

# Language-Theoretic Synthesis & Summaries for Context-Free Games

---

Lukáš Holík<sup>1</sup>, Roland Meyer<sup>2</sup>, and Sebastian Muskalla<sup>2</sup>

1 Brno University of Technology, [holik@fit.vutbr.cz](mailto:holik@fit.vutbr.cz)

2 TU Braunschweig, [{roland.meyer, s.muskalla}@tu-braunschweig.de](mailto:{roland.meyer, s.muskalla}@tu-braunschweig.de)

# Language-Theoretic Verification

---

# Verification

*Verification problem:*

Given: Source code of program  $P$  and specification  $\varphi$  .

Question: Does **runtime behavior** of  $P$  satisfy  $\varphi$  ?

# Verification

*Verification problem:*

Given: Source code of program  $P$  and specification  $\varphi$  .

Question: Does **runtime behavior** of  $P$  satisfy  $\varphi$  ?

*Language-theoretic approach:*

$\mathcal{L}_P$  = possible program executions

$\mathcal{L}_\varphi$  = valid executions

Decide:  $\mathcal{L}_P \subseteq \mathcal{L}_\varphi$

# Language-theoretic verification

$\mathcal{L}_P$  = possible program executions

$\mathcal{L}_\varphi$  = valid executions

# Language-theoretic verification

$\mathcal{L}_P$  = possible program executions

$\mathcal{L}_\varphi$  = valid executions

Good:  $\mathcal{L}_\varphi$  usually easy (regular)

Bad:  $\mathcal{L}_P$  usually not even context free

# Language-theoretic verification

$\mathcal{L}_P$  = possible program executions

$\mathcal{L}_\varphi$  = valid executions

Good:  $\mathcal{L}_\varphi$  usually easy (regular)

Bad:  $\mathcal{L}_P$  usually not even context free

↳ Problem is undecidable

↳ Need to approximate  $\mathcal{L}_P$

# Language-theoretic verification

*Semantics:*

$$\mathcal{L}_P = \mathcal{L}_{CF} \cap \mathcal{L}_{Data}$$



# Language-theoretic verification

*Semantics:*

$$\mathcal{L}_P = \mathcal{L}_{CF} \cap \mathcal{L}_{Data} = \mathcal{L}_{CF} \cap \bigcap_{x \in Var} \mathcal{L}_x$$

# Language-theoretic verification

*Semantics:*

$$\mathcal{L}_P = \mathcal{L}_{CF} \cap \mathcal{L}_{Data} = \mathcal{L}_{CF} \cap \bigcap_{x \in Var} \mathcal{L}_x$$

$\mathcal{L}_{CF}$  is context free

# Language-theoretic verification

*Semantics:*

$$\mathcal{L}_P = \mathcal{L}_{CF} \cap \mathcal{L}_{Data} = \mathcal{L}_{CF} \cap \bigcap_{x \in Var} \mathcal{L}_x$$

$\mathcal{L}_{CF}$  is context free

$\mathcal{L}_{Data}$  is anything:  $Var$  is infinite and  $\mathcal{L}_x$  is arbitrary

# Language-theoretic verification

*Semantics:*

$$\mathcal{L}_P = \mathcal{L}_{CF} \cap \mathcal{L}_{Data} = \mathcal{L}_{CF} \cap \bigcap_{x \in Var} \mathcal{L}_x$$

$\mathcal{L}_{CF}$  is context free

$\mathcal{L}_{Data}$  is anything:  $Var$  is infinite and  $\mathcal{L}_x$  is arbitrary

*Lessons in life:*

Handle **control flow** using techniques from **automata theory**

Handle **data** using techniques from **logic**

# Language-theoretic verification

*Semantics:*

$$\mathcal{L}_P = \mathcal{L}_{CF} \cap \mathcal{L}_{Data} = \mathcal{L}_{CF} \cap \bigcap_{x \in Var} \mathcal{L}_x$$

$\mathcal{L}_{CF}$  is context free

$\mathcal{L}_{Data}$  is anything:  $Var$  is infinite and  $\mathcal{L}_x$  is arbitrary

*Lessons in life:*

Handle **control flow** using techniques from **automata theory**

Handle **data** using techniques from **logic**

*Need to combine them*

# Language-theoretic verification

*Semantics:*

$$\mathcal{L}_P = \mathcal{L}_{CF} \cap \mathcal{L}_{Data} = \mathcal{L}_{CF} \cap \bigcap_{x \in Var} \mathcal{L}_x$$

$\mathcal{L}_{CF}$  is context free

$\mathcal{L}_{Data}$  is anything:  $Var$  is infinite and  $\mathcal{L}_x$  is arbitrary

*Lessons in life:*

Handle **control flow** using techniques from **automata theory**

Handle **data** using techniques from **logic**

*Need to combine them*

CEGAR loop [Podelski et al. since 2010]

# Counterexample-guided abstraction refinement

Init  $\mathcal{L}_S := \mathcal{L}_\varphi$

# Counterexample-guided abstraction refinement

Init  $\mathcal{L}_S := \mathcal{L}_\varphi$

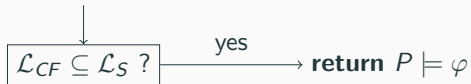


$\mathcal{L}_{CF} \subseteq \mathcal{L}_S ?$

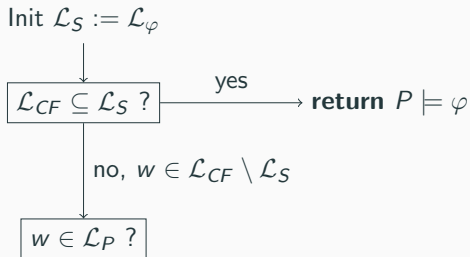


# Counterexample-guided abstraction refinement

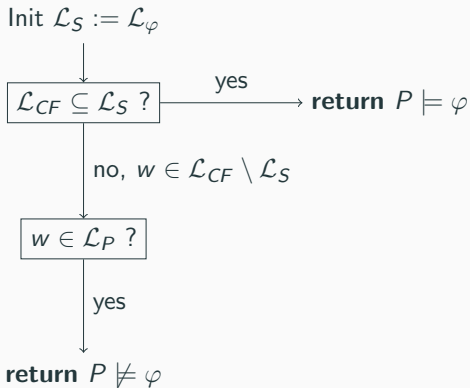
Init  $\mathcal{L}_S := \mathcal{L}_\varphi$



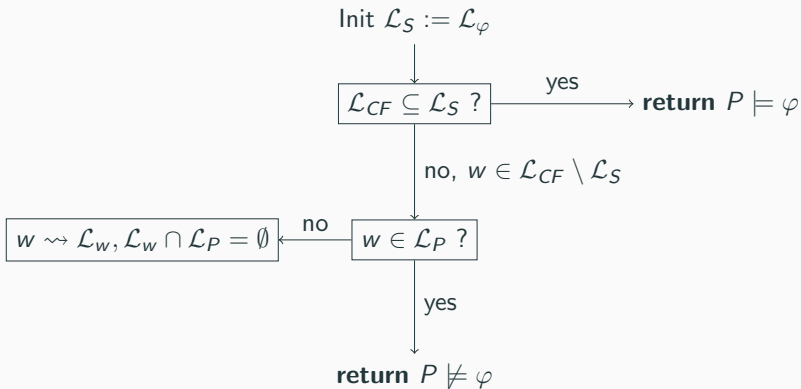
# Counterexample-guided abstraction refinement



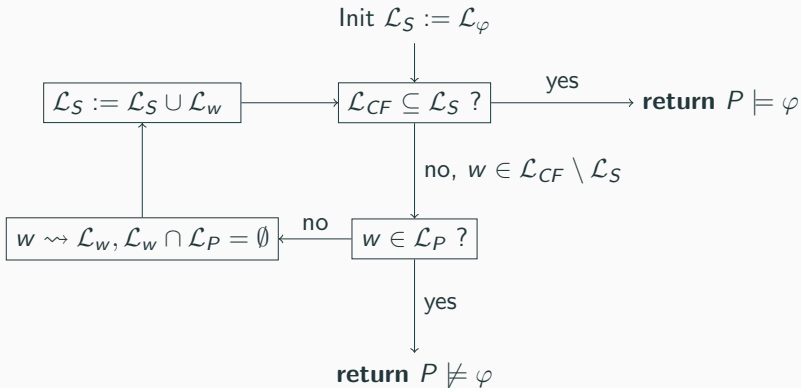
# Counterexample-guided abstraction refinement



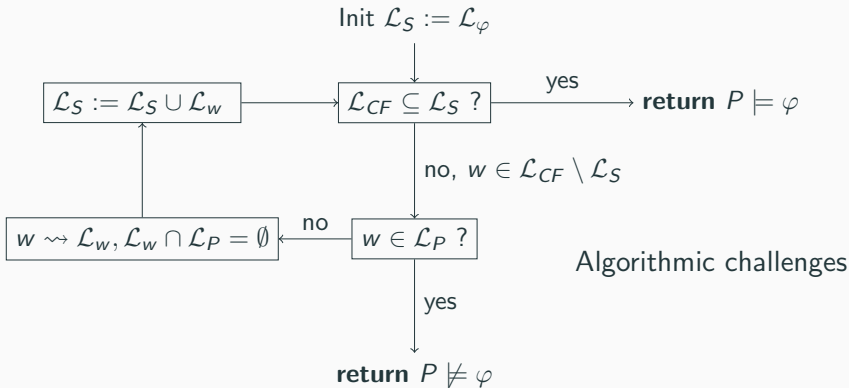
# Counterexample-guided abstraction refinement



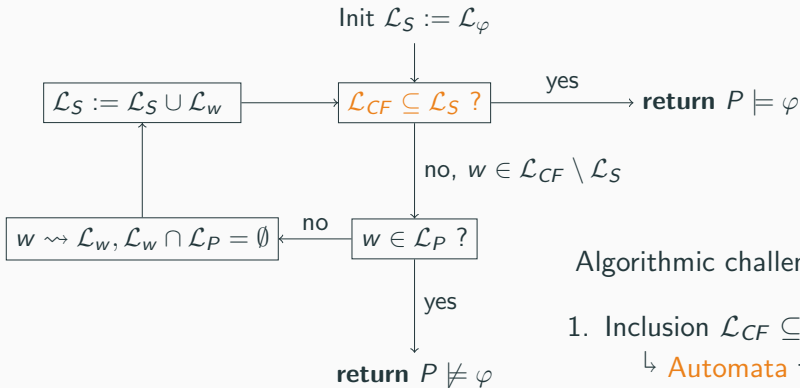
# Counterexample-guided abstraction refinement



