CAV'26

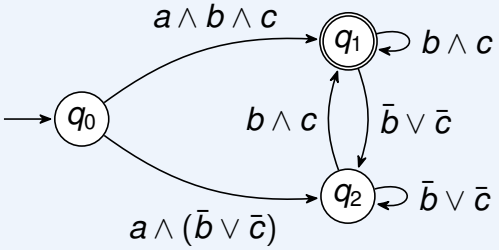
Fast Obligation Translation and Synthesis

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

MTBDDs Against the Alphabet-Explosion Problem

Consider a DBA for $a \wedge \mathrm{GF}(b \wedge c)$:

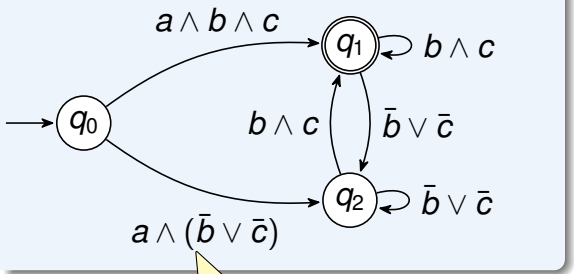
An Explicit Automaton



MTBDDs Against the Alphabet-Explosion Problem

Consider a DBA for $a \wedge \mathrm{GF}(b \wedge c)$:

An Explicit Automaton

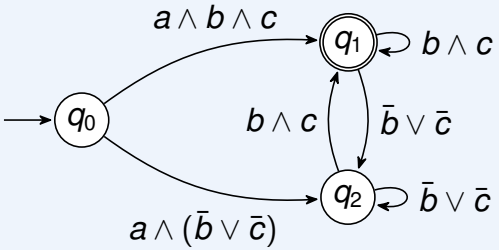


Labeling with Boolean formulas to avoid using the alphabet $2^{\{a,b,c\}}$

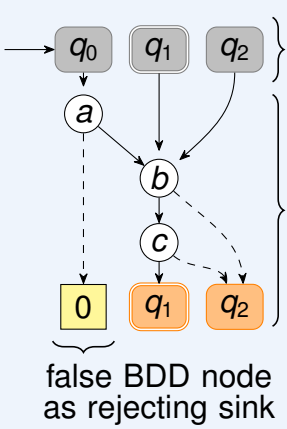
MTBDDs Against the Alphabet-Explosion Problem

Consider a DBA for $a \wedge \mathrm{GF}(b \wedge c)$:

An Explicit Automaton



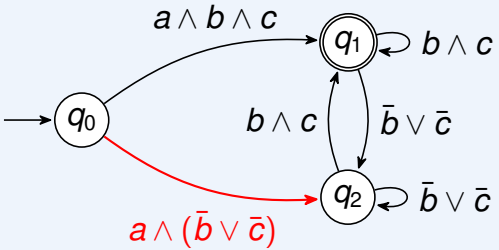
A Semi-Symbolic Encoding



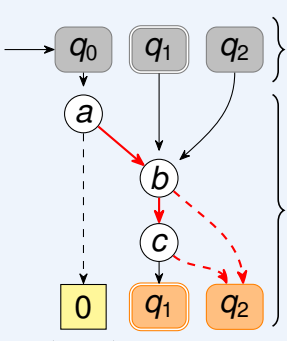
MTBDDs Against the Alphabet-Explosion Problem

Consider a DBA for $a \wedge \mathrm{GF}(b \wedge c)$:

An Explicit Automaton



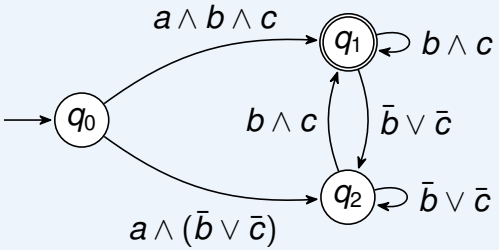
A Semi-Symbolic Encoding



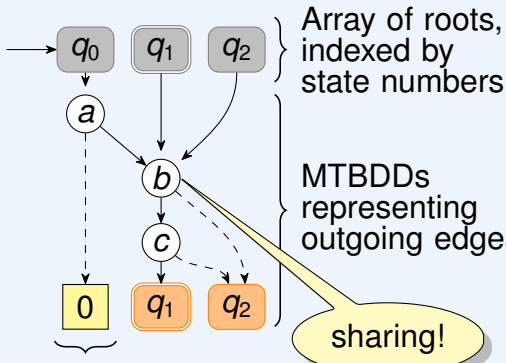
MTBDDs Against the Alphabet-Explosion Problem

Consider a DBA for $a \wedge \mathrm{GF}(b \wedge c)$:

An Explicit Automaton



A Semi-Symbolic Encoding



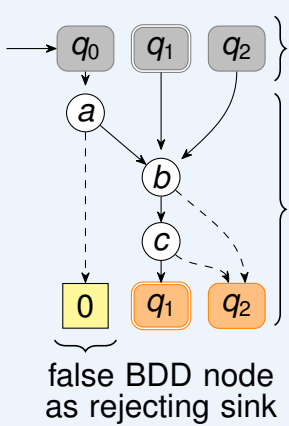
MTBDDs Against the Alphabet-Explosion Problem

Originally used by Mona to implement:

- DFAs over propositional alphabets,
- Boolean operations over DFAs,
- Minimization of DFAs.

Successfully used by several $\mathrm{LTL_f}$ synthesis tools.

A Semi-Symbolic Encoding

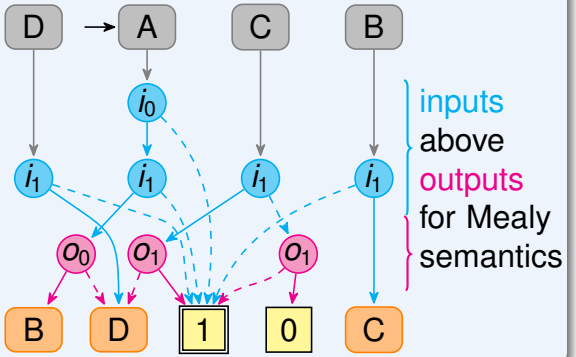


Henriksen et al. *Mona: Monadic second-order logic in practice.* **TACAS'95.** doi

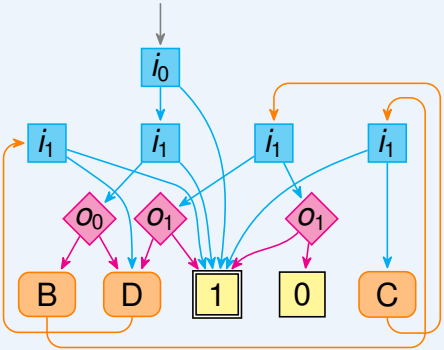
Previous Work: Spot's `ltlfsynt` Tool

- On-the-fly translation of $\mathrm{LTL_f}$ to MTBDD-based DFA,
- Interpretation of the MTBDD-based DFA directly as a reachability game,
- Game solved on-the-fly during its construction.

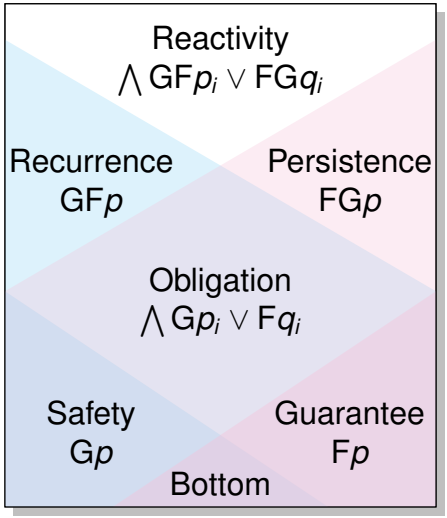
The DFA



The Game Interpretation

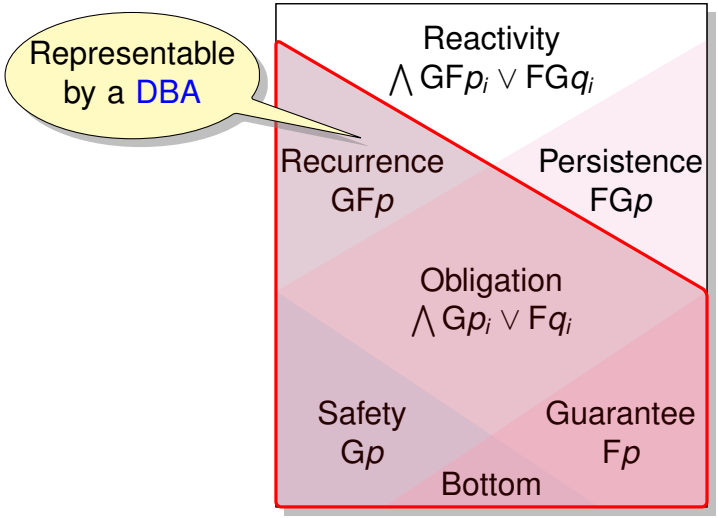


Going Back to LTL and ω-Automata



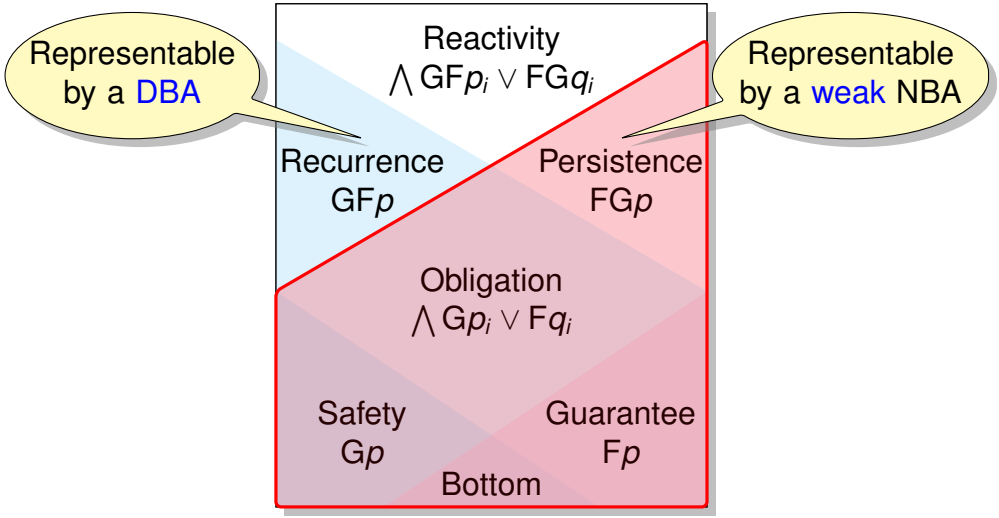
Manna and Pnueli. *A hierarchy of temporal properties*. **PODC'90**. doi

Going Back to LTL and ω-Automata



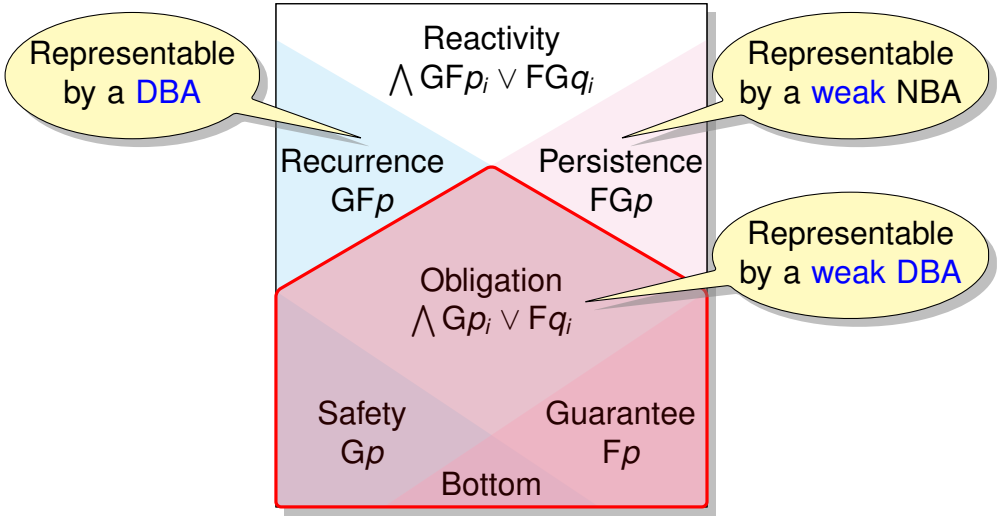
Manna and Pnueli. *A hierarchy of temporal properties*. **PODC'90**. doi

