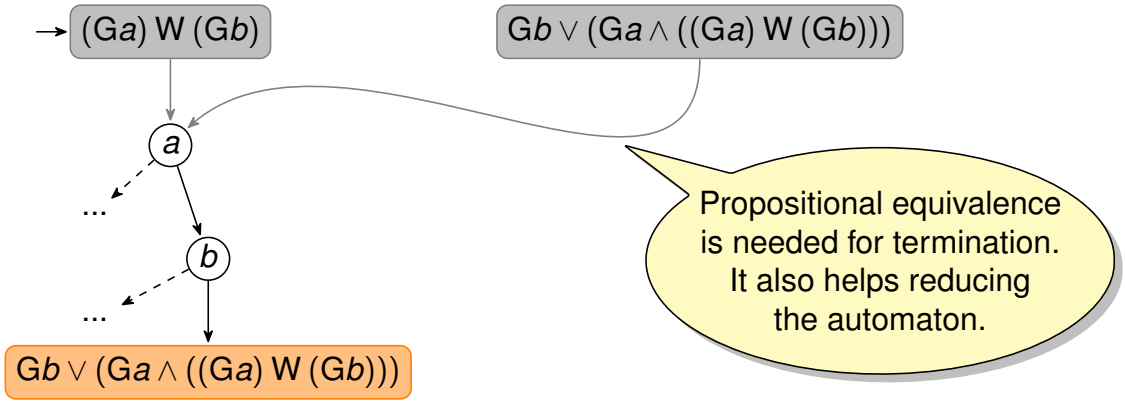
$p_3 \vee (p_2 \wedge p_1)$

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

Propositional Equivalence



Propositional Equivalence



Obligations Frequency

source	set size	property classes					syntactic classes				
		O	$O \setminus S \setminus G$	$S \setminus B$	$G \setminus B$	B	O	$O \setminus S \setminus G$	$S \setminus B$	$G \setminus B$	B
SyntComp	959	$\geq$262	$\geq$172	$\geq$73	$\geq$7	$\geq$7	110	59	48	1	2
Dwyer et al.	55	40	2	37	1		25	13	11	1	
Somenzi&Bloem	27	16	2	6	6	2	13	4	4	5	
Etessami&Holzmann	12	5			4	1	5			5	
BEEM	20	10	1	6	1	2	7	2	4	1	
Liberouter	55	30		27	1	2	26	1	23	2	

Jacobs et al. *The reactive synthesis competition (SYNTCOMP): 2018–2021*. arXiV, 2022. doi

Dwyer, Avrunin, and Corbett. *Property specification patterns for finite-state verification*. **FMSP'98**. doi

Somenzi and Bloem. *Efficient Büchi automata for LTL formulae*. **CAV'00**. doi

Etessami and Holzmann. *Optimizing Büchi automata*. **Concur'00**. doi

Pelánek. *BEEM: benchmarks for explicit model checkers*. **Spin'07**. doi

Holeček et al. *Verification results in Liberouter project*. Technical Report 03, CESNET, 2004