# Counterexample-guided abstraction refinement



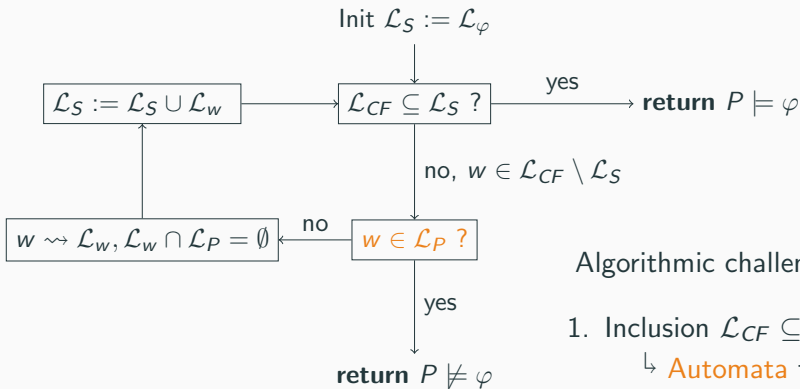
# Counterexample-guided abstraction refinement



Algorithmic challenges

1. Inclusion  $\mathcal{L}_{CF} \subseteq \mathcal{L}_S$   
↳ Automata theory

# Counterexample-guided abstraction refinement

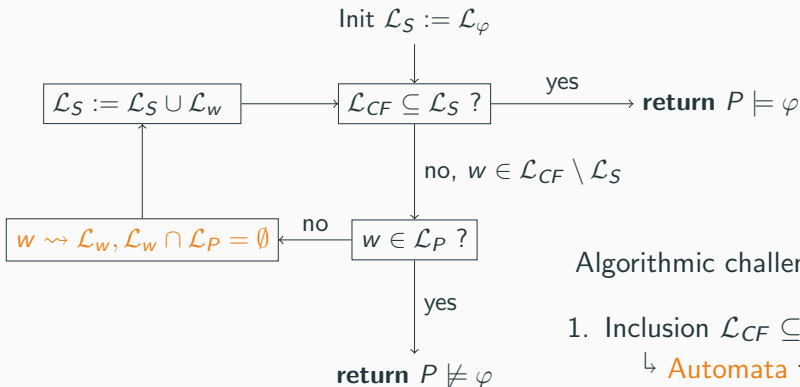


Algorithmic challenges

1. Inclusion  $\mathcal{L}_{CF} \subseteq \mathcal{L}_S$   
↳ Automata theory
2. Membership  $w \in \mathcal{L}_P$   
↳ Hoare logic



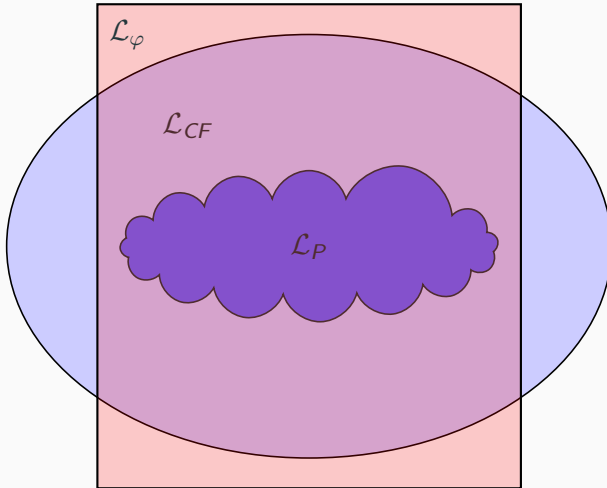
# Counterexample-guided abstraction refinement



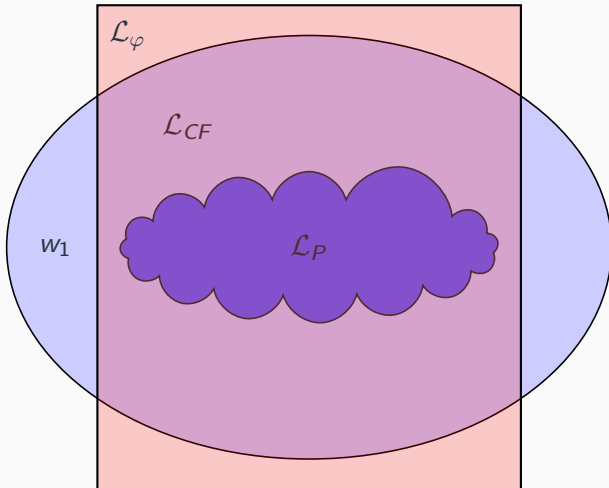
## Algorithmic challenges

1. Inclusion  $\mathcal{L}_{CF} \subseteq \mathcal{L}_S$   
↳ Automata theory
2. Membership  $w \in \mathcal{L}_P$   
↳ Hoare logic
3. Extrapolation  $w \rightsquigarrow \mathcal{L}_w$

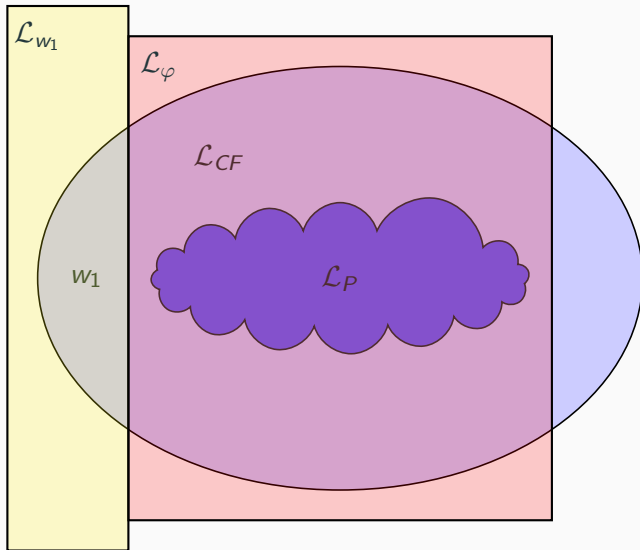
# Counterexample-guided abstraction refinement



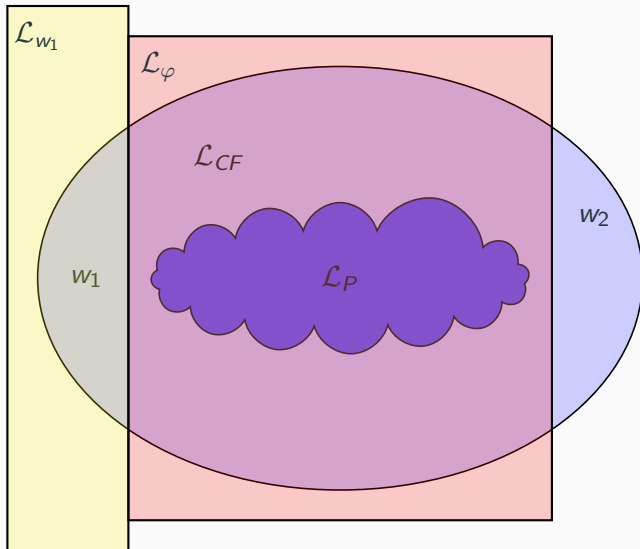
# Counterexample-guided abstraction refinement



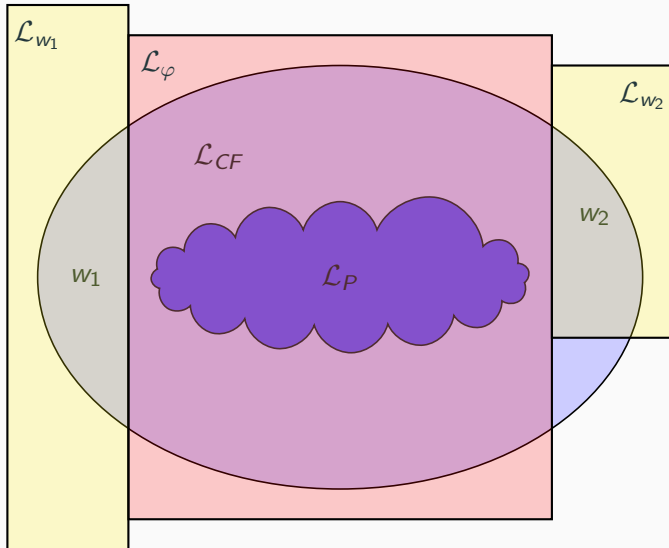
# Counterexample-guided abstraction refinement



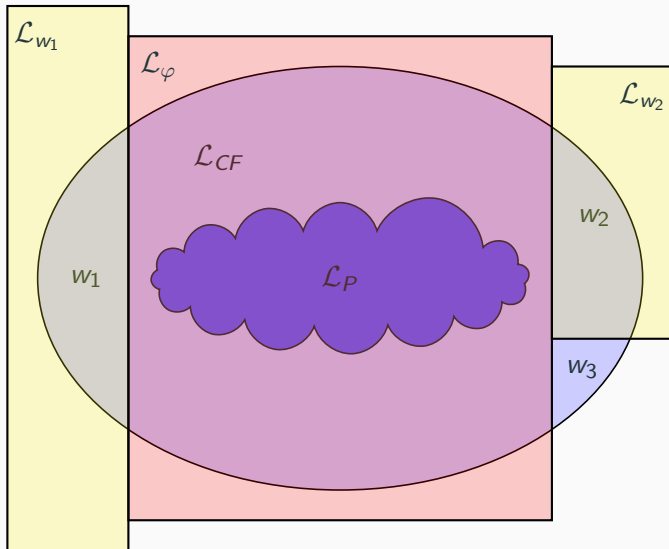
# Counterexample-guided abstraction refinement



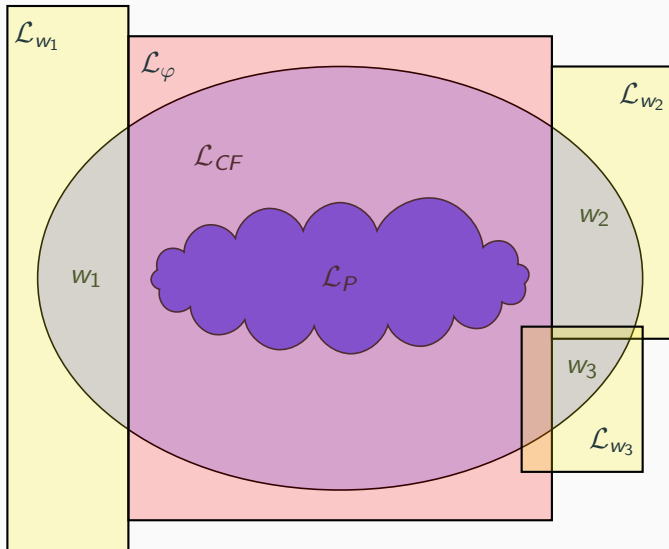
# Counterexample-guided abstraction refinement