Going Back to LTL and ω-Automata



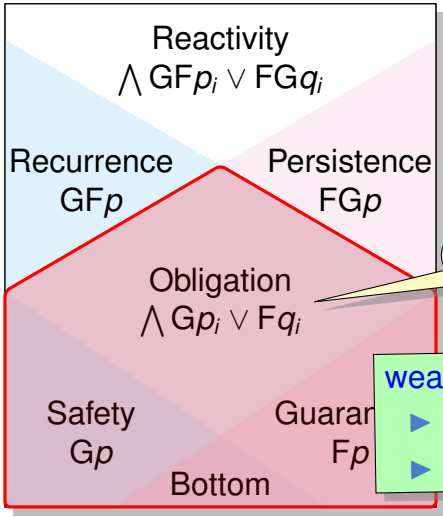
Manna and Pnueli. *A hierarchy of temporal properties*. **PODC'90**. doi

Going Back to LTL and ω-Automata



Manna and Pnueli. *A hierarchy of temporal properties*. **PODC'90**. doi

Going Back to LTL and ω-Automata

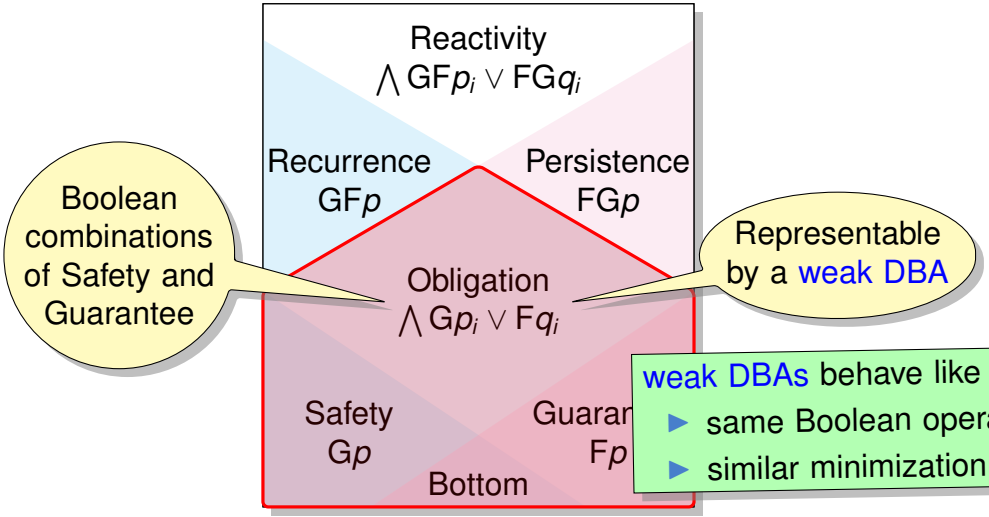


Representable by a weak DBA

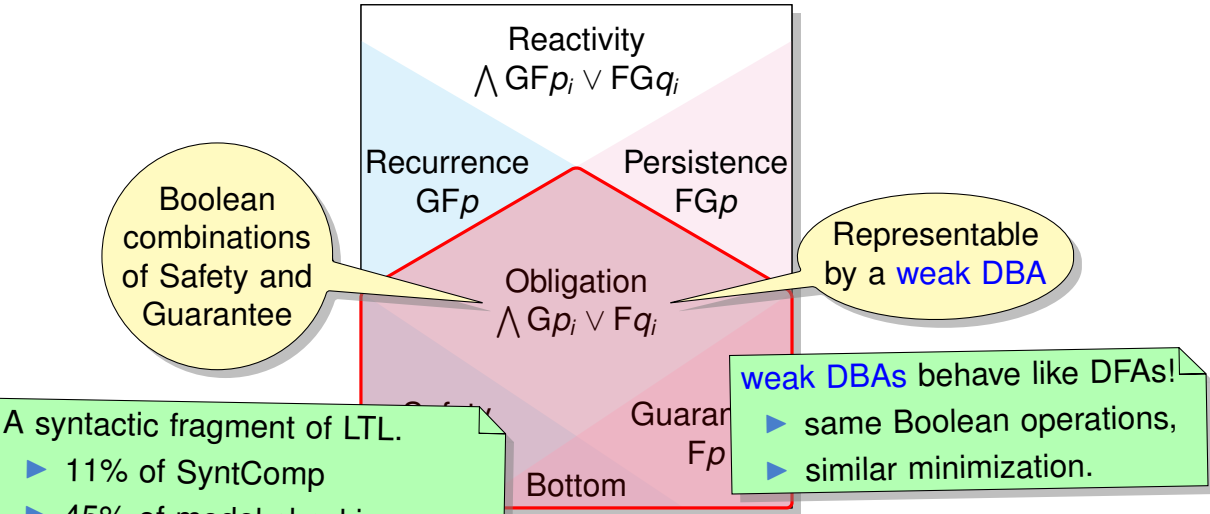
weak DBAs behave like DFAs!

- same Boolean operations,
- similar minimization.

Going Back to LTL and ω-Automata



Going Back to LTL and ω-Automata



Contributions

A new translation from syntactic obligations to weak DBA

- Produces an MTBDD-encoded weak DBA directly
- States are labeled by obligation formulas
- Acceptance of a state is decided by looking at the Boolean combination of safety and guarantee properties its formula represents.
- Minimization is cheap in practice.

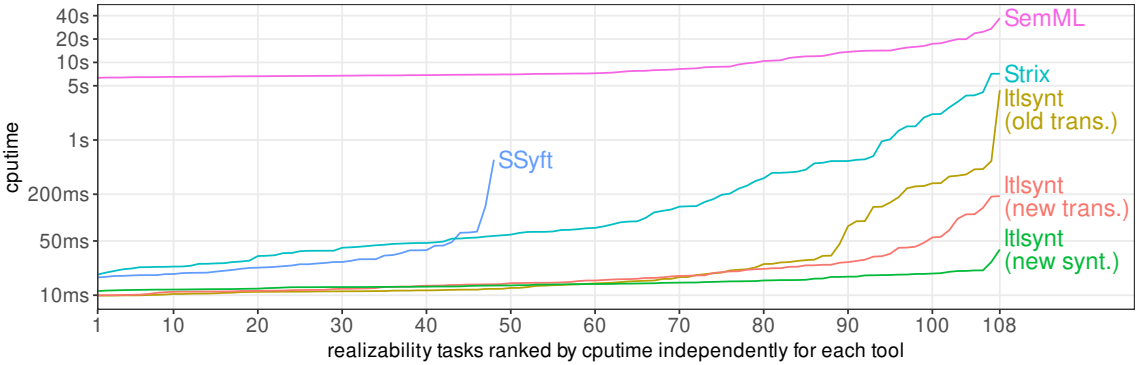
Faster reactive synthesis for syntactic obligations

- Translate the specification to MTBDD on-the-fly...
- ... while solving a weak game (linear)

All of this is implemented in Spot 2.15  www

Improving `ltlsynt` on Obligations

The SyntComp repository contains 108 obligations.



Conclusion

- "Obligation" as an **interesting fragment** of LTL:
  - Translation from obligations to WDBA is easy. — contrib.
  - WDBA can be minimized in polynomial time. — one optimization in paper
  - Weak games can be solved in linear time.
- **MTBDD** to represent **deterministic** automata:
  - A compact way to cope with propositional alphabets.
  - With the right order, can be interpreted as a game directly. — contrib.
- **Implemented** in Spot 2.15 
  - Translation from obligations to MTBDD-based WDBA. — contrib.
  - Minimization of WDBA using MTBDDs. — contrib.
  - On-the-fly synthesis (not just realizability) using MTBDDs. — contrib.

Warp Zone

Fast Obligation Synthesis

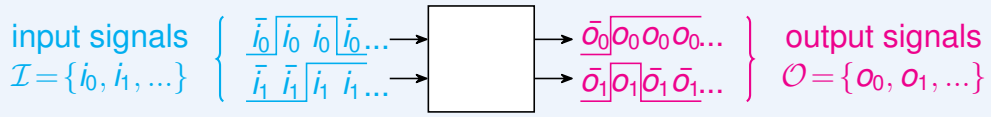
1. Title 2. MTBDD 3. CIAA'25 4. Hierarchy 5. Contributions 6. Speedup 7. Conclusion

Extra Slides

9. Synthesis Problem 10. Synthesis Pipeline 11. MTDBA 12. LTL→MTBDD
13. Obligations 14. λ 15. Game view 16. Propositional Equivalence 17. Obligations Frequency
18. LTL→MTWDBA 19. Weak Games 20. On-the-fly

Reactive Synthesis in a Nutshell

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



LTL reactive synthesis problems

Given an LTL specification relating input signals and output signals over time:

Realizability: decide if a controller exists; our focus

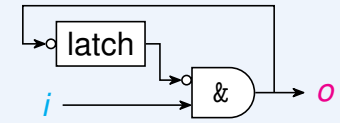
Synthesis: construct it (e.g., as an And-Inverter Graph).

Textbook Reactive Synthesis

1. LTL Spec.

$i \leftrightarrow \mathsf{F}(o)$

6. Output AIG



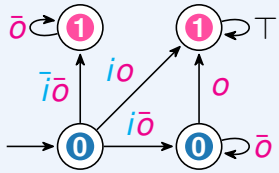
Textbook Reactive Synthesis

1. LTL Spec.

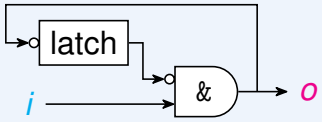
$i \leftrightarrow \mathsf{F}(o)$

2. DPA (max odd)

$\mathsf{Fin}(0) \vee \mathsf{Inf}(1)$



6. Output AIG

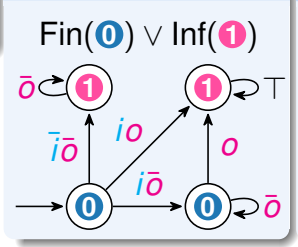


Textbook Reactive Synthesis

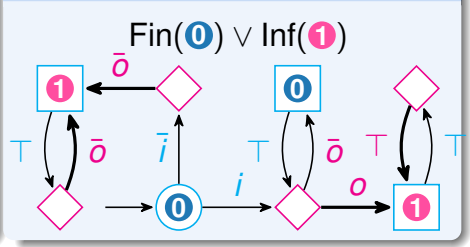
1. LTL Spec.

$i \leftrightarrow \mathsf{F}(o)$

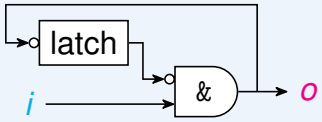
2. DPA (max odd)



3. Parity Game (max odd)



6. Output AIG

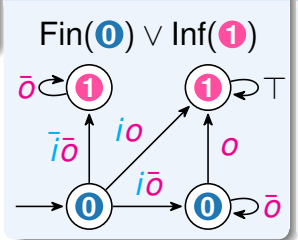


Textbook Reactive Synthesis

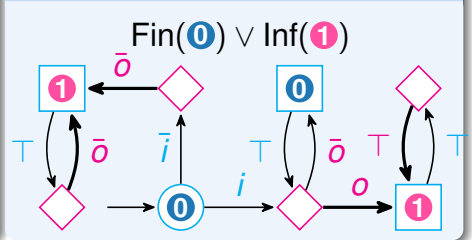
1. LTL Spec.

