Summaries for Context-Free Games

Lukáš Holík[1], Roland Meyer[2], and **Sebastian Muskalla**[2]

December 15, FSTTCS 2016, Chennai

1 Brno University of Technology, `holik@fit.vutbr.cz`
2 TU Braunschweig, {`roland.meyer, s.muskalla`}`@tu-braunschweig.de`

Motivation

Verification

Verification of context-free systems:

Verification

Verification of context-free systems:

Saturation

Compute state space of a pushdown

Stack content represented as a regular language

Verification

Verification of context-free systems:

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

Synthesis

Synthesis:

Synthesis

Synthesis:

Two types of non-determinism:

- controllable non-determinism
- uncontrollable non-determinism

Synthesis

Synthesis:

Two types of non-determinism:

controllable non-determinism

uncontrollable non-determinism

↳ Model as a 2-player game

Synthesis

Synthesis:

Two types of non-determinism:

controllable non-determinism

uncontrollable non-determinism

↳ Model as a 2-player game

To solve it: Lift the techniques for verification

Synthesis

Synthesis:

Two types of non-determinism:

controllable non-determinism

uncontrollable non-determinism

↳ Model as a 2-player game

To solve it: Lift the techniques for verification

State-of-the-art:

Problem \ Algorithm	Saturation	Summarization
Verification		
Synthesis		

Synthesis

Synthesis:

Two types of non-determinism:

controllable non-determinism

uncontrollable non-determinism

↳ Model as a 2-player game

To solve it: Lift the techniques for verification

State-of-the-art:

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

Synthesis

Synthesis:

Two types of non-determinism:

controllable non-determinism

uncontrollable non-determinism

↳ Model as a 2-player game

To solve it: Lift the techniques for verification

State-of-the-art:

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

Synthesis

Synthesis:

Two types of non-determinism:

controllable non-determinism

uncontrollable non-determinism

↳ Model as a 2-player game

To solve it: Lift the techniques for verification

State-of-the-art:

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

Synthesis

Synthesis:

Two types of non-determinism:

controllable non-determinism

uncontrollable non-determinism

↳ Model as a 2-player game

To solve it: Lift the techniques for verification

State-of-the-art:

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

Synthesis

Synthesis:

Two types of non-determinism:

controllable non-determinism

uncontrollable non-determinism

↳ Model as a 2-player game

To solve it: Lift the techniques for verification

State-of-the-art:

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

Context-Free Games

Context-free games - Input

Input:

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

$$\begin{aligned} X_{\bigcirc} &\rightarrow \quad aY \quad \mid \quad \varepsilon \\ Y_{\square} &\rightarrow \quad bX \end{aligned}$$

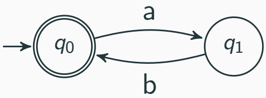
Context-free games - Input

Input:

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

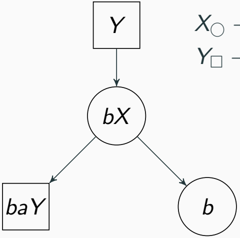
$$\begin{aligned} X_{\bigcirc} &\rightarrow \quad aY \quad | \quad \varepsilon \\ Y_{\square} &\rightarrow \quad bX \end{aligned}$$

Finite automaton over terminals T_G



Context-free games - Game arena

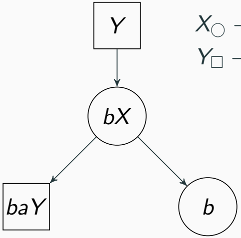
Game arena:



$$X_{\bigcirc} \to aY \mid \varepsilon$$
$$Y_{\Box} \to bX$$

Context-free games - Game arena

Game arena:

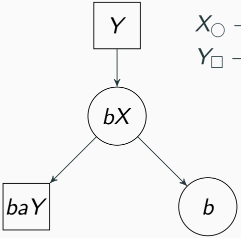


$$X_{\bigcirc} \to aY \mid \varepsilon$$
$$Y_{\Box} \to bX$$

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

Context-free games - Game arena

Game arena:



$X_{\bigcirc} \to aY \mid \varepsilon$

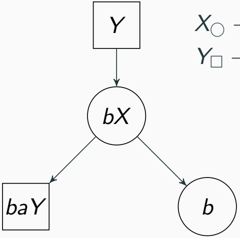
$Y_{\Box} \to bX$

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

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

Context-free games - Game arena

Game arena:



$X_{\bigcirc} \to aY \mid \varepsilon$
$Y_{\Box} \to bX$

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

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

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

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

$\hookrightarrow$ Safety Game

Non-Inclusion game:

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

$\hookrightarrow$ Reachability game

Here:

Consider inclusion game for player prover $\Box$

Consider non-inclusion game for player refuter $\bigcirc$

Summaries for context-free games

How to decide which player wins the game?