# Counterexample-guided abstraction refinement



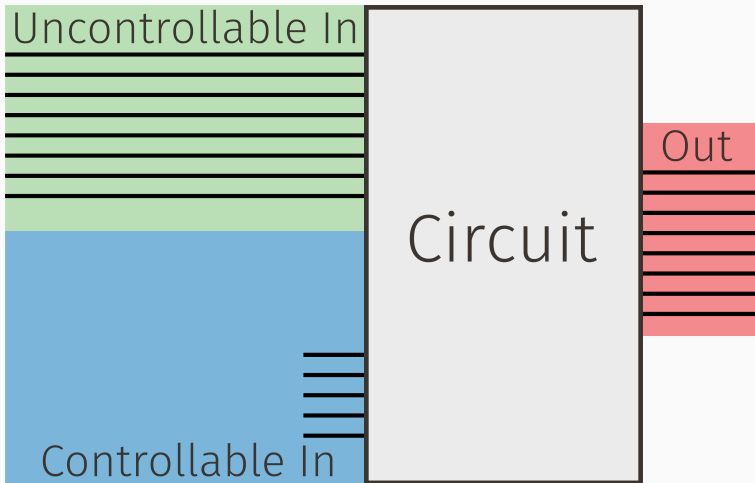
# Counterexample-guided abstraction refinement



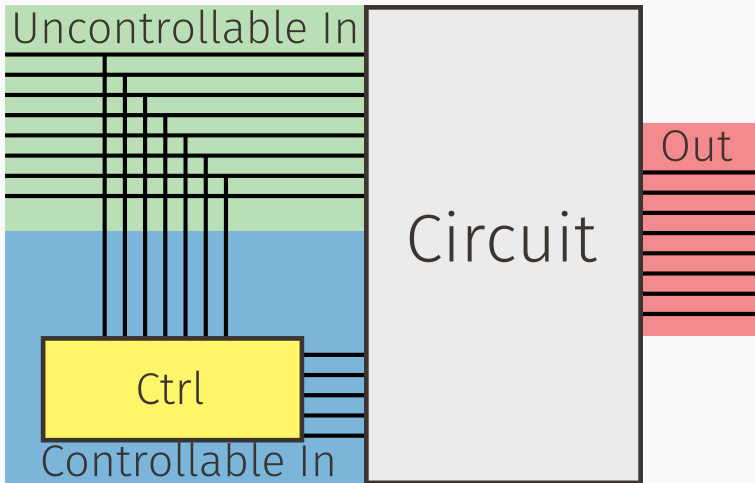


# Language-Theoretic Synthesis

---



# Synthesis



*Synthesis problem:*

Given: Program template  $T$  and specification  $\varphi$  .

Decide: Is there an instantiation  $T@i$  of  $T$  satisfying  $\varphi$  ?

*Synthesis problem:*

Given: Program template  $T$  and specification  $\varphi$  .

Decide: Is there an instantiation  $T@i$  of  $T$  satisfying  $\varphi$  ?

*Approach:*

Language-theoretic synthesis

CEGAR loop

# Language-theoretic synthesis

Model the control flow of a template as a grammar

Two types of non-determinism

# Language-theoretic synthesis

Model the control flow of a template as a grammar

Two types of non-determinism

Demonic / Uncontrollable  
non-determinism

```
proc F()  
  if (x == 0)  
    G()  
  else  
    H()
```

$$F \rightarrow \begin{array}{l} \text{read}(x,0)G \\ | \\ \text{read}(x,1)H \end{array}$$

# Language-theoretic synthesis

Model the control flow of a template as a grammar

## Two types of non-determinism

Demonic / Uncontrollable  
non-determinism

```
proc F()  
  if (x == 0)  
    G()  
  else  
    H()
```

$$F \rightarrow \begin{array}{l} \text{read}(x,0)G \\ | \\ \text{read}(x,1)H \end{array}$$

Angelic / Controllable  
non-determinism

```
proc F()  
  if ???  
    G()  
  else  
    H()
```

$$F \rightarrow \begin{array}{l} G \\ | \\ H \end{array}$$



# Language-theoretic synthesis

*Algorithmically:*

Model as a (context-free) two player perfect information game

# Language-theoretic synthesis

*Algorithmically:*

Model as a (context-free) **two player perfect information game**

Player  $\bigcirc$  represents uncontrollable non-determinism

# Language-theoretic synthesis

*Algorithmically:*

Model as a (context-free) **two player perfect information game**

Player  $\bigcirc$  represents uncontrollable non-determinism

Player  $\square$  represents controllable non-determinism

# Language-theoretic synthesis

*Algorithmically:*

Model as a (context-free) **two player perfect information game**

Player  $\bigcirc$  represents uncontrollable non-determinism

Player  $\square$  represents controllable non-determinism

Is there a **strategy**  $s$  for player  $\square$  to resolve the controllable non-determinism so that

$$\mathcal{L}(G@s) \subseteq \mathcal{L}(A) ?$$

# Language-theoretic synthesis

*Algorithmically:*

Model as a (context-free) **two player perfect information game**

Player  $\bigcirc$  represents uncontrollable non-determinism

Player  $\square$  represents controllable non-determinism

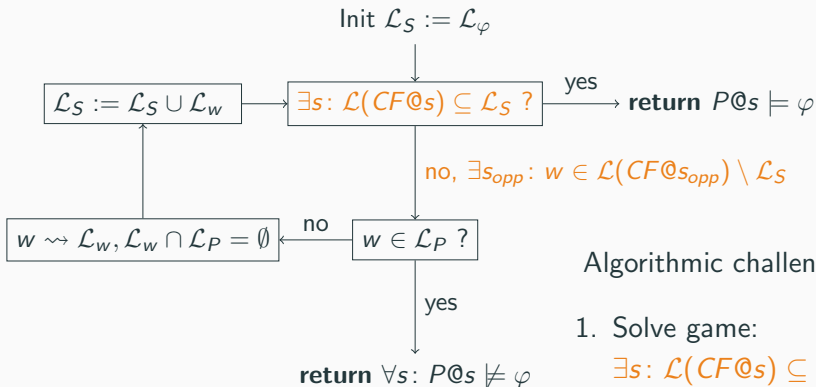
Is there a **strategy**  $s$  for player  $\square$  to resolve the controllable non-determinism so that

$$\mathcal{L}(G@s) \subseteq \mathcal{L}(A) ?$$

*From language-theoretic verification to synthesis:*

Replace the inclusion check  $\mathcal{L}(G) \subseteq \mathcal{L}(A)$  in the CEGAR loop by a strategy synthesis

# Language-theoretic synthesis



## Algorithmic challenges

1. Solve game:  
 $\exists s: \mathcal{L}(CF@s) \subseteq \mathcal{L}_S ?$
2. Membership  $w \in \mathcal{L}_P$
3. Extrapolation  $w \rightsquigarrow \mathcal{L}_w$

## Context-Free Games

---

## Context-free games - Input

*Input:*

Context-free grammar with ownership partitioning of the non-terminals

$$X_{\circ} \rightarrow aY \mid \varepsilon$$

$$Y_{\square} \rightarrow bX$$



## Context-free games - Input

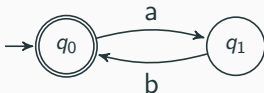
*Input:*

Context-free grammar with ownership partitioning of the non-terminals

$$X_{\circlearrowleft} \rightarrow aY \mid \varepsilon$$

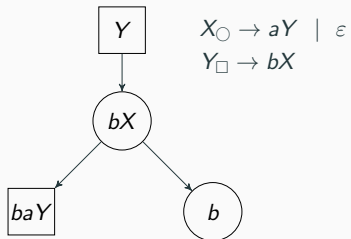
$$Y_{\square} \rightarrow bX$$

Finite automaton over terminals  $T_G$



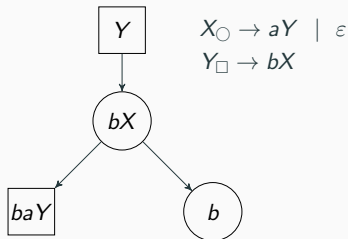
## Context-free games - Game arena

*Game arena:*



## Context-free games - Game arena

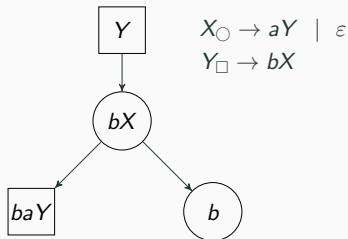
Game arena:



Vertices: **Sentential forms**  $\vartheta = (N_G \cup T_G)^*$

# Context-free games - Game arena

Game arena:

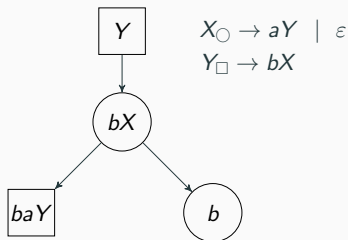


Vertices: **Sentential forms**  $\vartheta = (N_G \cup T_G)^*$

Arcs: **Left derivations**  $wX\gamma \Rightarrow_L w\eta\gamma$  if  $X \rightarrow \eta \in P_G$

## Context-free games - Game arena

Game arena:



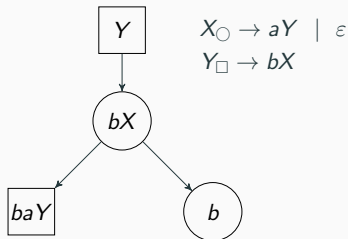
Vertices: **Sentential forms**  $\vartheta = (N_G \cup T_G)^*$

Arcs: **Left derivations**  $wX\gamma \Rightarrow_L w\eta\gamma$  if  $X \rightarrow \eta \in P_G$

Ownership: **Owner** of  $wX\gamma$  is the owner of  $X$

## Context-free games - Game arena

Game arena:



Vertices: **Sentential forms**  $\vartheta = (N_G \cup T_G)^*$

Arcs: **Left derivations**  $wX\gamma \Rightarrow_L w\eta\gamma$  if  $X \rightarrow \eta \in P_G$

Ownership: **Owner** of  $wX\gamma$  is the owner of  $X$

↳ Play = Sequence of left derivations

## Context-free games - Winning conditions

*Winning conditions:*

## Context-free games - Winning conditions

*Winning conditions:*

**Inclusion game:**

Derive a terminal word  $w \in \mathcal{L}(A)$



## Context-free games - Winning conditions

*Winning conditions:*

**Inclusion game:**

Derive a terminal word  $w \in \mathcal{L}(A)$

**Non-Inclusion game:**

Derive a terminal word  $w \notin \mathcal{L}(A)$

## Context-free games - Winning conditions

*Winning conditions:*

**Inclusion game:**

Derive a terminal word  $w \in \mathcal{L}(A)$  or infinite derivation

**Non-Inclusion game:**

Derive a terminal word  $w \notin \mathcal{L}(A)$  after finitely many steps

## Context-free games - Winning conditions

*Winning conditions:*

**Inclusion game:**

Derive a terminal word  $w \in \mathcal{L}(A)$  or infinite derivation

↳ Safety Game

**Non-Inclusion game:**

Derive a terminal word  $w \notin \mathcal{L}(A)$  after finitely many steps

↳ Reachability game

Here:

Consider inclusion game for player **prover**  $\square$

Consider non-inclusion game for player **refuter**  $\circ$

## Context-free games - Algorithms

State-of-the-art in **verification**:

# Context-free games - Algorithms

State-of-the-art in **verification**:

## *Saturation*

Compute state space of a pushdown

Stack content represented as a regular language

# Context-free games - Algorithms

State-of-the-art in **verification**:

## *Saturation*

Compute state space of a pushdown

Stack content represented as a regular language

## *Summarization*

Compute effect of function calls as input output relation

Stack content **not** represented

Used more often in SVComp

# Context-free games - Algorithms

State-of-the-art in **verification**:

## *Saturation*

Compute state space of a pushdown

Stack content represented as a regular language

## *Summarization*

Compute effect of function calls as input output relation

Stack content **not** represented

Used more often in SVComp

State-of-the-art in **synthesis**:

# Context-free games - Algorithms

State-of-the-art in **verification**:

## *Saturation*

Compute state space of a pushdown

Stack content represented as a regular language

## *Summarization*

Compute effect of function calls as input output relation

Stack content **not** represented

Used more often in SVComp

State-of-the-art in **synthesis**: **No summaries for games**



# Context-free games - Algorithms

State-of-the-art in **verification**:

## *Saturation*

Compute state space of a pushdown

Stack content represented as a regular language

## *Summarization*

Compute effect of function calls as input output relation

Stack content **not** represented

Used more often in SVComp

State-of-the-art in **synthesis**: **No summaries for games**

Problem \ Algorithm	Saturation	Summarization
Verification		
Synthesis		

# Context-free games - Algorithms

State-of-the-art in **verification**:

## *Saturation*

Compute state space of a pushdown

Stack content represented as a regular language

## *Summarization*

Compute effect of function calls as input output relation

Stack content **not** represented

Used more often in SVComp

State-of-the-art in **synthesis**: **No summaries for games**

Problem \ Algorithm	Saturation	Summarization
Verification		[SP78] [RHS95]
Synthesis		

# Context-free games - Algorithms

State-of-the-art in **verification**:

## *Saturation*

Compute state space of a pushdown

Stack content represented as a regular language

## *Summarization*

Compute effect of function calls as input output relation

Stack content **not** represented

Used more often in SVComp

State-of-the-art in **synthesis**: **No summaries for games**

Problem \ Algorithm	Saturation	Summarization
Verification	[BEM97] [FWW97]	[SP78] [RHS95]
Synthesis		

# Context-free games - Algorithms

State-of-the-art in **verification**:

## *Saturation*

Compute state space of a pushdown

Stack content represented as a regular language

## *Summarization*

Compute effect of function calls as input output relation

Stack content **not** represented

Used more often in SVComp

State-of-the-art in **synthesis**: **No summaries for games**

Problem \ Algorithm	Saturation	Summarization
Verification	[BEM97] [FWW97]	[SP78] [RHS95]
Synthesis	[C02] [MSS05] [HO09]	

# Context-free games - Algorithms

State-of-the-art in **verification**:

## *Saturation*

Compute state space of a pushdown

Stack content represented as a regular language

## *Summarization*

Compute effect of function calls as input output relation

Stack content **not** represented

Used more often in SVComp

State-of-the-art in **synthesis**: **No summaries for games**

Problem \ Algorithm	Saturation	Summarization
Verification	[BEM97] [FWW97]	[SP78] [RHS95]
Synthesis	[C02] [MSS05] [HO09]	???

# Context-free games - Algorithms

State-of-the-art in **verification**:

## *Saturation*

Compute state space of a pushdown

Stack content represented as a regular language

## *Summarization*

Compute effect of function calls as input output relation

Stack content **not** represented

Used more often in SVComp

State-of-the-art in **synthesis**: **No summaries for games**

Problem \ Algorithm	Saturation	Summarization
Verification	[BEM97] [FWW97]	[SP78] [RHS95]
Synthesis	[C02] [MSS05] [HO09]	??? <b>Next</b>

*How to decide which player wins the game?*

Fixed-point iteration over a suitable summary domain

Now:

1. Explain & define domain
2. Explain fixed-point iteration

# Formulas over the Transition Monoid

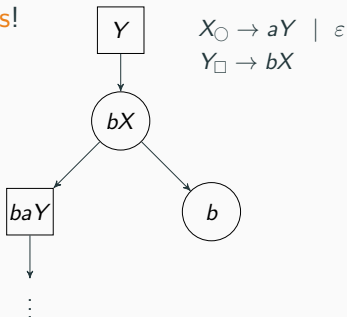
---



# The tree of plays

How to decide whether refuter can win from a given position?

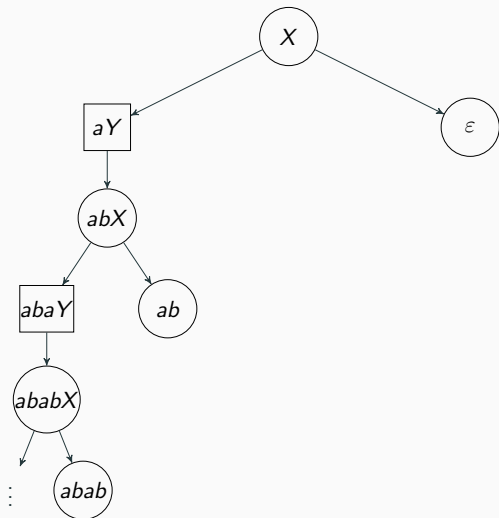
Consider the **tree of plays!**



Refuter wins non-inclusion in  $(ab)^*$  by picking  $X \rightarrow \epsilon$

$Y$  is a **winning position for refuter**  $\circ$

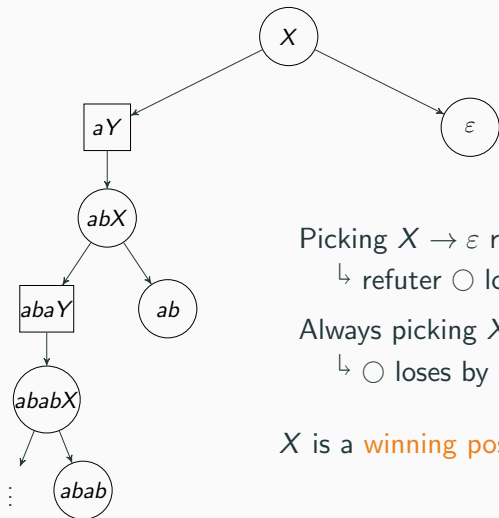
# The tree of plays - Example



$X_{\circ} \rightarrow aY \mid \varepsilon$

$Y_{\square} \rightarrow bX$

# The tree of plays - Example



$X_{\circlearrowleft} \rightarrow aY \mid \varepsilon$

$Y_{\square} \rightarrow bX$

Picking  $X \rightarrow \varepsilon$  results in word in  $(ab)^*$

↳ refuter  $\circlearrowleft$  loses non-inclusion

Always picking  $X \rightarrow aY$  results in infinite play

↳  $\circlearrowleft$  loses by definition

$X$  is a **winning position for prover**  $\square$

*Problem:*

Tree is usually infinite

*Problem:*

Tree is usually infinite

*Observation 1:*

Labels of inner nodes do not matter for inclusion

*Problem:*

Tree is usually infinite

*Observation 1:*

Labels of inner nodes do not matter for inclusion

Only ownership is important

*Problem:*

Tree is usually infinite

*Observation 1:*

Labels of inner nodes do not matter for inclusion

Only ownership is important

↪ Replace inner nodes of **refuter** by  $\vee$

*Problem:*

Tree is usually infinite

*Observation 1:*

Labels of inner nodes do not matter for inclusion

Only ownership is important

↪ Replace inner nodes of **refuter** by  $\vee$

↪ Replace inner nodes of **prover** by  $\wedge$



*Problem:*

Tree is usually infinite

*Observation 1:*

Labels of inner nodes do not matter for inclusion

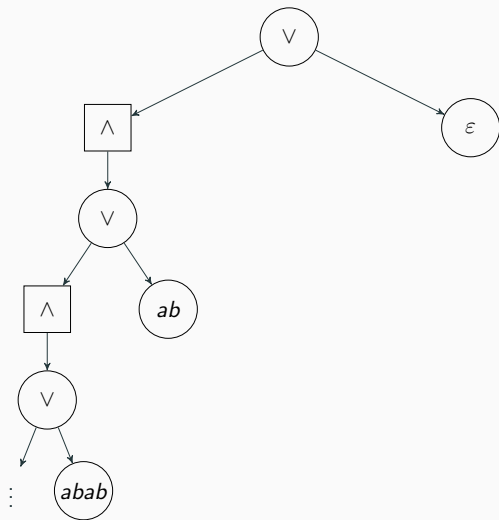
Only ownership is important

↪ Replace inner nodes of **refuter** by  $\vee$

↪ Replace inner nodes of **prover** by  $\wedge$

Understand tree as (infinite) **positive Boolean formula** over words

## Formulas - Example



*Remaining problems:*

1. Formulas are *still* infinite
2. Even the set of atomic propositions  $T_G^*$  is infinite

↳ Tackle 2. first

## Equivalence relation

*Observation 2:*

The words are not important — only the **state changes matter**

# Equivalence relation

*Observation 2:*

The words are not important — only the **state changes matter**

Define **equivalence relation**  $\sim_A$  such that words are equivalent iff they induce the same **state changes** on  $A$

# Equivalence relation

*Observation 2:*

The words are not important — only the **state changes matter**

Define **equivalence relation**  $\sim_A$  such that words are equivalent iff they induce the same **state changes** on  $A$

$$w \sim_A v$$

iff

# Equivalence relation

*Observation 2:*

The words are not important — only the **state changes matter**

Define **equivalence relation**  $\sim_A$  such that words are equivalent iff they induce the same **state changes** on  $A$

$$w \sim_A v \\ \text{iff } \forall q, q' \in Q :$$

# Equivalence relation

*Observation 2:*

The words are not important — only the **state changes matter**

Define **equivalence relation**  $\sim_A$  such that words are equivalent iff they induce the same **state changes** on  $A$

$$w \sim_A v \\ \text{iff } \forall q, q' \in Q : q \xrightarrow{w} q' \quad \text{iff} \quad q \xrightarrow{v} q'$$



# Equivalence relation

*Observation 2:*

The words are not important — only the **state changes matter**

Define **equivalence relation**  $\sim_A$  such that words are equivalent iff they induce the same **state changes** on  $A$

$$w \sim_A v \\ \text{iff } \forall q, q' \in Q : q \xrightarrow{w} q' \text{ iff } q \xrightarrow{v} q'$$

$M_A$  is the set of all equivalence classes  $[w]$  of  $\sim_A$

$T_G^*$  is partitioned into equivalence classes of  $\sim_A$

# Transition monoid

Represent equivalence classes by **boxes**:

$$\text{box}(w) = \left\{ (q, q') \in Q \times Q \mid q \xrightarrow{w} q' \right\} \in \mathcal{P}(Q \times Q)$$