$i \leftrightarrow \mathsf{F}(o)$

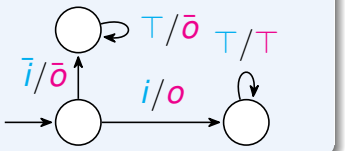
2. DPA (max odd)



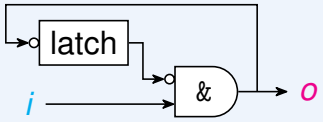
3. Parity Game (max odd)



4. Winning Strategy



6. Output AIG

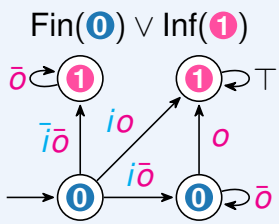


Textbook Reactive Synthesis

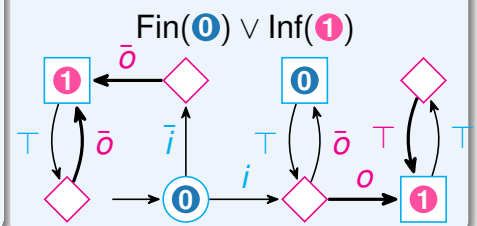
1. LTL Spec.

$i \leftrightarrow \mathsf{F}(o)$

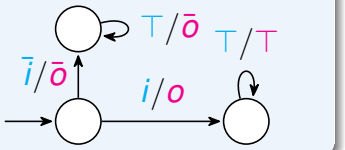
2. DPA (max odd)



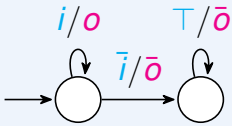
3. Parity Game (max odd)



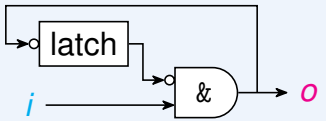
4. Winning Strategy



5. Simplified Strategy



6. Output AIG

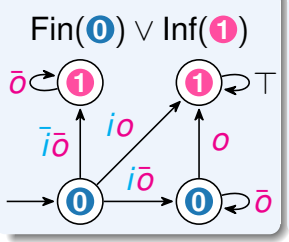


Textbook Reactive Realizability

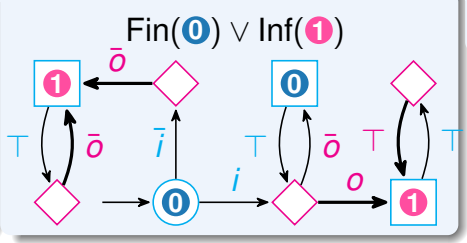
1. LTL Spec.

$i \leftrightarrow \mathsf{F}(o)$

2. DPA (max odd)



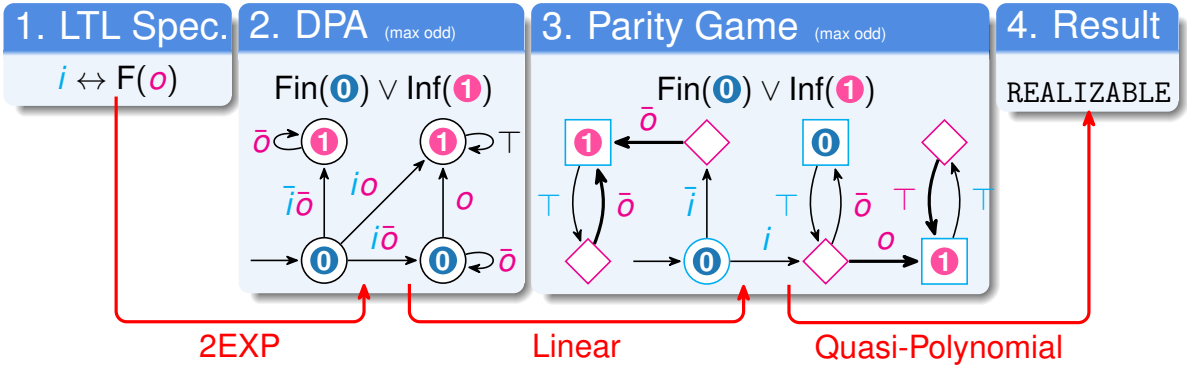
3. Parity Game (max odd)



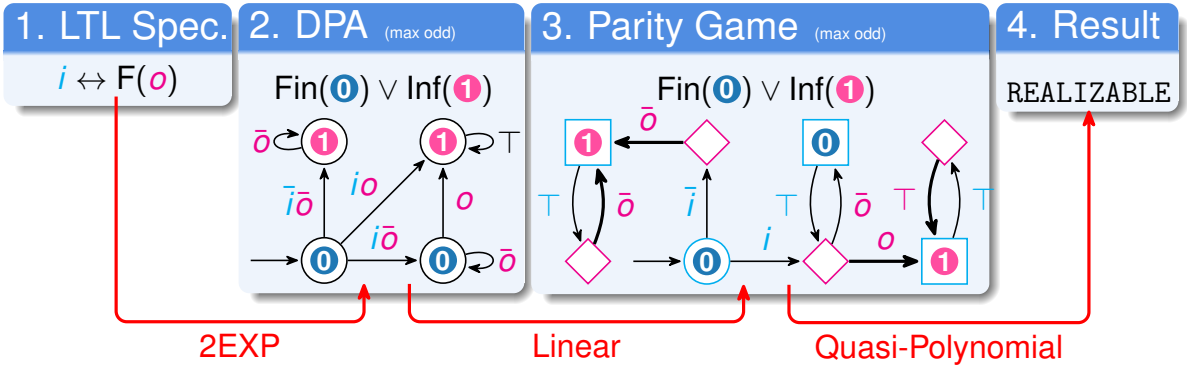
4. Result

REALIZABLE

Textbook Reactive Realizability

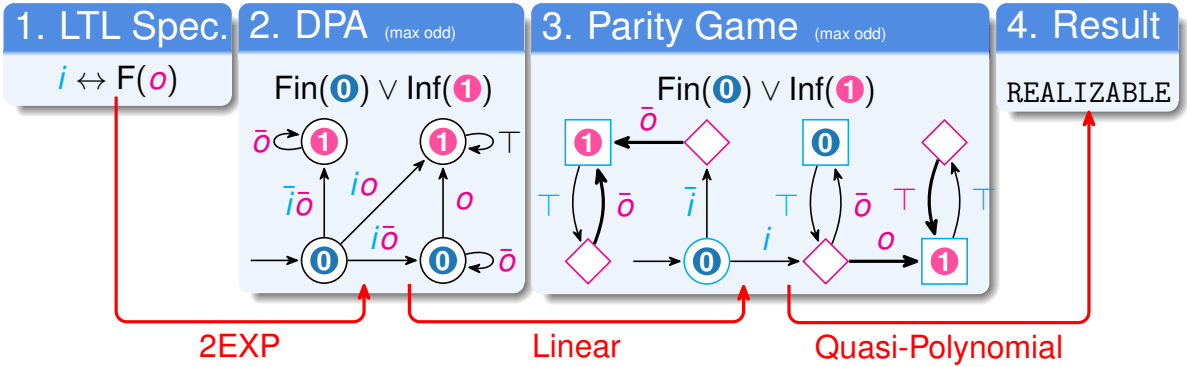


Textbook Reactive Realizability



Contribution: faster pipeline for the “syntactic obligation” fragment of LTL.

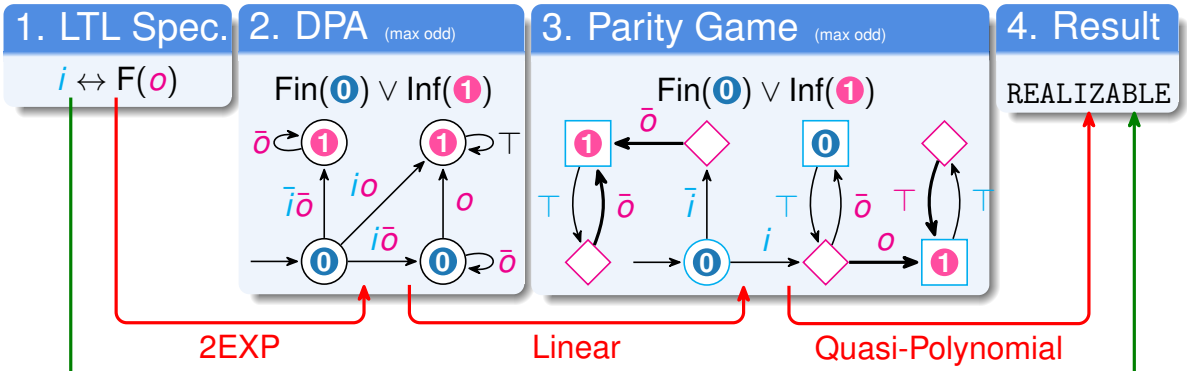
Reactive Realizability



Contribution: faster pipeline for the “syntactic obligation” fragment of LTL.

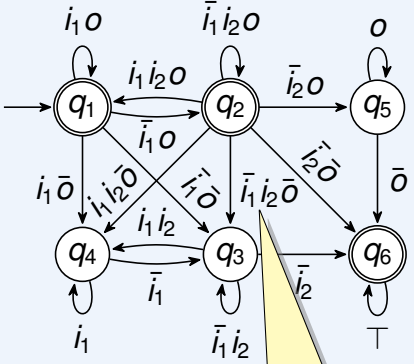
MTBDD-based repr. of Weak DBA, usable directly as a game

Reactive Realizability



MTBDD-based Representation of a DBA

An “explicit” DBA



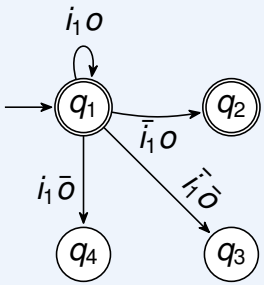
alphabet explosion problem:

$\Sigma = 2^{\mathcal{I} \cup \mathcal{O}}$

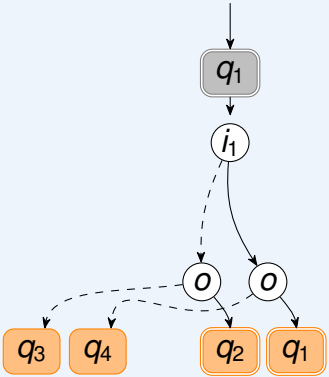
A “semi-symbolic” DBA

MTBDD-based Representation of a DBA

An “explicit” DBA

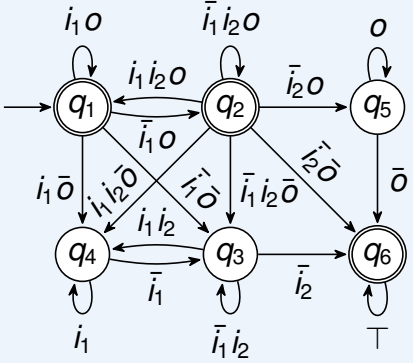


A “semi-symbolic” DBA

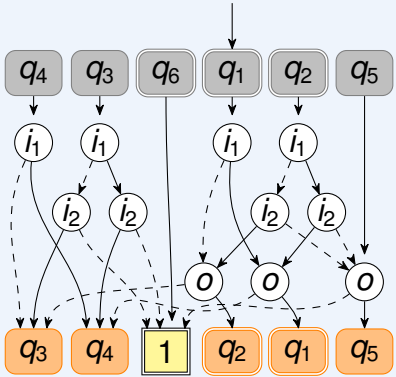


MTBDD-based Representation of a DBA

An “explicit” DBA

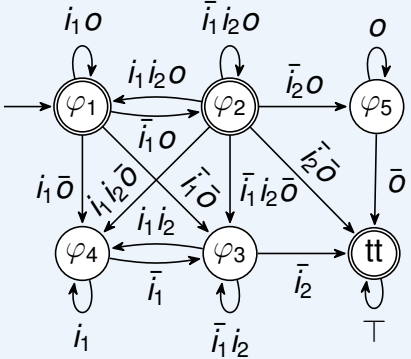


A “semi-symbolic” DBA

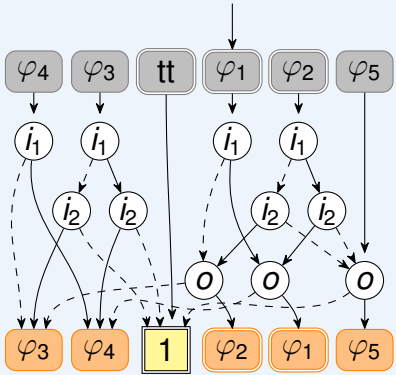


MTBDD-based Representation of an LTL-labeled DBA

An “explicit” DBA



A “semi-symbolic” DBA



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

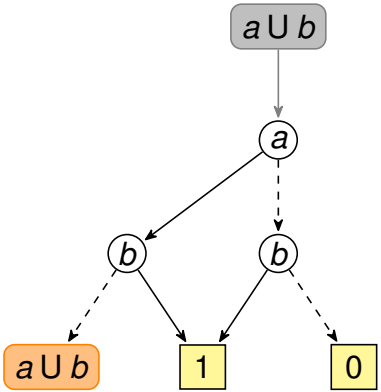
$\varphi_2 = (i_2 \wedge \mathsf{G}(i_1 \vee \mathsf{X} i_2)) \leftrightarrow \mathsf{G} o$