Fixed-point iteration over a suitable summary domain

Now:

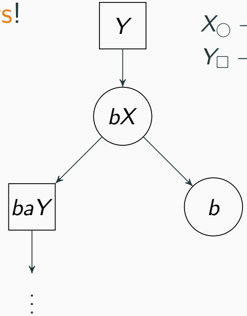
1. Explain & define domain
2. Explain fixed-point iteration

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!

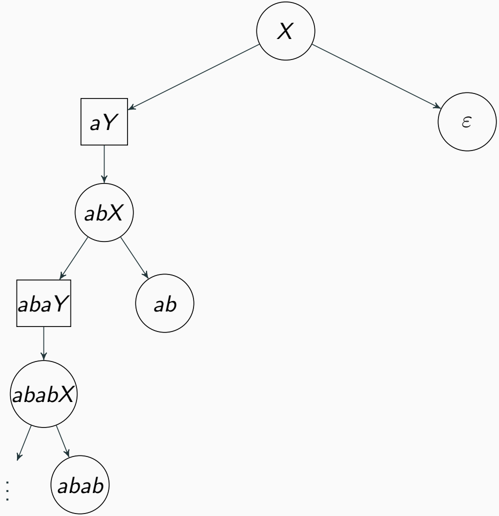


$$X_{\bigcirc} \to aY \mid \varepsilon$$
$$Y_{\square} \to bX$$

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

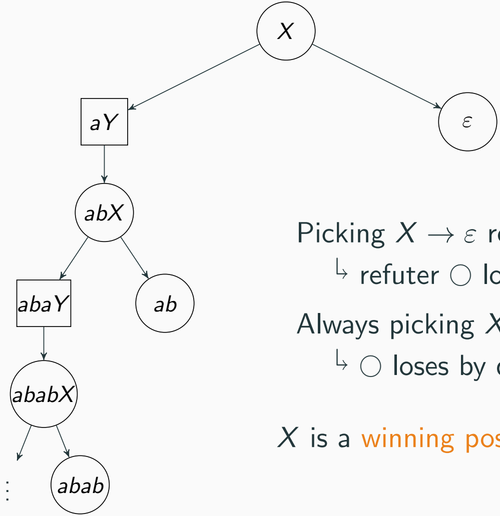
Y is a winning position for refuter $\bigcirc$

The tree of plays - Example



$$X_{\bigcirc} \to aY \;\mid\; \varepsilon$$
$$Y_{\square} \to bX$$

The tree of plays - Example



$$X_{\bigcirc} \to aY \mid \varepsilon$$
$$Y_{\square} \to bX$$

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

↳ refuter $\bigcirc$ loses non-inclusion

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

↳ $\bigcirc$ loses by definition

X is a winning position for prover $\square$

Formulas

Problem:

Tree is usually infinite

Formulas

Problem:

Tree is usually infinite

Observation 1:

Labels of inner nodes do not matter for inclusion

Formulas

Problem:

Tree is usually infinite

Observation 1:

Labels of inner nodes do not matter for inclusion

Only ownership is important

Formulas

Problem:

Tree is usually infinite

Observation 1:

Labels of inner nodes do not matter for inclusion

Only ownership is important

$\rightsquigarrow$ Replace inner nodes of refuter by $\vee$

Formulas

Problem:

Tree is usually infinite

Observation 1:

Labels of inner nodes do not matter for inclusion

Only ownership is important

$\rightsquigarrow$ Replace inner nodes of refuter by $\vee$

$\rightsquigarrow$ Replace inner nodes of prover by $\wedge$

Formulas

Problem:

Tree is usually infinite

Observation 1:

Labels of inner nodes do not matter for inclusion

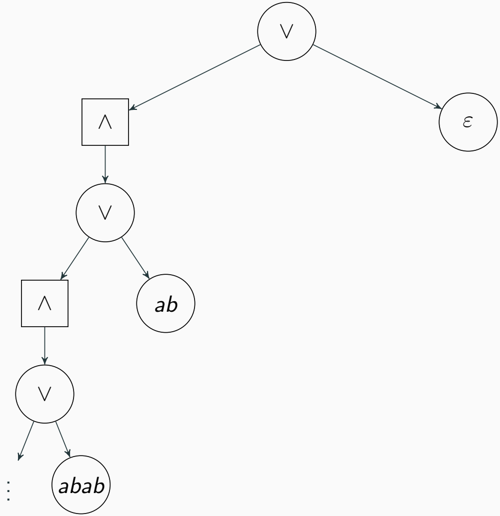
Only ownership is important

$\rightsquigarrow$ Replace inner nodes of refuter by $\vee$

$\rightsquigarrow$ Replace inner nodes of prover by $\wedge$

Understand tree as (infinite) positive Boolean formula over words

Formulas - Example



Formulas

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

$$
\begin{aligned}
& w \sim_A v \\
\text{iff} \quad & \forall q, q' \in Q :
\end{aligned}
$$

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

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

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

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

Transition monoid 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:

$$\mathrm{box}(w) = \left\{ (q, q') \in Q \times Q \;\middle|\; q \xrightarrow{w} q' \right\} \in \mathcal{P}(Q \times Q)$$

Transition monoid

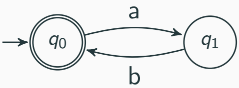
Represent equivalence classes by boxes:

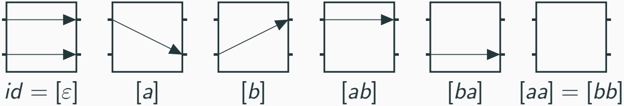
$$\mathrm{box}(w) = \left\{ (q, q') \in Q \times Q \;\middle|\; q \xrightarrow{w} q' \right\} \in \mathcal{P}(Q \times Q)$$

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

Transition monoid - Example

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

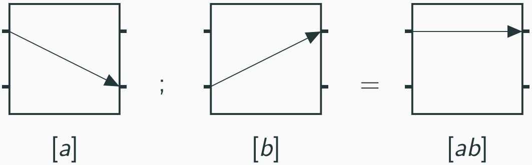




All other boxes represent empty equivalence classes

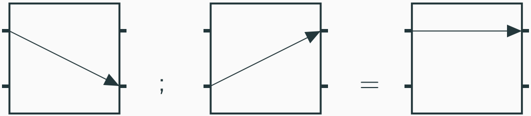
Relational composition of boxes

Boxes can be composed using relational composition ;



Relational composition of boxes

Boxes can be composed using relational composition ;



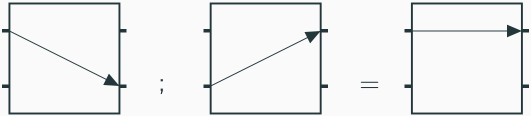
$[a]$ $[b]$ $[ab]$

Monoids are isomorphic:

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

Relational composition of boxes

Boxes can be composed using relational composition ;



$[a]$ $[b]$ $[ab]$

Monoids are isomorphic:

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

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

Back to games

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

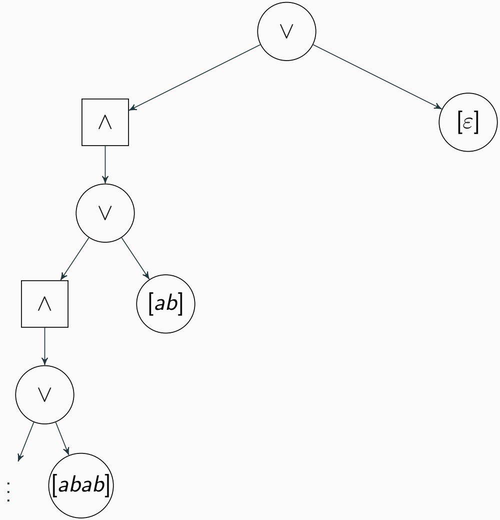
Now: (Infinite) positive Boolean formulas over M_A