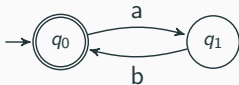
Represent equivalence classes by **boxes**:

$$\text{box}(w) = \left\{ (q, q') \in Q \times Q \mid q \xrightarrow{w} q' \right\} \in \mathcal{P}(Q \times Q)$$

Boxes correspond to **procedure summaries** for programs  
(in a precise sense)

## Transition monoid - Example

$$\text{box}(w) = \left\{ (q, q') \in Q \times Q \mid q \xrightarrow{w} q' \right\}$$



$id = [\varepsilon]$



$[a]$



$[b]$



$[ab]$



$[ba]$

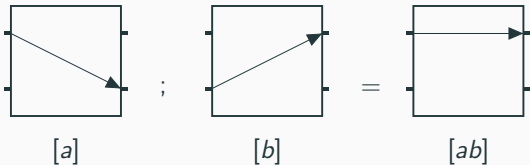


$[aa] = [bb]$

All other boxes represent empty equivalence classes

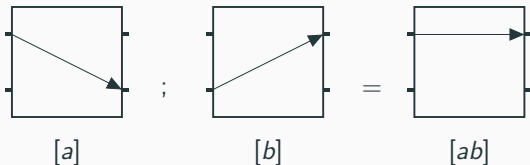
## Relational composition of boxes

Boxes can be composed using **relational composition** ;



# Relational composition of boxes

Boxes can be composed using **relational composition** ;

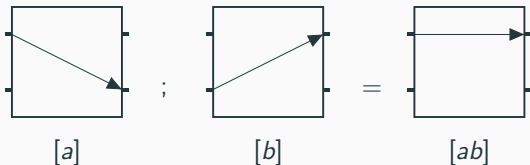


Monoids are **isomorphic**:

$$(M_A, \cdot, [\varepsilon]) \cong (\underbrace{\text{box}(T_G^*)}_{\subseteq \mathcal{P}(Q \times Q)}, ;, \text{box}(\varepsilon))$$

## Relational composition of boxes

Boxes can be composed using **relational composition** ;



Monoids are **isomorphic**:

$$(M_A, \cdot, [\varepsilon]) \cong (\underbrace{\text{box}(T_G^*)}_{\subseteq \mathcal{P}(Q \times Q)}, ;, \text{box}(\varepsilon))$$

↳ Up to  $|M_A| \leq 2^{|Q|^2}$  equivalence classes

Previously: (Infinite) positive Boolean formulas **over words**



## Back to games

Previously: (Infinite) positive Boolean formulas *over words*

Now: (Infinite) positive Boolean formulas *over  $M_A$*

## Back to games

Previously: (Infinite) positive Boolean formulas *over words*

Now: (Infinite) positive Boolean formulas *over  $M_A$*

Down to *finitely many atomic propositions*

## Back to games

Previously: (Infinite) positive Boolean formulas *over words*

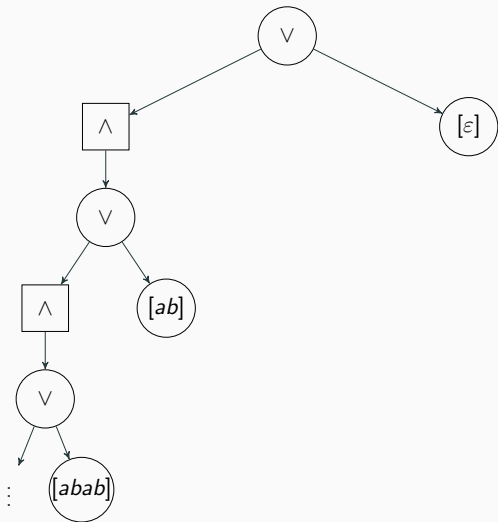
Now: (Infinite) positive Boolean formulas *over  $M_A$*

Down to *finitely many atomic propositions*

*Remaining problem:*

Formulas themselves are infinite

# Formulas - Example



## From infinite to finite formulas

*Observation 3:*

Every infinite formula over  $M_A$  is **logically equivalent** (under suitable evaluation semantics) to some **finite** formula

## From infinite to finite formulas

*Observation 3:*

Every infinite formula over  $M_A$  is **logically equivalent** (under suitable evaluation semantics) to some **finite** formula

Infinite formulas define functions  $F : 2^{M_A} \rightarrow \{0, 1\}$

## From infinite to finite formulas

*Observation 3:*

Every infinite formula over  $M_A$  is **logically equivalent** (under suitable evaluation semantics) to some **finite** formula

Infinite formulas define functions  $F : 2^{M_A} \rightarrow \{0, 1\}$

All such functions can be represented by finite formulas

## From infinite to finite formulas

*Observation 3:*

Every infinite formula over  $M_A$  is **logically equivalent** (under suitable evaluation semantics) to some **finite** formula

Infinite formulas define functions  $F : 2^{M_A} \rightarrow \{0, 1\}$

All such functions can be represented by finite formulas

Restrict to **finite positive Boolean formulas over  $M_A$**



## From infinite to finite formulas

*Observation 3:*

Every infinite formula over  $M_A$  is **logically equivalent** (under suitable evaluation semantics) to some **finite** formula

Infinite formulas define functions  $F : 2^{M_A} \rightarrow \{0, 1\}$

All such functions can be represented by finite formulas

Restrict to **finite positive Boolean formulas over  $M_A$**

*In the example:*

Infinite formula:  $[\varepsilon] \vee ([ab] \vee ([abab] \vee \dots))$

Note:  $[ab] = [abab] = [ababab] = \dots$

Finite formula:  $[\varepsilon] \vee [ab]$

## From infinite to finite formulas

*Observation 3:*

Every infinite formula over  $M_A$  is **logically equivalent** (under suitable evaluation semantics) to some **finite** formula

Infinite formulas define functions  $F : 2^{M_A} \rightarrow \{0, 1\}$

All such functions can be represented by finite formulas

Restrict to **finite positive Boolean formulas over  $M_A$**

*In the example:*

Infinite formula:  $[\varepsilon] \vee ([ab] \vee ([abab] \vee \dots))$

Note:  $[ab] = [abab] = [ababab] = \dots$

Finite formula:  $[\varepsilon] \vee [ab]$

How to compute these **finite formulas** in general?

# Fixed-Point Iteration

---

# Fixed point iteration

*Problem:*

How to compute the formulas?

*Fixed-point iteration:*

Translate the grammar into a **system of equations**

Solve using **Kleene iteration**

# Fixed point iteration

*Problem:*

How to compute the formulas?

*Fixed-point iteration:*

Translate the grammar into a **system of equations**

Solve using **Kleene iteration**

*Domain:*

Finite positive Boolean formulas over  $M_A$  (up to  $\Leftrightarrow$ )

Partial order: Implication  $\Rightarrow$

Least element: *false*

## Fixed-point iteration - Example

Grammar

$$X_{\circ} \rightarrow aY \quad | \quad \varepsilon$$

$$Y_{\square} \rightarrow bX$$

System of equations

$$F_X = [a]; F_Y \vee [\varepsilon]$$

$$F_Y = [b]; F_X$$

## Fixed-point iteration - Example

*Iteration:*

Nr.	$F_X$	$F_Y$
-----	-------	-------

Grammar

$$X_{\circ} \rightarrow aY \quad | \quad \varepsilon$$

$$Y_{\square} \rightarrow bX$$

System of equations

$$F_X = [a]; F_Y \vee [\varepsilon]$$

$$F_Y = [b]; F_X$$

# Fixed-point iteration - Example

Grammar

$$X_{\circ} \rightarrow aY \quad | \quad \varepsilon$$

$$Y_{\square} \rightarrow bX$$

System of equations

$$F_X = [a]; F_Y \vee [\varepsilon]$$

$$F_Y = [b]; F_X$$

*Iteration:*

Nr.	$F_X$	$F_Y$
0	<i>false</i>	<i>false</i>



## Fixed-point iteration - Example

Grammar

$$X_{\circ} \rightarrow aY \quad | \quad \varepsilon$$

$$Y_{\square} \rightarrow bX$$

System of equations

$$F_X = [a]; F_Y \vee [\varepsilon]$$

$$F_Y = [b]; F_X$$

*Iteration:*

Nr.	$F_X$	$F_Y$
0	<i>false</i>	<i>false</i>
1	$[\varepsilon]$	<i>false</i>

## Fixed-point iteration - Example

Grammar

$$X_{\circ} \rightarrow aY \quad | \quad \varepsilon$$

$$Y_{\square} \rightarrow bX$$

System of equations

$$F_X = [a]; F_Y \vee [\varepsilon]$$

$$F_Y = [b]; F_X$$

Iteration:

Nr.	$F_X$	$F_Y$
0	<i>false</i>	<i>false</i>
1	$[\varepsilon]$	<i>false</i>
2	$[\varepsilon]$	$[b] = [b]; [\varepsilon]$

# Fixed-point iteration - Example

Grammar

$$X_{\circ} \rightarrow aY \quad | \quad \varepsilon$$

$$Y_{\square} \rightarrow bX$$

System of equations

$$F_X = [a]; F_Y \vee [\varepsilon]$$

$$F_Y = [b]; F_X$$

Iteration:

Nr.	$F_X$	$F_Y$
0	<i>false</i>	<i>false</i>
1	$[\varepsilon]$	<i>false</i>
2	$[\varepsilon]$	$[b] = [b]; [\varepsilon]$
3	$[ab] \vee [\varepsilon]$	$[b]$

# Fixed-point iteration - Example

Grammar

$$X_{\circ} \rightarrow aY \quad | \quad \varepsilon$$

$$Y_{\square} \rightarrow bX$$

System of equations

$$F_X = [a]; F_Y \vee [\varepsilon]$$

$$F_Y = [b]; F_X$$

Iteration:

Nr.	$F_X$	$F_Y$
0	<i>false</i>	<i>false</i>
1	$[\varepsilon]$	<i>false</i>
2	$[\varepsilon]$	$[b] = [b]; [\varepsilon]$
3	$[ab] \vee [\varepsilon]$	$[b]$
4	$[ab] \vee [\varepsilon]$	$[b]; ([ab] \vee [\varepsilon])$

# Fixed-point iteration - Example

Grammar

$$X_{\circ} \rightarrow aY \quad | \quad \varepsilon$$

$$Y_{\square} \rightarrow bX$$

System of equations

$$F_X = [a]; F_Y \vee [\varepsilon]$$

$$F_Y = [b]; F_X$$

Iteration:

Nr.	$F_X$	$F_Y$
0	<i>false</i>	<i>false</i>
1	$[\varepsilon]$	<i>false</i>
2	$[\varepsilon]$	$[b] = [b]; [\varepsilon]$
3	$[ab] \vee [\varepsilon]$	$[b]$
4	$[ab] \vee [\varepsilon]$	$[b]; ([ab] \vee [\varepsilon])$ $= [bab] \vee [b]$

# Fixed-point iteration - Example

Grammar

$$X_{\circ} \rightarrow aY \quad | \quad \varepsilon$$

$$Y_{\square} \rightarrow bX$$

System of equations

$$F_X = [a]; F_Y \vee [\varepsilon]$$

$$F_Y = [b]; F_X$$

Iteration:

Nr.	$F_X$	$F_Y$
0	<i>false</i>	<i>false</i>
1	$[\varepsilon]$	<i>false</i>
2	$[\varepsilon]$	$[b] = [b]; [\varepsilon]$
3	$[ab] \vee [\varepsilon]$	$[b]$
4	$[ab] \vee [\varepsilon]$	$[b]; ([ab] \vee [\varepsilon])$ $= [bab] \vee [b]$ $\Leftrightarrow [b]$

## Winning Regions

---

## Rejecting

Define the evaluation  $\varphi$  so that



## Rejecting

Define the evaluation  $\varphi$  so that

$$\varphi([w]) = 1 \quad \text{iff} \quad w \notin \mathcal{L}(A)$$

## Rejecting

Define the evaluation  $\varphi$  so that

$$\varphi([w]) = 1 \quad \text{iff} \quad w \notin \mathcal{L}(A) \quad \text{iff} \quad [w] \subseteq \overline{\mathcal{L}(A)}$$

## Rejecting

Define the evaluation  $\varphi$  so that

$$\varphi([w]) = 1 \quad \text{iff} \quad w \notin \mathcal{L}(A) \quad \text{iff} \quad [w] \subseteq \overline{\mathcal{L}(A)}$$

by

$$\begin{aligned} \varphi : M_A &\rightarrow \{0, 1\} \\ [w] &\mapsto \begin{cases} 1 & (q_0, q_f) \notin \text{box}(w) \text{ for all } q_f \in Q_f \\ 0 & \text{else} \end{cases} \end{aligned}$$

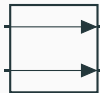
# Rejecting

Define the evaluation  $\varphi$  so that

$$\varphi([w]) = 1 \quad \text{iff} \quad w \notin \mathcal{L}(A) \quad \text{iff} \quad [w] \subseteq \overline{\mathcal{L}(A)}$$

by

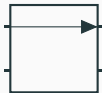
$$\varphi: M_A \rightarrow \{0, 1\}$$
$$[w] \mapsto \begin{cases} 1 & (q_0, q_f) \notin \text{box}(w) \text{ for all } q_f \in Q_f \\ 0 & \text{else} \end{cases}$$



$$\varphi([\varepsilon]) = 0$$



$$\varphi([b]) = 1$$



$$\varphi([ab]) = 0$$

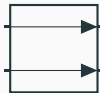
# Rejecting

Define the evaluation  $\varphi$  so that

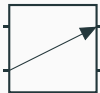
$$\varphi([w]) = 1 \quad \text{iff} \quad w \notin \mathcal{L}(A) \quad \text{iff} \quad [w] \subseteq \overline{\mathcal{L}(A)}$$

by

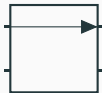
$$\begin{aligned} \varphi : M_A &\rightarrow \{0, 1\} \\ [w] &\mapsto \begin{cases} 1 & (q_0, q_f) \notin \text{box}(w) \text{ for all } q_f \in Q_f \\ 0 & \text{else} \end{cases} \end{aligned}$$



$$\varphi([\varepsilon]) = 0$$



$$\varphi([b]) = 1$$



$$\varphi([ab]) = 0$$

Sentential form  $\alpha \in \mathcal{V}$  is called **rejecting** if  $\varphi(F_\alpha) = 1$

# Winning region of prover

## Theorem

The set of **non-rejecting positions**

$$W^{\subseteq} = \{\alpha \in \vartheta \mid \varphi(F_{\alpha}) = 0\}$$

is the **winning region of prover**  $\square$  for the inclusion game.

# Winning region of prover

## Theorem

The set of **non-rejecting positions**

$$W^{\subseteq} = \{\alpha \in \vartheta \mid \varphi(F_{\alpha}) = 0\}$$

is the **winning region of prover**  $\square$  for the inclusion game.

## Proof

Position  $w \in \overline{\mathcal{L}(A)}$  has formula  $F_w = [w]$  with  $\varphi([w]) = 1$

# Winning region of prover

## Theorem

The set of **non-rejecting positions**

$$W^{\subseteq} = \{\alpha \in \vartheta \mid \varphi(F_{\alpha}) = 0\}$$

is the **winning region of prover**  $\square$  for the inclusion game.

## Proof

Position  $w \in \overline{\mathcal{L}(A)}$  has formula  $F_w = [w]$  with  $\varphi([w]) = 1$

$$\Rightarrow \overline{\mathcal{L}(A)} \cap W^{\subseteq} = \emptyset$$



# Winning region of prover

## Theorem

The set of **non-rejecting positions**

$$W^{\subseteq} = \{\alpha \in \vartheta \mid \varphi(F_{\alpha}) = 0\}$$

is the **winning region of prover**  $\square$  for the inclusion game.

## Proof

Position  $w \in \overline{\mathcal{L}(A)}$  has formula  $F_w = [w]$  with  $\varphi([w]) = 1$

$$\Rightarrow \overline{\mathcal{L}(A)} \cap W^{\subseteq} = \emptyset$$

**Show:** If the current position is non-rejecting and it is the turn of

# Winning region of prover

## Theorem

The set of **non-rejecting positions**

$$W^{\subseteq} = \{\alpha \in \vartheta \mid \varphi(F_{\alpha}) = 0\}$$

is the **winning region of prover**  $\square$  for the inclusion game.

## Proof

Position  $w \in \overline{\mathcal{L}(A)}$  has formula  $F_w = [w]$  with  $\varphi([w]) = 1$

$$\Rightarrow \overline{\mathcal{L}(A)} \cap W^{\subseteq} = \emptyset$$

**Show:** If the current position is non-rejecting and it is the turn of

(1) Prover: **There is** a move to a non-rejecting position,

# Winning region of prover

## Theorem

The set of **non-rejecting positions**

$$W^{\subseteq} = \{\alpha \in \vartheta \mid \varphi(F_{\alpha}) = 0\}$$

is the **winning region of prover**  $\square$  for the inclusion game.

## Proof

Position  $w \in \overline{\mathcal{L}(A)}$  has formula  $F_w = [w]$  with  $\varphi([w]) = 1$

$$\Rightarrow \overline{\mathcal{L}(A)} \cap W^{\subseteq} = \emptyset$$

**Show:** If the current position is non-rejecting and it is the turn of

- (1) Prover: **There is** a move to a non-rejecting position,
- (2) Refuter: **All** moves go to non-rejecting positions.

# Winning region of prover

## Theorem

The set of **non-rejecting positions**

$$W^{\subseteq} = \{\alpha \in \vartheta \mid \varphi(F_{\alpha}) = 0\}$$

is the **winning region of prover**  $\square$  for the inclusion game.

## Proof

Position  $w \in \overline{\mathcal{L}(A)}$  has formula  $F_w = [w]$  with  $\varphi([w]) = 1$

$$\Rightarrow \overline{\mathcal{L}(A)} \cap W^{\subseteq} = \emptyset$$

**Show:** If the current position is non-rejecting and it is the turn of

- (1) Prover: **There is** a move to a non-rejecting position,
- (2) Refuter: **All** moves go to non-rejecting positions.

Since the inclusion game is a **safety game**, staying in  $W^{\subseteq}$  suffices.

# Winning region of prover

## Theorem

The set of **non-rejecting positions**

$$W^{\subseteq} = \{\alpha \in \vartheta \mid \varphi(F_{\alpha}) = 0\}$$

is the **winning region of prover**  $\square$  for the inclusion game.

In the example, starting from  $X$ :

Both  $[ab]$ ,  $[\varepsilon]$  contain  $(q_0, q_0)$

# Winning region of prover

## Theorem

The set of **non-rejecting positions**

$$W^{\subseteq} = \{\alpha \in \vartheta \mid \varphi(F_{\alpha}) = 0\}$$

is the **winning region of prover**  $\square$  for the inclusion game.

In the example, starting from  $X$ :

Both  $[ab]$ ,  $[\varepsilon]$  contain  $(q_0, q_0)$

$$\hookrightarrow \varphi([ab]) = 0, \varphi([\varepsilon]) = 0$$

# Winning region of prover

## Theorem

The set of **non-rejecting positions**

$$W^{\subseteq} = \{\alpha \in \vartheta \mid \varphi(F_{\alpha}) = 0\}$$

is the **winning region of prover**  $\square$  for the inclusion game.

In the example, starting from  $X$ :

Both  $[ab]$ ,  $[\varepsilon]$  contain  $(q_0, q_0)$

$$\hookrightarrow \varphi([ab]) = 0, \varphi([\varepsilon]) = 0$$

$$\hookrightarrow \varphi(F_X) = \varphi([ab] \vee [\varepsilon]) = 0$$

# Winning region of prover

## Theorem

The set of **non-rejecting positions**

$$W^{\subseteq} = \{\alpha \in \vartheta \mid \varphi(F_{\alpha}) = 0\}$$

is the **winning region of prover**  $\square$  for the inclusion game.

In the example, starting from  $X$ :

Both  $[ab]$ ,  $[\varepsilon]$  contain  $(q_0, q_0)$

$\hookrightarrow \varphi([ab]) = 0, \varphi([\varepsilon]) = 0$

$\hookrightarrow \varphi(F_X) = \varphi([ab] \vee [\varepsilon]) = 0$

$\hookrightarrow X$  is non-rejecting



# Winning region of prover

## Theorem

The set of **non-rejecting positions**

$$W^{\subseteq} = \{\alpha \in \vartheta \mid \varphi(F_{\alpha}) = 0\}$$

is the **winning region of prover**  $\square$  for the inclusion game.

In the example, starting from  $X$ :

Both  $[ab]$ ,  $[\varepsilon]$  contain  $(q_0, q_0)$

$\hookrightarrow \varphi([ab]) = 0, \varphi([\varepsilon]) = 0$

$\hookrightarrow \varphi(F_X) = \varphi([ab] \vee [\varepsilon]) = 0$

$\hookrightarrow X$  is non-rejecting

Indeed, prover wins inclusion from  $X$

# Winning region of refuter

## Theorem

The set of **rejecting positions**

$$W^{\neq} = \{\alpha \in \vartheta \mid \varphi(F_\alpha) = 1\}$$

is the **winning region of refuter**  $\bigcirc$  for the non-inclusion game.

# Winning region of refuter

## Theorem

The set of **rejecting positions**

$$W^{\neq} = \{\alpha \in \mathcal{V} \mid \varphi(F_\alpha) = 1\}$$

is the **winning region of refuter**  $\bigcirc$  for the non-inclusion game.

## Proof

Position  $w \in \mathcal{L}(A)$  has formula  $F_w = [w]$  with  $\varphi([w]) = 0$

# Winning region of refuter

## Theorem

The set of **rejecting positions**

$$W^{\mathcal{L}} = \{\alpha \in \mathcal{V} \mid \varphi(F_\alpha) = 1\}$$

is the **winning region of refuter**  $\bigcirc$  for the non-inclusion game.