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

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

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

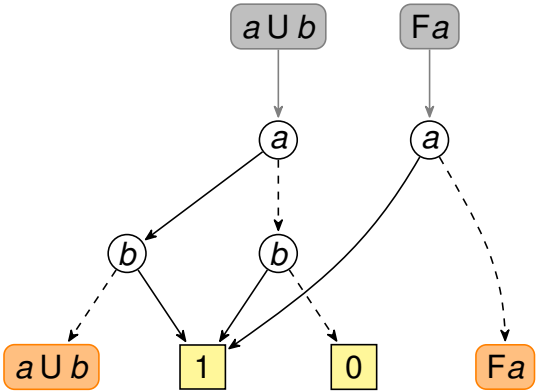
From LTL to MTBDD



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

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

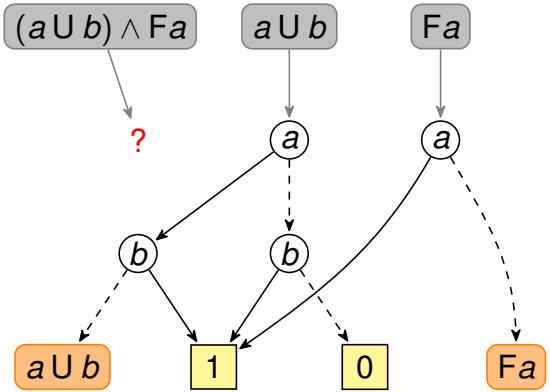
From LTL to MTBDD



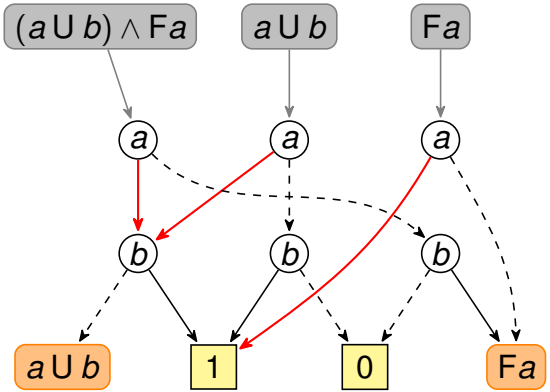
This is a deterministic representation of the *next normal form* (XNF):

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

From LTL to MTBDD

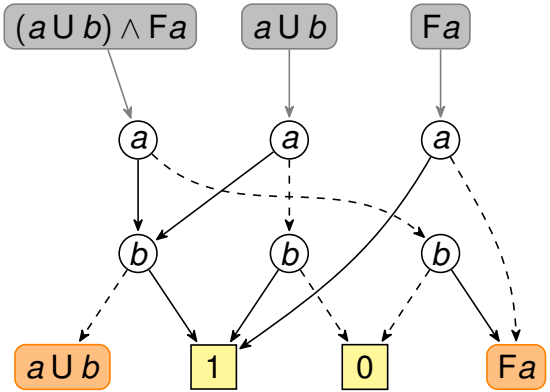


From LTL to MTBDD



Classical BDD `apply` procedure, but combine terminals with "∧".
(Leaves 0 and 1 can help shortcut the recursion.)

From LTL to MTBDD



We are just missing a way to decide which states are accepting.

This is where the restriction to obligations matters.

Obligations = Weak Deterministic Büchi Automata

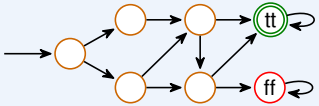
Bottom (Safety ∩ Guarantee)

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

Obligations = Weak Deterministic Büchi Automata

Bottom (Safety ∩ Guarantee)

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

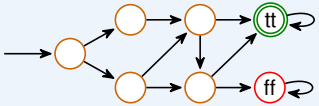


DBA is a DAG, until (tt) or (ff)

Obligations = Weak Deterministic Büchi Automata

Bottom (Safety ∩ Guarantee)

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



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

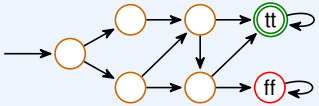
Safety

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

Obligations = Weak Deterministic Büchi Automata

Bottom (Safety ∩ Guarantee)

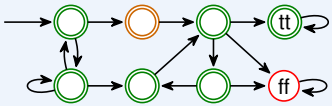
$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$



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

Safety

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

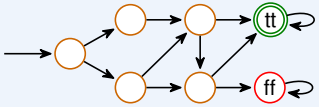


all states but (ff) are accepting

Obligations = Weak Deterministic Büchi Automata

Bottom (Safety ∩ Guarantee)

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



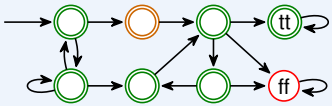
DBA is a DAG, until (tt) or (ff)

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

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

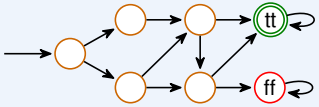


all states but (ff) are accepting

Obligations = Weak Deterministic Büchi Automata

Bottom (Safety ∩ Guarantee)

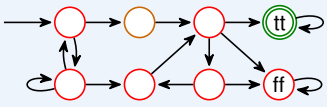
$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$



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

Guarantee

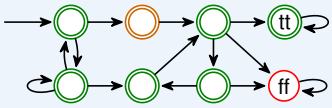
$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \mathsf{X}\varphi_G \mid \mathsf{F}\varphi_G \mid \varphi_G \,\mathsf{U}\, \varphi_G$



all states but (tt) are rejecting

Safety

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

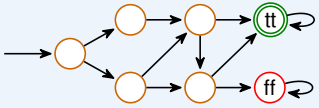


all states but (ff) are accepting

Obligations = Weak Deterministic Büchi Automata

Bottom (Safety ∩ Guarantee)

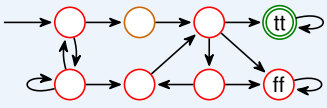
$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$



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

Guarantee

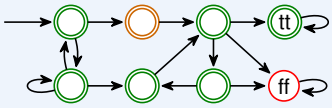
$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \neg\varphi_S \mid \mathsf{X}\varphi_G \mid \mathsf{F}\varphi_G \mid \varphi_G \,\mathsf{U}\, \varphi_G$



all states but (tt) are rejecting

Safety

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

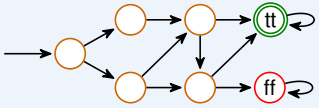


all states but (ff) are accepting

Obligations = Weak Deterministic Büchi Automata

Bottom (Safety ∩ Guarantee)

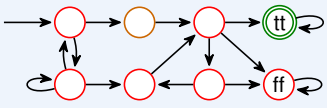
$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$



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

Guarantee

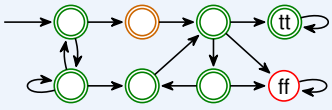
$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \neg\varphi_S \mid \mathsf{X}\varphi_G \mid \mathsf{F}\varphi_G \mid \varphi_G \,\mathsf{U}\, \varphi_G$



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

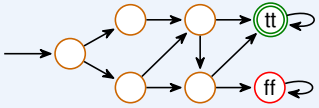
Obligation

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

Obligations = Weak Deterministic Büchi Automata

Bottom (Safety ∩ Guarantee)

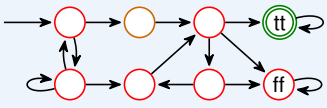
$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$



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

Guarantee

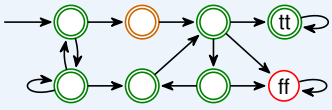
$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \neg\varphi_S \mid \mathsf{X}\varphi_G \mid \mathsf{F}\varphi_G \mid \varphi_G \,\mathsf{U}\, \varphi_G$



all states but (tt) are rejecting

Safety

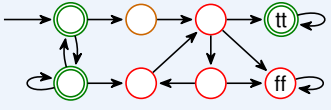
$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \neg\varphi_G \mid \mathsf{X}\varphi_S \mid \mathsf{G}\varphi_S \mid \varphi_S \,\mathsf{R}\, \varphi_S$



all states but (ff) are accepting

Obligation

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

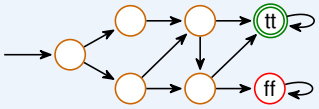


each SCC contains only accepting or rejecting states

Obligations = Weak Deterministic Büchi Automata

Bottom (Safety ∩ Guarantee)

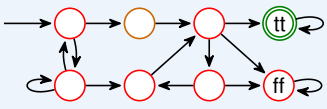
$\varphi_B ::= \mathsf{ff} \mid \mathsf{tt} \mid p \mid \neg\varphi_B \mid \varphi_B \wedge \varphi_B \mid \varphi_B \vee \varphi_B \mid \mathsf{X}\varphi_B$



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

Guarantee

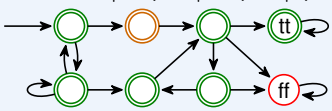
$\varphi_G ::= \varphi_B \mid \varphi_G \wedge \varphi_G \mid \varphi_G \vee \varphi_G \mid \neg\varphi_S \mid \mathsf{X}\varphi_G \mid \mathsf{F}\varphi_G \mid \varphi_G \,\mathsf{U}\, \varphi_G$



all states but (tt) are rejecting

Safety

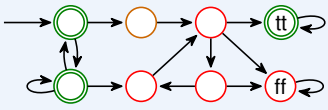
$\varphi_S ::= \varphi_B \mid \varphi_S \wedge \varphi_S \mid \varphi_S \vee \varphi_S \mid \neg\varphi_G \mid \mathsf{X}\varphi_S \mid \mathsf{G}\varphi_S \mid \varphi_S \,\mathsf{R}\, \varphi_S$



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

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

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

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 "$*$"

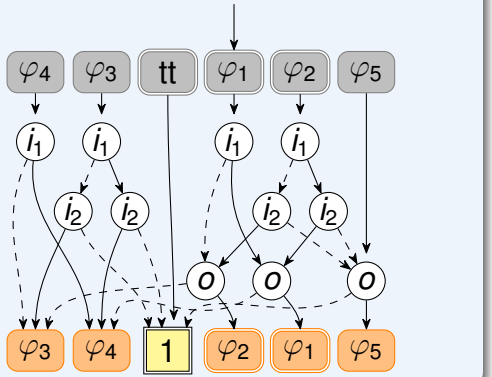
Example:

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

follows from the above rules

MTBDD-based Representation of an LTL-labeled DBA

A “semi-symbolic” WDBA



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

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

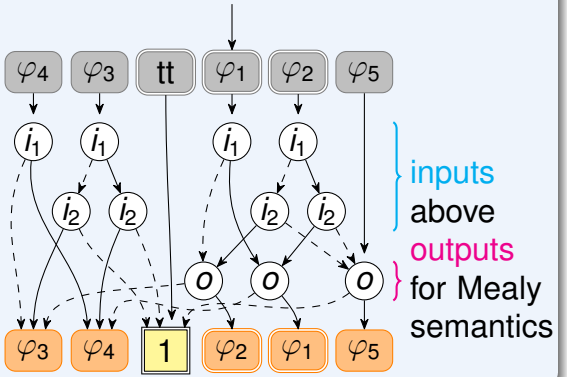
$\varphi_3 = \neg(i_2 \wedge \mathsf{G}(i_1 \vee \mathsf{X}i_2))$

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

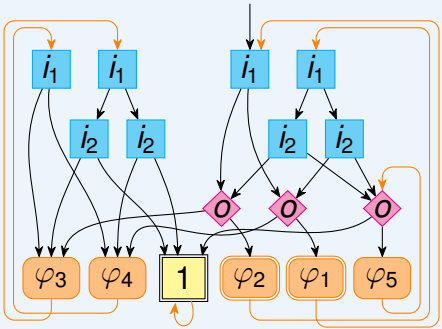
$\varphi_5 = \neg\mathsf{G}o$

Game Interpretation Is Straightforward

A “semi-symbolic” WDBA

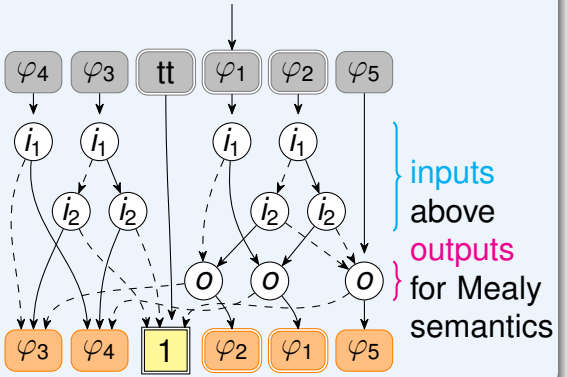


Weak Game Interpretation

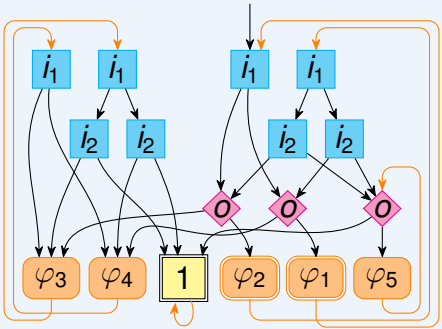


Game Interpretation Is Straightforward

A “semi-symbolic” WDBA

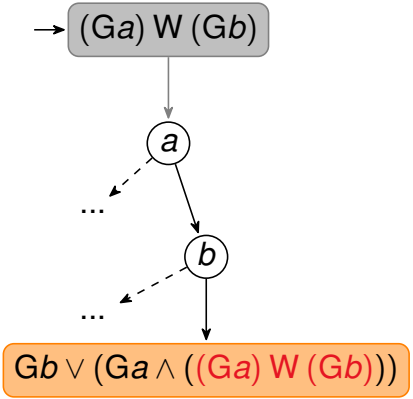


Weak Game Interpretation

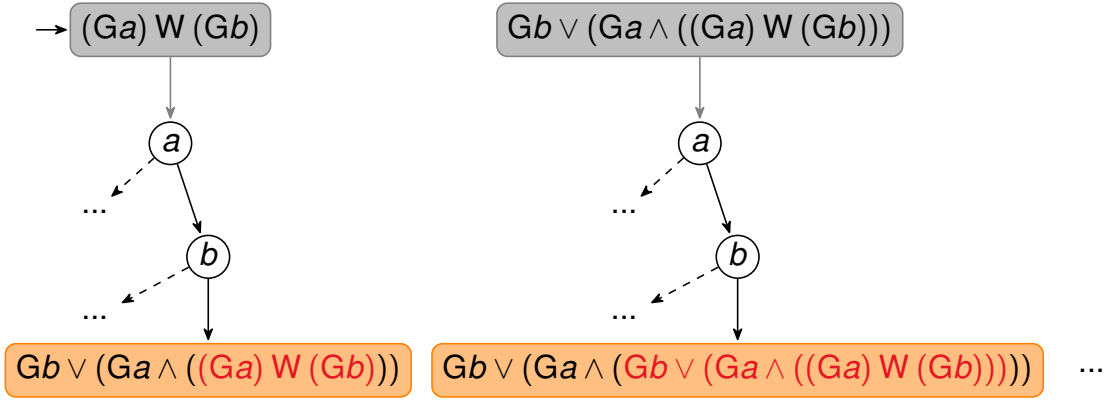


Weak games can be solved in linear time by enumerating SCCs bottom-up.
The whole translation+game solving is implemented on-the-fly.

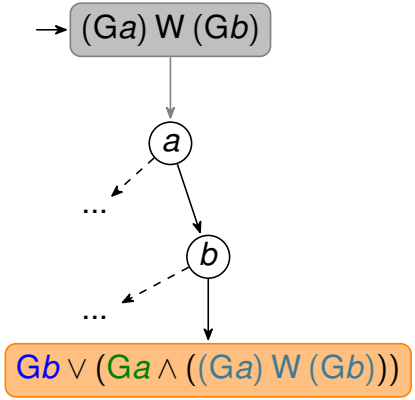
Propositional Equivalence

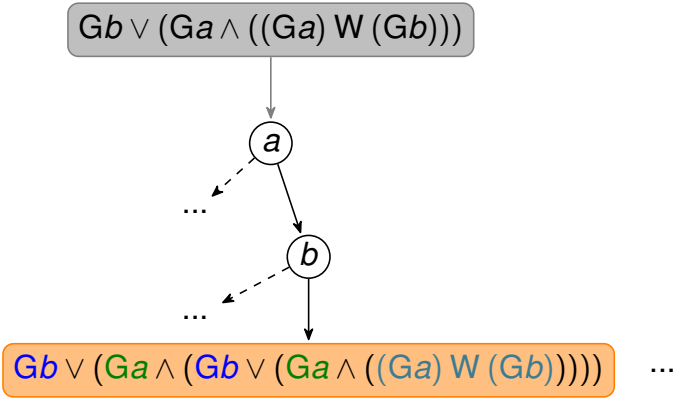


Propositional Equivalence



Propositional Equivalence

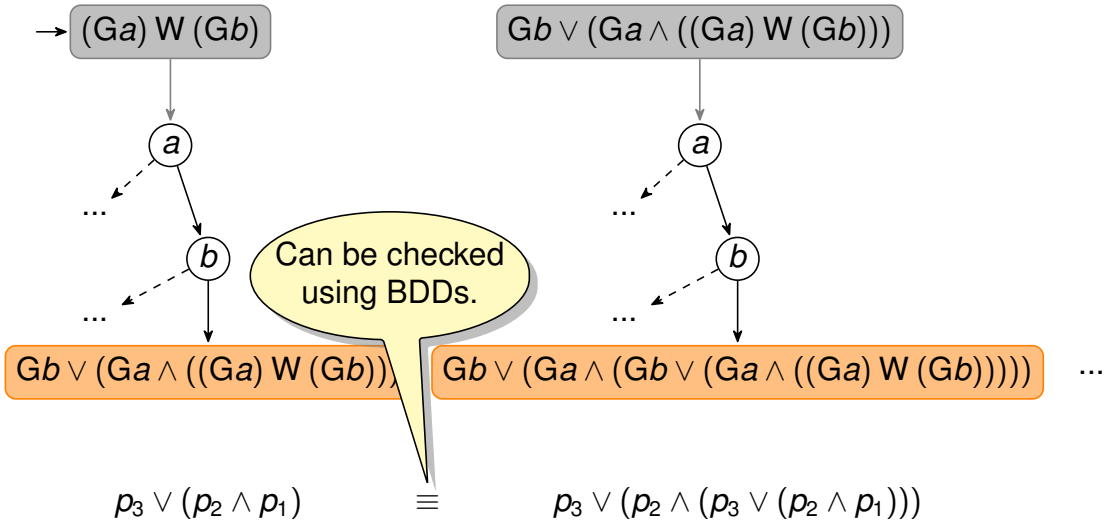




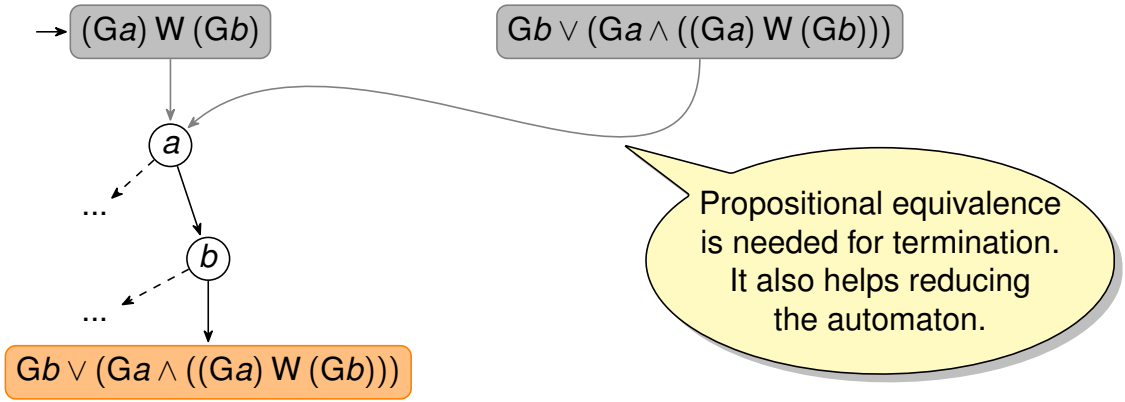
$p_3 \vee (p_2 \wedge p_1)$

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

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

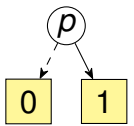

Pelánek. *BEEM: benchmarks for explicit model checkers*. **Spin'07**. doi


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

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 $\odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$

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

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

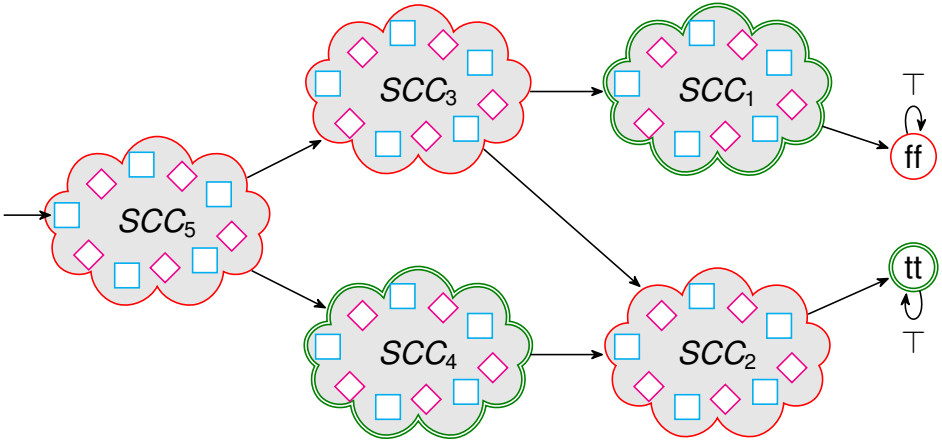
$\mathrm{tr}(\mathsf{F}\alpha) = \mathrm{tr}(\alpha) \vee \mathsf{F}\alpha$

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

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

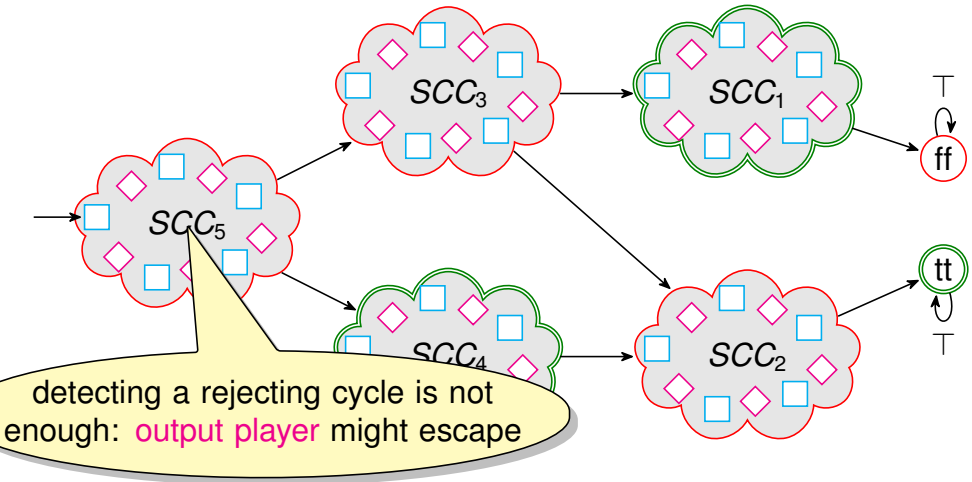
Solving Weak Games Bottom-Up

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.