## Proof

Position  $w \in \mathcal{L}(A)$  has formula  $F_w = [w]$  with  $\varphi([w]) = 0$

$$\Rightarrow \mathcal{L}(A) \cap W^{\mathcal{L}} = \emptyset$$

# Winning region of refuter

## Theorem

The set of **rejecting positions**

$$W^{\mathcal{L}} = \{\alpha \in \mathcal{V} \mid \varphi(F_\alpha) = 1\}$$

is the **winning region of refuter**  $\bigcirc$  for the non-inclusion game.

## Proof

Position  $w \in \mathcal{L}(A)$  has formula  $F_w = [w]$  with  $\varphi([w]) = 0$

$$\Rightarrow \mathcal{L}(A) \cap W^{\mathcal{L}} = \emptyset$$

**Show:** If the current position is rejecting and it is the turn of

# Winning region of refuter

## Theorem

The set of **rejecting positions**

$$W^{\mathcal{L}} = \{\alpha \in \mathcal{V} \mid \varphi(F_\alpha) = 1\}$$

is the **winning region of refuter**  $\bigcirc$  for the non-inclusion game.

## Proof

Position  $w \in \mathcal{L}(A)$  has formula  $F_w = [w]$  with  $\varphi([w]) = 0$

$$\Rightarrow \mathcal{L}(A) \cap W^{\mathcal{L}} = \emptyset$$

**Show:** If the current position is rejecting and it is the turn of

(1) Refuter: **There is** a move to a rejecting position,

# Winning region of refuter

## Theorem

The set of **rejecting positions**

$$W^{\mathcal{L}} = \{\alpha \in \mathcal{V} \mid \varphi(F_\alpha) = 1\}$$

is the **winning region of refuter**  $\bigcirc$  for the non-inclusion game.

## Proof

Position  $w \in \mathcal{L}(A)$  has formula  $F_w = [w]$  with  $\varphi([w]) = 0$

$$\Rightarrow \mathcal{L}(A) \cap W^{\mathcal{L}} = \emptyset$$

**Show:** If the current position is rejecting and it is the turn of

- (1) Refuter: **There is** a move to a rejecting position,
- (2) Prover: **All** moves go to rejecting positions.

# Winning region of refuter

## Theorem

The set of **rejecting positions**

$$W^{\mathcal{L}} = \{\alpha \in \mathcal{V} \mid \varphi(F_\alpha) = 1\}$$

is the **winning region of refuter**  $\bigcirc$  for the non-inclusion game.

## Proof

Position  $w \in \mathcal{L}(A)$  has formula  $F_w = [w]$  with  $\varphi([w]) = 0$

$$\Rightarrow \mathcal{L}(A) \cap W^{\mathcal{L}} = \emptyset$$

**Show:** If the current position is rejecting and it is the turn of

- (1) Refuter: **There is** a move to a rejecting position,
- (2) Prover: **All** moves go to rejecting positions.

Not sufficient to win **reachability game**, need to minimize distance to  $\overline{\mathcal{L}(A)}$  in every step.



# Winning region of refuter

## Theorem

The set of **rejecting positions**

$$W^{\neq} = \{\alpha \in \vartheta \mid \varphi(F_\alpha) = 1\}$$

is the **winning region of refuter**  $\bigcirc$  for the non-inclusion game.

In the example, starting from  $Y$ :

$[b]$  does not contain  $(q_0, q_0)$

# Winning region of refuter

## Theorem

The set of **rejecting positions**

$$W^{\neq} = \{\alpha \in \vartheta \mid \varphi(F_\alpha) = 1\}$$

is the **winning region of refuter**  $\bigcirc$  for the non-inclusion game.

In the example, starting from  $Y$ :

$[b]$  does not contain  $(q_0, q_0)$

$$\hookrightarrow \varphi(F_Y) = \varphi([b]) = 1$$

# Winning region of refuter

## Theorem

The set of **rejecting positions**

$$W^{\neq} = \{\alpha \in \vartheta \mid \varphi(F_\alpha) = 1\}$$

is the **winning region of refuter**  $\bigcirc$  for the non-inclusion game.

In the example, starting from  $Y$ :

$[b]$  does not contain  $(q_0, q_0)$

↳  $\varphi(F_Y) = \varphi([b]) = 1$

↳  $Y$  is rejecting

# Winning region of refuter

## Theorem

The set of **rejecting positions**

$$W^{\neq} = \{\alpha \in \vartheta \mid \varphi(F_\alpha) = 1\}$$

is the **winning region of refuter**  $\bigcirc$  for the non-inclusion game.

In the example, starting from  $Y$ :

$[b]$  does not contain  $(q_0, q_0)$

↳  $\varphi(F_Y) = \varphi([b]) = 1$

↳  $Y$  is rejecting

Indeed, refuter wins non-inclusion from  $Y$

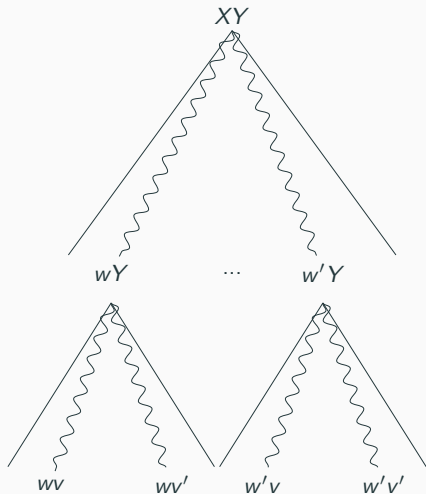
# Composition

---

How to define the **composition operator** ; that replaces concatenation . in the system of equations?

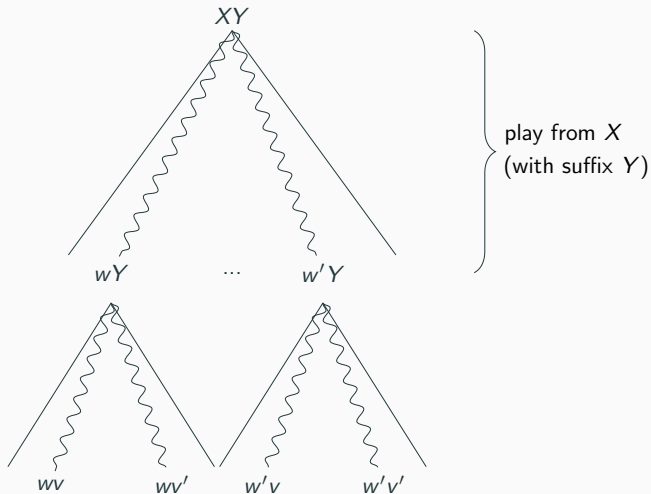
# Composition

*Plays from  $XY$  decompose:*



# Composition

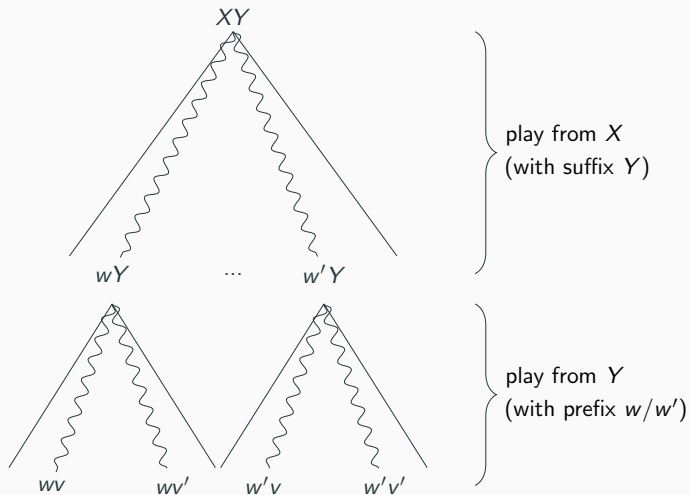
*Plays from  $XY$  decompose:*



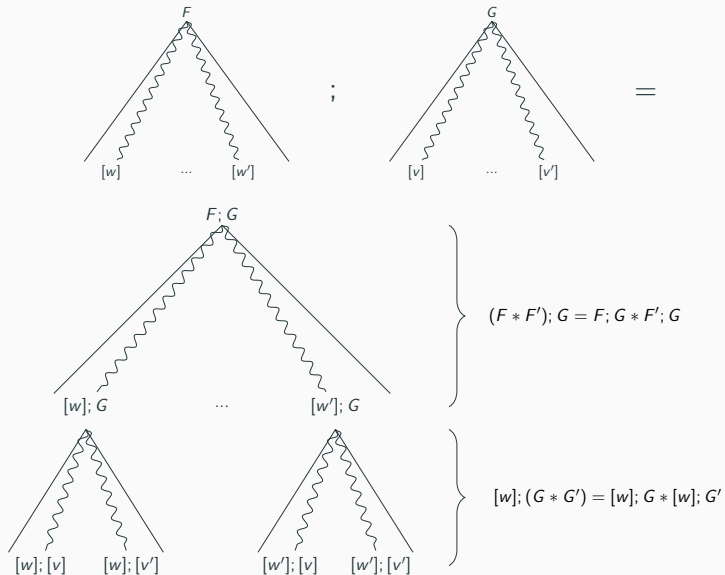


# Composition

*Plays from  $XY$  decompose:*



# Composition



# Complexity & Performance

---

# Algorithm

*Given:* Game  $G$ ,  $A$  and initial position  $\alpha$

*Algorithm for solving non-inclusion:*

# Algorithm

*Given:* Game  $G$ ,  $A$  and initial position  $\alpha$

*Algorithm for solving non-inclusion:*

(1) Set  $F_X = \text{false}$  for all  $X \in N$

# Algorithm

*Given:* Game  $G$ ,  $A$  and initial position  $\alpha$

*Algorithm for solving non-inclusion:*

(1) Set  $F_X = \text{false}$  for all  $X \in N$

(2) Do until  $F_X^{\text{old}} \Leftrightarrow F_X^{\text{new}}$  for all  $X \in N$ :

$$F = \text{rhs}(F)$$

# Algorithm

*Given:* Game  $G$ ,  $A$  and initial position  $\alpha$

*Algorithm for solving non-inclusion:*

(1) Set  $F_X = \text{false}$  for all  $X \in N$

(2) Do until  $F_X^{old} \Leftrightarrow F_X^{new}$  for all  $X \in N$ :

$$F = \text{rhs}(F)$$

(3) Compute  $F_\alpha$ , and return *true* iff  $\varphi(F_\alpha) = 1$

# Algorithm

*Given:* Game  $G, A$  and initial position  $\alpha$

*Algorithm for solving non-inclusion:*

(1) Set  $F_X = \text{false}$  for all  $X \in N$

(2) Do until  $F_X^{\text{old}} \Leftrightarrow F_X^{\text{new}}$  for all  $X \in N$ :

$$F = \text{rhs}(F)$$

(3) Compute  $F_\alpha$ , and return *true* iff  $\varphi(F_\alpha) = 1$

Compose solutions  $F_X$  for non-terminals to obtain the solutions for  
**all sentential forms**  $\alpha = \alpha_1 \dots \alpha_k \in \vartheta$ :  $F_\alpha = F_{\alpha_1}; \dots; F_{\alpha_k}$



# Algorithm

Given: Game  $G, A$  and initial position  $\alpha$

Algorithm for solving non-inclusion:

(1) Set  $F_X = false$  for all  $X \in N$

(2) Do until  $F_X^{old} \Leftrightarrow F_X^{new}$  for all  $X \in N$ :

$$F = rhs(F)$$

(3) Compute  $F_\alpha$ , and return *true* iff  $\varphi(F_\alpha) = 1$

Compose solutions  $F_X$  for non-terminals to obtain the solutions for  
all sentential forms  $\alpha = \alpha_1 \dots \alpha_k \in \vartheta$ :  $F_\alpha = F_{\alpha_1}; \dots; F_{\alpha_k}$

Solve system *once* and decide game for *any* position  $\alpha$

## Theorem

1. *Deciding non-inclusion games is 2EXPTIME-complete.*

## Theorem

1. *Deciding non-inclusion games is 2EXPTIME-complete.*
2. *The algorithm solves non-inclusion games in*

$$\mathcal{O}\left(|G|^2 \cdot 2^{2^{|Q|^{c_1}}} + |\alpha| \cdot 2^{2^{|Q|^{c_2}}}\right)$$

where  $c_1, c_2 \in \mathbb{N}$  are constants.

## Theorem

1. *Deciding non-inclusion games is 2EXPTIME-complete.*
2. *The algorithm solves non-inclusion games in*

$$\mathcal{O}\left(|G|^2 \cdot 2^{2^{|Q|^{c_1}}} + |\alpha| \cdot 2^{2^{|Q|^{c_2}}}\right)$$

where  $c_1, c_2 \in \mathbb{N}$  are constants.

3. *Hardness by reduction from acceptance in **alternating** Turing machines with exponential space.*

## Related Work

*Cachat [C02]:*

## Related Work

*Cachat [C02]:*

Consider **pushdown system** with ownership partitioning of control states

## Related Work

*Cachat [C02]:*

Consider **pushdown system** with ownership partitioning of control states

Can one player enforce a configuration such that the **stack content** is accepted by an **alternating finite automaton (AFA)**?

## Related Work

*Cachat [C02]:*

Consider **pushdown system** with ownership partitioning of control states

Can one player enforce a configuration such that the **stack content** is accepted by an **alternating finite automaton (AFA)**?

Solve by **saturating** the transitions of the AFA



## Related Work

*Cachat [C02]:*

Consider **pushdown system** with ownership partitioning of control states

Can one player enforce a configuration such that the **stack content** is accepted by an **alternating finite automaton (AFA)**?

Solve by **saturating** the transitions of the AFA

Saturated AFA accepts the **winning region**

## Related Work

*Cachat [C02]:*

Consider **pushdown system** with ownership partitioning of control states

Can one player enforce a configuration such that the **stack content** is accepted by an **alternating finite automaton (AFA)**?

Solve by **saturating** the transitions of the AFA

Saturated AFA accepts the **winning region**

**EXPTIME**

## Related Work

*Cachat [C02]:*

Consider **pushdown system** with ownership partitioning of control states

Can one player enforce a configuration such that the **stack content** is accepted by an **alternating finite automaton (AFA)**?

Solve by **saturating** the transitions of the AFA

Saturated AFA accepts the **winning region**

**EXPTIME**

↳ Our game **can be reduced** to Cachat

## Related Work

*Walukiewicz [W96/01]:*

## Related Work

*Walukiewicz [W96/01]:*

Consider **pushdown system** with ownership partitioning and priorities of control states

## Related Work

*Walukiewicz [W96/01]:*

Consider **pushdown system** with ownership partitioning and priorities of control states

Pushdown **parity game**

## Related Work

*Walukiewicz [W96/01]:*

Consider **pushdown system** with ownership partitioning and priorities of control states

Pushdown **parity game**

Reduce to a parity game on a **finite graph**

## Related Work

*Walukiewicz [W96/01]:*

Consider **pushdown system** with ownership partitioning and priorities of control states

Pushdown **parity game**

Reduce to a parity game on a **finite graph**

On push, one player guesses the **effect** of the push



## Related Work

*Walukiewicz [W96/01]:*

Consider **pushdown system** with ownership partitioning and priorities of control states

Pushdown **parity game**

Reduce to a parity game on a **finite graph**

On push, one player guesses the **effect** of the push

Other player decides to **verify** the guess or **skip** it

## Related Work

*Walukiewicz [W96/01]:*

Consider **pushdown system** with ownership partitioning and priorities of control states

Pushdown **parity game**

Reduce to a parity game on a **finite graph**

On push, one player guesses the **effect** of the push

Other player decides to **verify** the guess or **skip** it

**EXPTIME**

## Related Work

*Walukiewicz [W96/01]:*

Consider **pushdown system** with ownership partitioning and priorities of control states

Pushdown **parity game**

Reduce to a parity game on a **finite graph**

On push, one player guesses the **effect** of the push

Other player decides to **verify** the guess or **skip** it

**EXPTIME**

↳ Similar technique **can be applied** to our problem

## Related Work

*Muscholl, Schwentick, Segoufin [MSS05]:*

## Related Work

*Muscholl, Schwentick, Segoufin [MSS05]:*

Consider **context-free grammar**

## Related Work

*Muscholl, Schwentick, Segoufin [MSS05]:*

Consider **context-free grammar**

One player picks position that should be replaced

Other player picks rule

## Related Work

*Muscholl, Schwentick, Segoufin [MSS05]:*

Consider **context-free grammar**

One player picks position that should be replaced

Other player picks rule

Can one player enforce a **sentential form in a regular language**  
over  $N_G \cup T_G$ ?

## Related Work

*Muscholl, Schwentick, Segoufin [MSS05]:*

Consider **context-free grammar**

One player picks position that should be replaced

Other player picks rule

Can one player enforce a **sentential form in a regular language**  
over  $N_G \cup T_G$ ?

**Undecidable**



## Related Work

*Muscholl, Schwentick, Segoufin [MSS05]:*

Consider **context-free grammar**

One player picks position that should be replaced

Other player picks rule

Can one player enforce a **sentential form in a regular language**  
over  $N_G \cup T_G$ ?

**Undecidable**

**2EXPTIME** for **left-to-right strategies**

*Muscholl, Schwentick, Segoufin [MSS05]:*

Consider **context-free grammar**

One player picks position that should be replaced

Other player picks rule

Can one player enforce a **sentential form in a regular language**  
over  $N_G \cup T_G$ ?

**Undecidable**

**2EXPTIME** for **left-to-right strategies**

Similar to our game

*Muscholl, Schwentick, Segoufin [MSS05]:*

Consider **context-free grammar**

One player picks position that should be replaced

Other player picks rule

Can one player enforce a **sentential form in a regular language**  
over  $N_G \cup T_G$ ?

**Undecidable**

**2EXPTIME** for **left-to-right strategies**

Similar to our game

Hardness proof **carries over**

# Performance

*Comparison of 2EXPTIME algorithms:*

Input		Computation	
<b>Our algorithm</b>			
System of equations	P	Fixed-point iteration	2EXP
<b>Reduction to Cachat [C02]</b>			
Determinized automaton	EXP	Saturation	EXP
<b>Idea of Walukiewicz [W96/01]</b>			
Finite reachability game	2EXP	Saturation	P

guaranteed blow-up

may be lucky

We have implemented and compared:

- Our algorithm with naive Kleene iteration

- Our algorithm with worklist-based Kleene iteration

- Reduction to Cachat's pushdown games

*Problems with Cachat's algorithm:*

- Automaton  $A$  needs to be determinized

  - ↳ Guaranteed blow-up

- Algorithmic tricks for Cachat (worklist, ...) not suitable for the instances generated by the reduction

# Performance

Q / N / T	naive Kleene		worklist Kleene		Cachat	
	avg. time	% timeout	avg. time	% timeout	avg. time	% timeout
5/ 5/ 5	65.2	2	0.8	0	94.7	0
5/ 5/10	5.4	4	7.4	0	701.7	0
5/10/ 5	13.9	0	0.3	0	375.7	0
5/ 5/15	6.0	0	1.1	0	1618.6	0
5/10/10	32.0	2	122.1	0	2214.4	0
5/15/ 5	44.5	0	0.2	0	620.7	0
5/ 5/20	3.4	0	1.4	0	3434.6	4
5/10/15	217.7	0	7.4	0	5263.0	16
10/ 5/ 5	8.8	2	0.6	0	2737.8	2
10/ 5/10	9.0	6	69.8	0	6484.9	66
15/ 5/ 5	30.7	0	0.2	0	5442.4	52
10/10/ 5	9.7	0	0.2	0	7702.1	92
10/15/15	252.3	0	1.9	0	n/a	100
10/15/20	12.9	0	1.8	0	n/a	100

Experiments executed on i7-6700K, 4GHz, times in milliseconds, timeout 10 seconds

## **Future Work**

---

Liveness synthesis (infinite words)



Liveness synthesis (infinite words)

Synthesis for systems with branching behavior (trees)

Liveness synthesis (infinite words)

Synthesis for systems with branching behavior (trees)

Games on higher-order systems

## Future work

Liveness synthesis (infinite words)

Synthesis for systems with branching behavior (trees)

Games on higher-order systems

Solver technology for systems of equations (Newton iteration)

## Future work

Liveness synthesis (infinite words)

Synthesis for systems with branching behavior (trees)

Games on higher-order systems

Solver technology for systems of equations (Newton iteration)

Applications in hardware synthesis

**Thank you!**

**Questions?**

# References

---

[BEM97]: M. Heizmann, J. Hoenicke, and A. Podelski.  
*Nested interpolants*. POPL 2010



[BEM97]: A. Bouajjani, J. Esparza, and O. Maler.  
*Reachability analysis of pushdown automata:  
Application to model-checking.* CONCUR 1997

[FWW97]: A. Finkel, B. Willems, and P. Wolper.  
*A direct symbolic approach to model checking  
pushdown systems.* ENTCS 1997

[C02]: Th. Cachat.

*Symbolic strategy synthesis for games on pushdown graphs.* ICALP 2002

[HO09]: M. Hague and C.-H. L. Ong.

*Winning regions of pushdown parity games: A saturation method.* CONCUR 2009

[SP78]: M. Sharir and A. Pnueli.

*Two approaches to interprocedural data flow analysis.*

1978

[RHS95]: T. Reps, S. Horwitz, and M. Sagiv.

*Precise interprocedural dataflow analysis via graph reachability.* POPL 1995

[MSS05]: A. Muscholl, T. Schwentick, and L. Segoufin.  
*Active context-free games*. 2002

[W96/01]: I. Walukiewicz.  
*Pushdown processes: Games and model-checking*.  
1996 / IC 2001