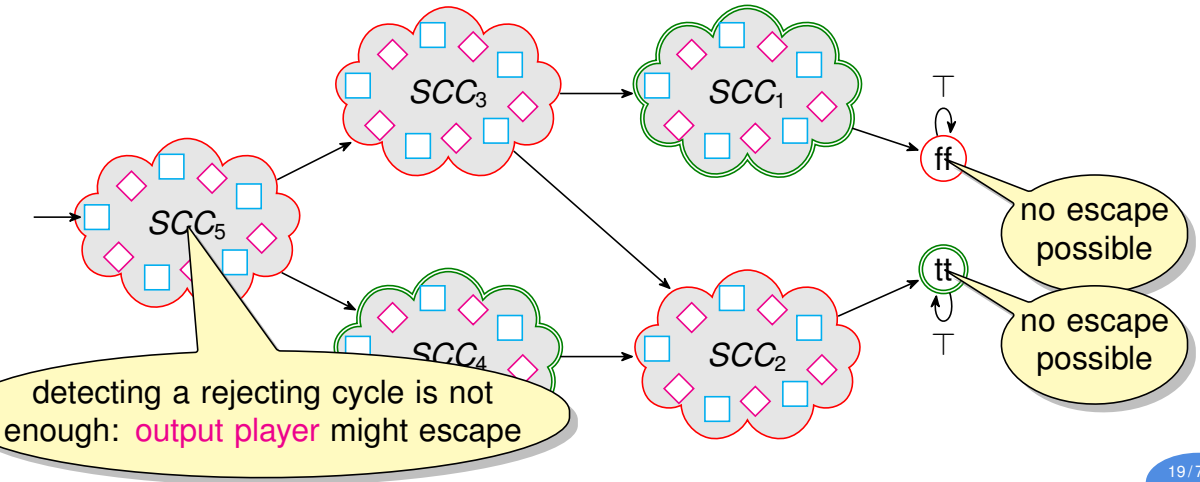
Solving Weak Games Bottom-Up

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.



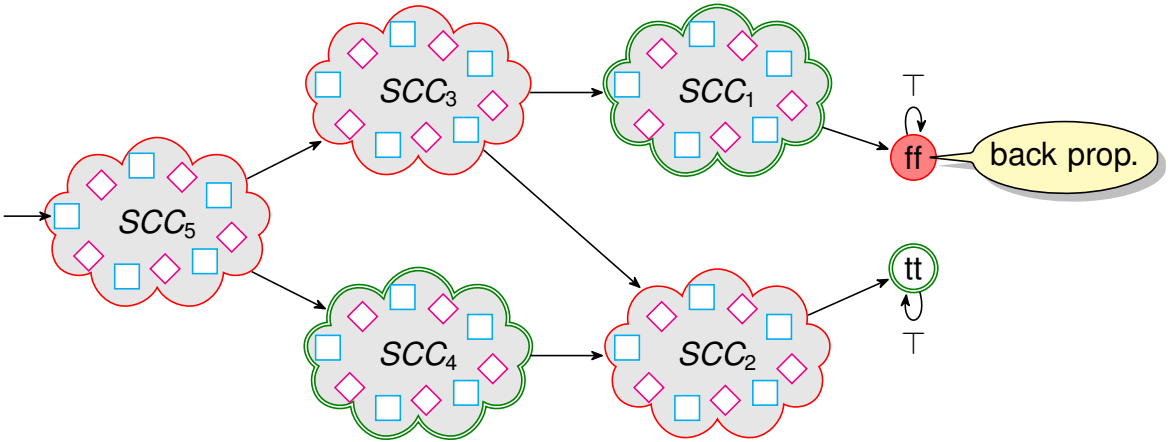
Solving Weak Games Bottom-Up

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.



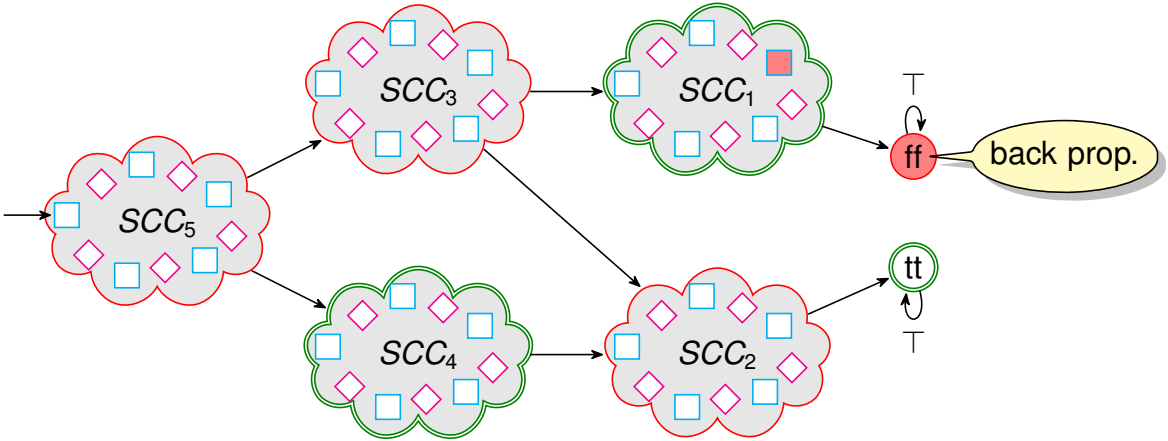
Solving Weak Games Bottom-Up

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.



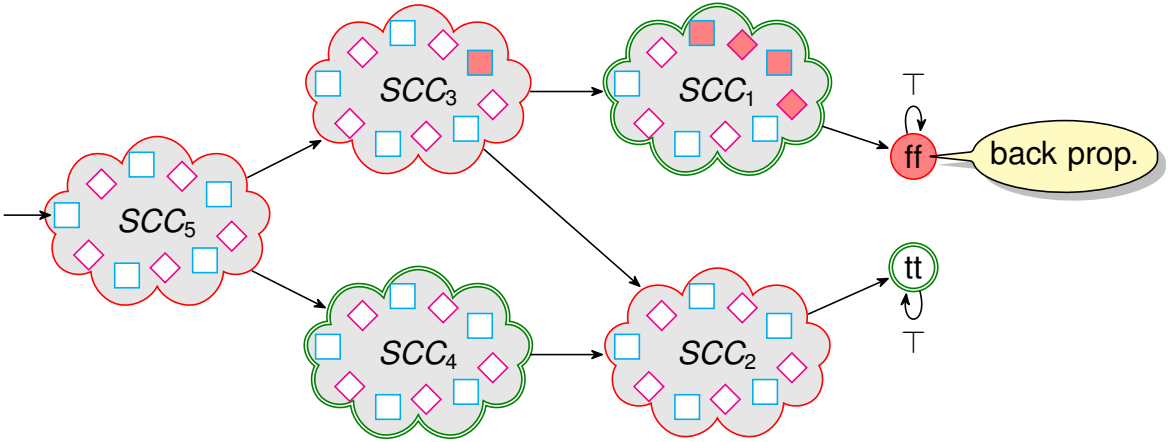
Solving Weak Games Bottom-Up

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.



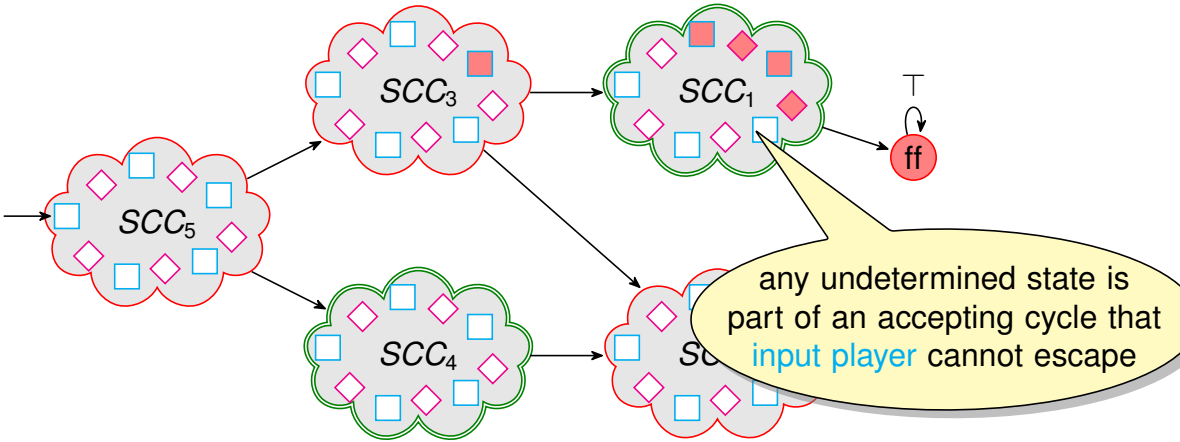
Solving Weak Games Bottom-Up

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.



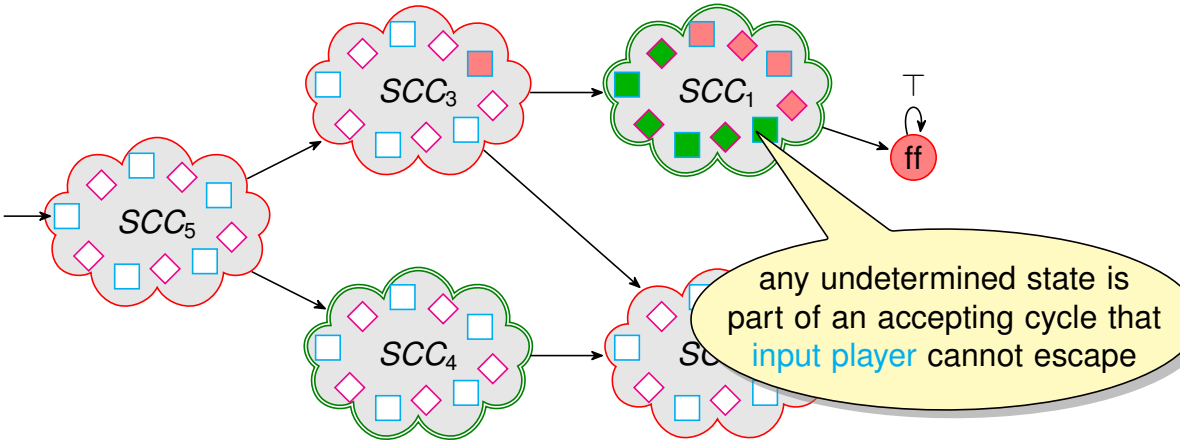
Solving Weak Games Bottom-Up

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.



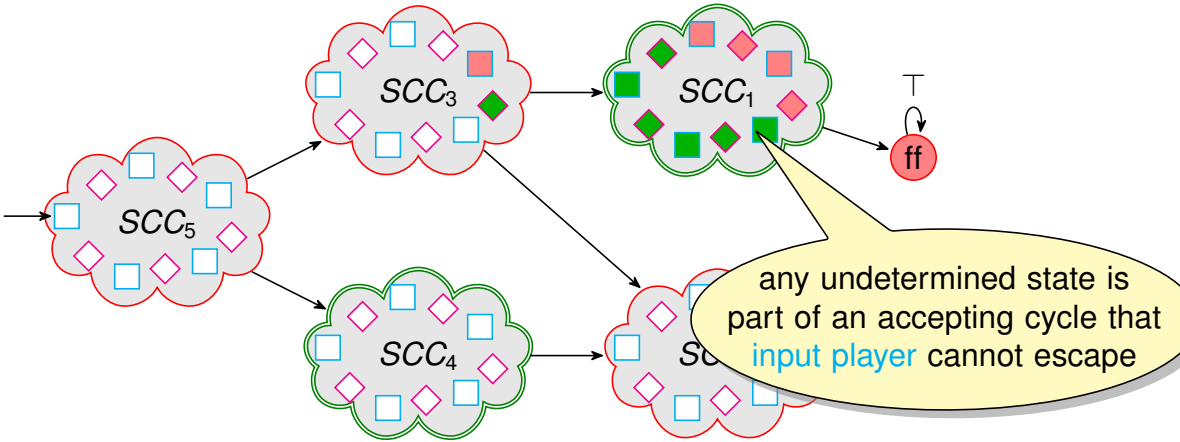
Solving Weak Games Bottom-Up

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.



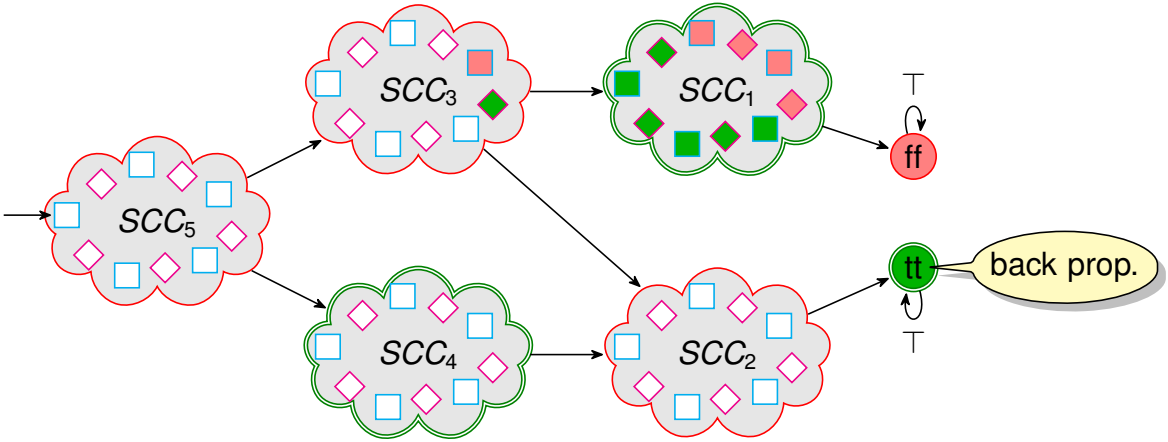
Solving Weak Games Bottom-Up

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.



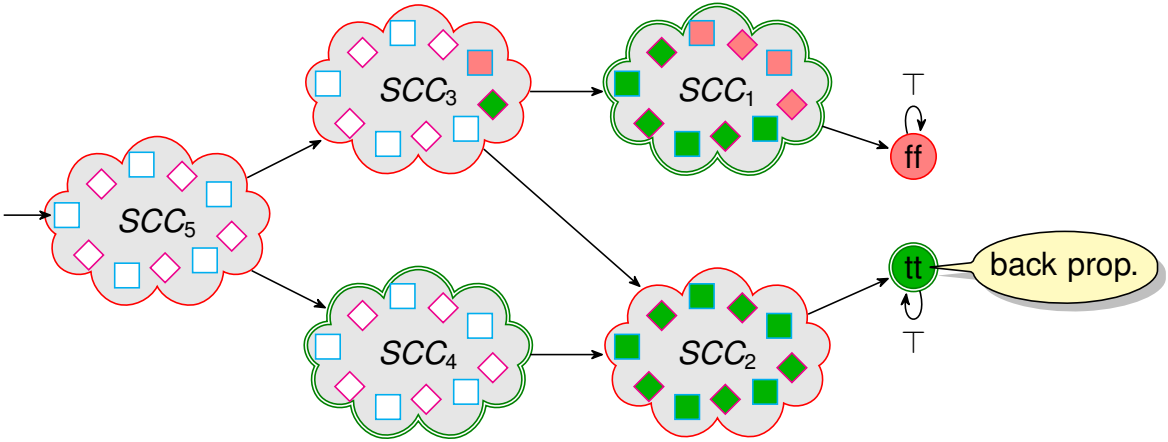
Solving Weak Games Bottom-Up

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.



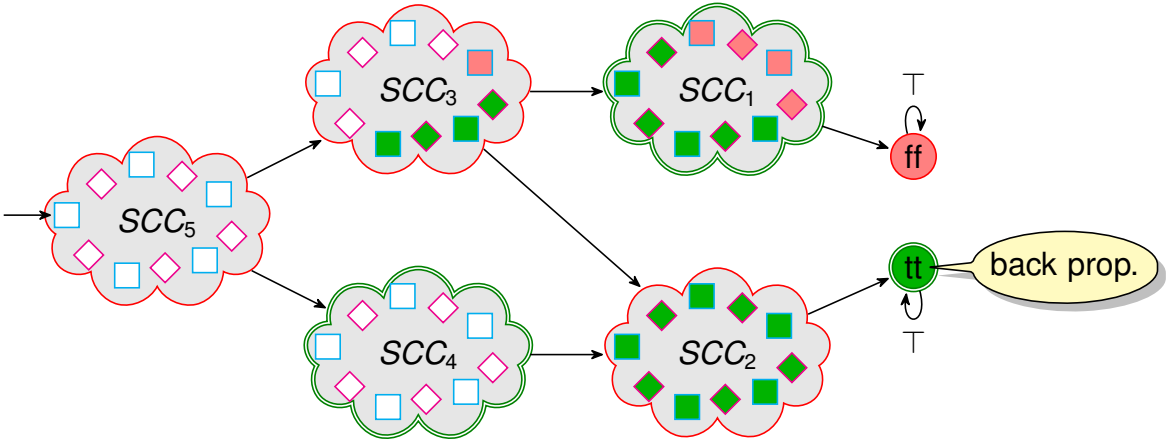
Solving Weak Games Bottom-Up

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.



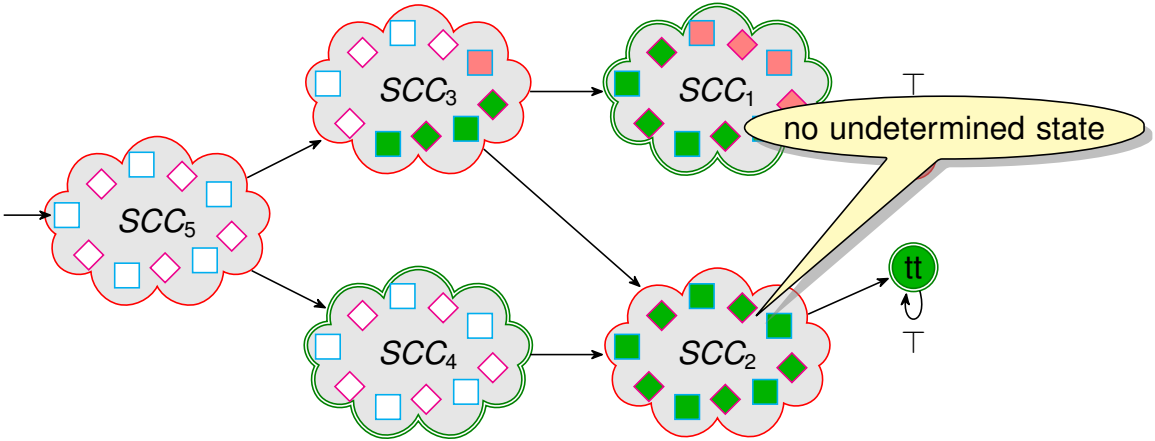
Solving Weak Games Bottom-Up

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.



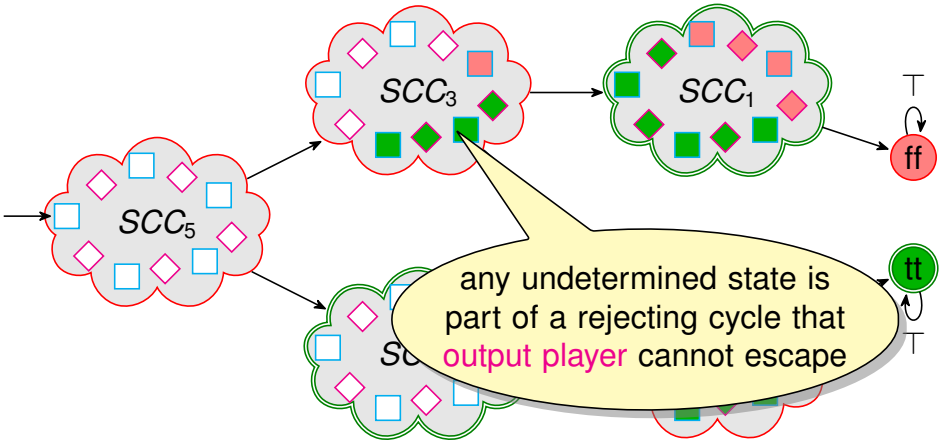
Solving Weak Games Bottom-Up

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.



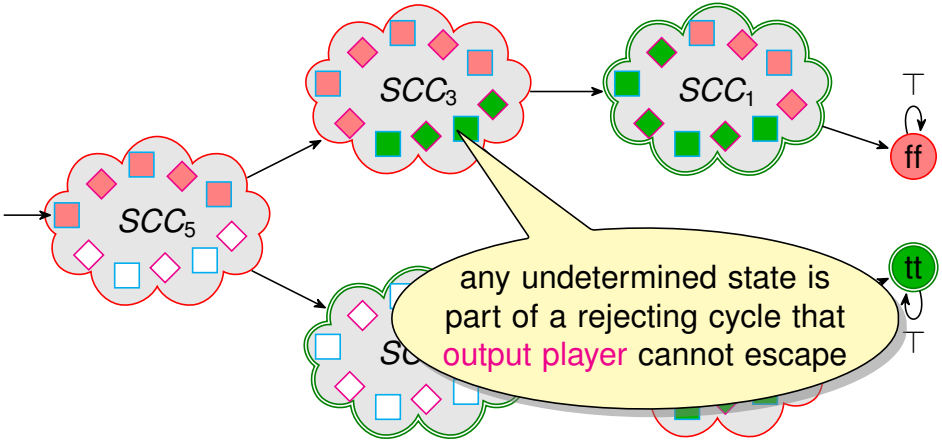
Solving Weak Games Bottom-Up

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.



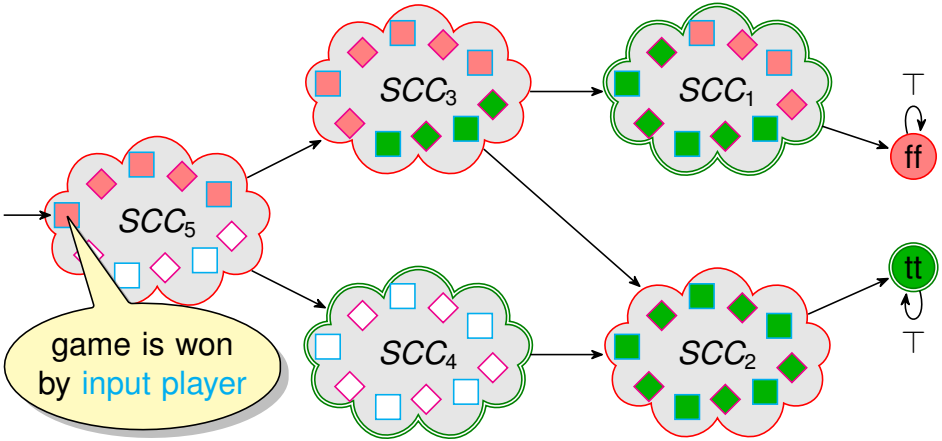
Solving Weak Games Bottom-Up

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.



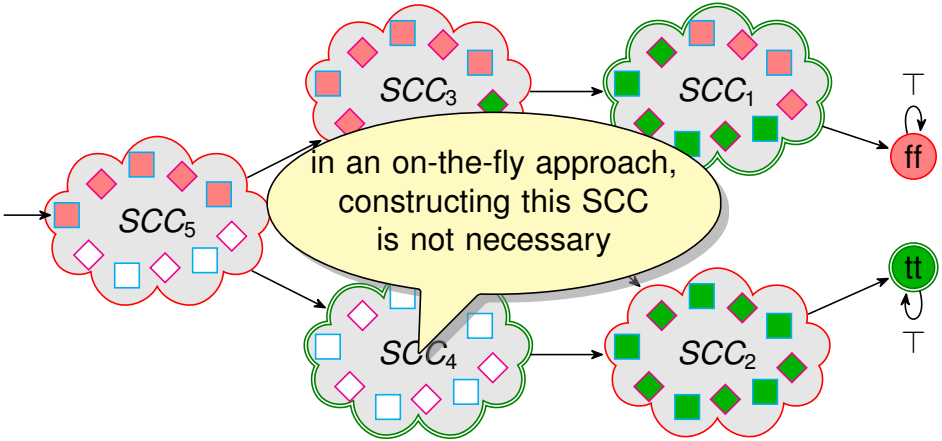
Solving Weak Games Bottom-Up

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.

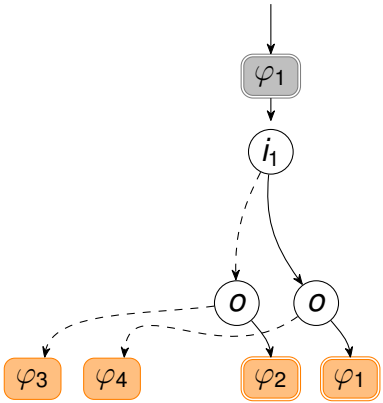


Solving Weak Games Bottom-Up

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.



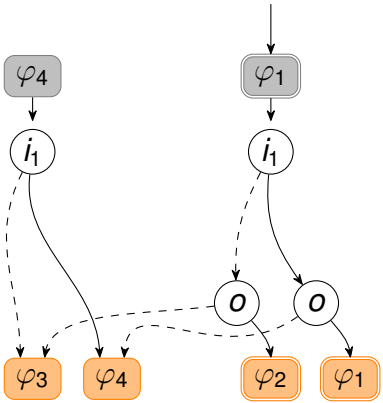
On-the-Fly MTBDD-based Obligation Synthesis



$\{\varphi_1\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis

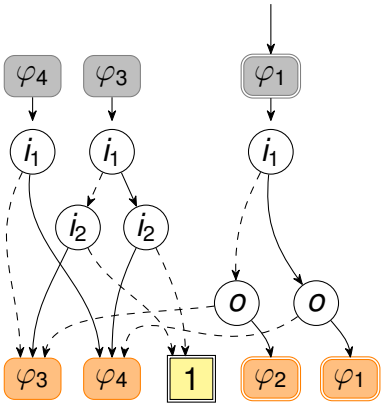


$\{\varphi_1\}$

$\{\varphi_4\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis

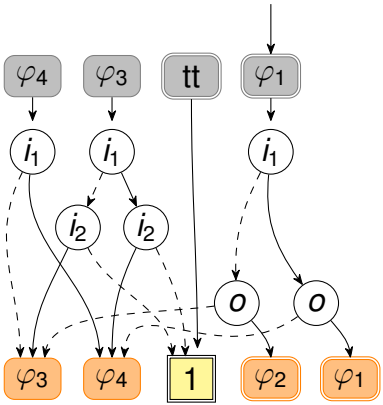


$\{\varphi_1\}$

$\{\varphi_4, \varphi_3\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis



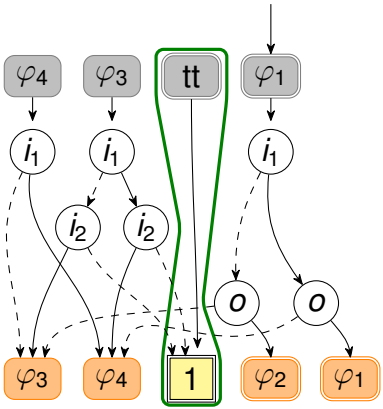
$\{\varphi_1\}$

$\{\varphi_4, \varphi_3\}$

$\{\text{tt}\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis



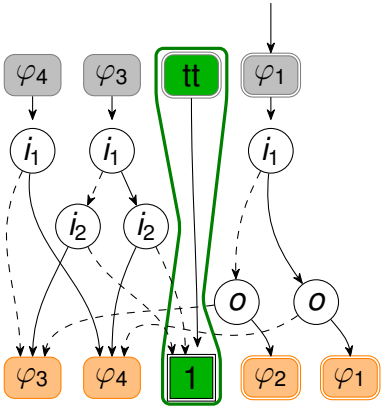
$\{\varphi_1\}$

$\{\varphi_4, \varphi_3\}$

$\{\mathrm{tt}\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis

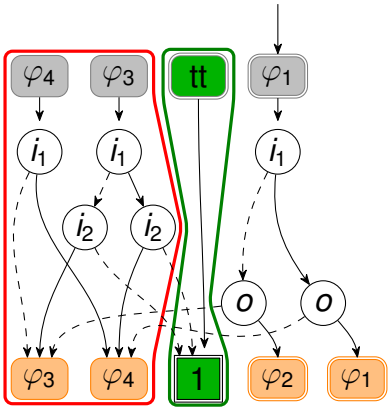


$\{\varphi_1\}$

$\{\varphi_4, \varphi_3\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis

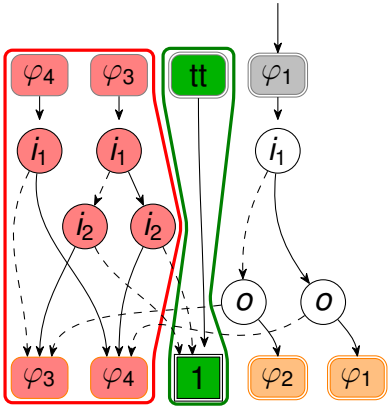


$\{\varphi_1\}$

$\{\varphi_4, \varphi_3\}$

DFS stack organized as partial SCCs

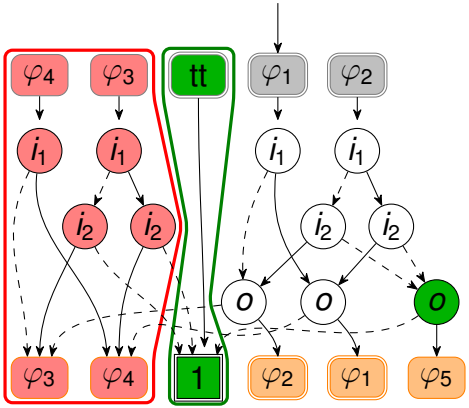
On-the-Fly MTBDD-based Obligation Synthesis



$\{\varphi_1\}$

DFS stack organized as partial SCCs

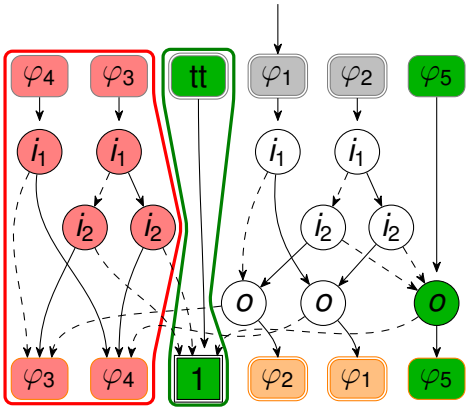
On-the-Fly MTBDD-based Obligation Synthesis



$\{\varphi_1, \varphi_2\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis

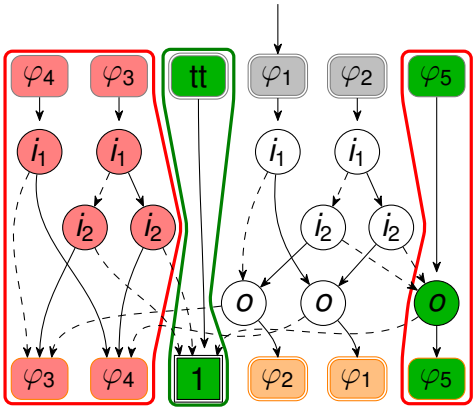


$\{\varphi_1, \varphi_2\}$

$\{\varphi_5\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis

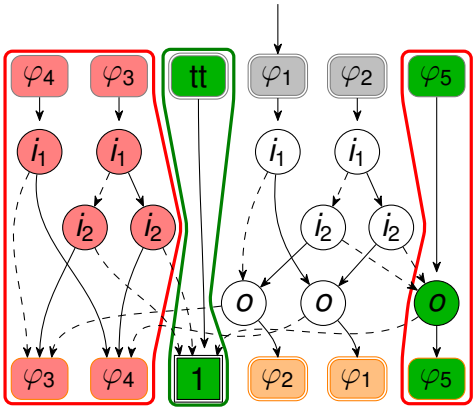


$\{\varphi_1, \varphi_2\}$

$\{\varphi_5\}$

DFS stack organized as partial SCCs

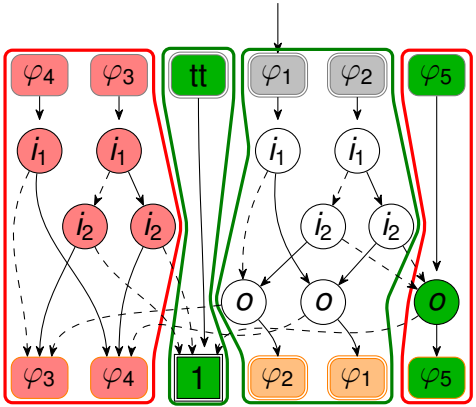
On-the-Fly MTBDD-based Obligation Synthesis



$\{\varphi_1, \varphi_2\}$

DFS stack organized as partial SCCs

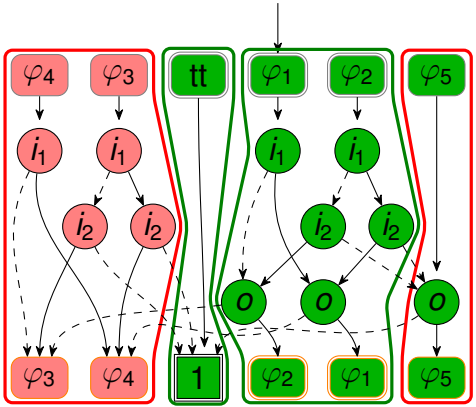
On-the-Fly MTBDD-based Obligation Synthesis



$\{\varphi_1, \varphi_2\}$

DFS stack organized as partial SCCs

On-the-Fly MTBDD-based Obligation Synthesis



DFS stack organized as partial SCCs