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 $F : 2^{M_A} \to \{0,1\}$

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 $F : 2^{M_A} \to \{0, 1\}$
- 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 $F : 2^{M_A} \to \{0, 1\}$

All such functions can be represented by finite formulas

Restrict to finite positive Boolean formulas over M_A

In the example:

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

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

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 $F : 2^{M_A} \to \{0, 1\}$
All such functions can be represented by finite formulas

Restrict to finite positive Boolean formulas over M_A

In the example:

Infinite formula: $[\varepsilon] \lor \big([ab] \lor ([abab] \lor \ldots)\big)$
Note: $[ab] = [abab] = [ababab] = \ldots$
Finite formula: $[\varepsilon] \lor [ab]$

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

$$
\begin{aligned}
X_{\bigcirc} \to \quad & aY \qquad \mid \quad \varepsilon \\
Y_{\Box} \to \quad & bX
\end{aligned}
$$

System of equations

$$
\begin{aligned}
F_X = \quad & [a]; F_Y \quad \vee \quad [\varepsilon] \\
F_Y = \quad & [b]; F_X
\end{aligned}
$$

Fixed-point iteration - Example

Iteration:

Nr.	F_X	F_Y

Grammar

$$
\begin{array}{lll}
X_{\bigcirc} \rightarrow & aY & \mid \quad \varepsilon \\
Y_{\square} \rightarrow & bX &
\end{array}
$$

System of equations

$$
\begin{array}{lll}
F_X = & [a]; F_Y & \vee \quad [\varepsilon] \\
F_Y = & [b]; F_X &
\end{array}
$$

Fixed-point iteration - Example

Grammar

$$
\begin{aligned}
X_{\bigcirc} \rightarrow &\quad aY \qquad \mid \quad \varepsilon \\
Y_{\Box} \rightarrow &\quad bX
\end{aligned}
$$

System of equations

$$
\begin{aligned}
F_X = &\quad [a]; F_Y \quad \vee \quad [\varepsilon] \\
F_Y = &\quad [b]; F_X
\end{aligned}
$$

Iteration:

Nr.	F_X	F_Y
0	*false*	*false*

Fixed-point iteration - Example

Iteration:

Nr.	F_X	F_Y
0	*false*	*false*
1	$[\varepsilon]$	*false*

Grammar

$$
\begin{aligned}
X_{\bigcirc} &\rightarrow \quad aY \qquad \mid \quad \varepsilon \\
Y_{\square} &\rightarrow \quad bX
\end{aligned}
$$

System of equations

$$
\begin{aligned}
F_X &= \quad [a]; F_Y \quad \vee \quad [\varepsilon] \\
F_Y &= \quad [b]; F_X
\end{aligned}
$$

Fixed-point iteration - Example

Grammar

$$X_{\bigcirc} \rightarrow \quad aY \qquad | \quad \varepsilon$$
$$Y_{\square} \rightarrow \quad bX$$

System of equations

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

Iteration:

Nr.	F_X	F_Y
0	*false*	*false*
1	$[\varepsilon]$	*false*
2	$[\varepsilon]$	$[b] = [b]; [\varepsilon]$

Fixed-point iteration - Example

Grammar

$$X_{\bigcirc} \rightarrow \quad aY \qquad | \quad \varepsilon$$
$$Y_{\Box} \rightarrow \quad bX$$

System of equations

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

Iteration:

Nr.	F_X	F_Y
0	*false*	*false*
1	$[\varepsilon]$	*false*
2	$[\varepsilon]$	$[b] = [b]; [\varepsilon]$
3	$[ab] \vee [\varepsilon]$	$[b]$

Fixed-point iteration - Example

Grammar

$$X_{\bigcirc} \to \quad aY \qquad | \quad \varepsilon$$
$$Y_{\square} \to \quad bX$$

System of equations

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

Iteration:

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

Fixed-point iteration - Example

Grammar

$X_{\bigcirc} \rightarrow \quad aY \qquad | \quad \varepsilon$

$Y_{\Box} \rightarrow \quad bX$

System of equations

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

$F_Y = \quad [b]; F_X$

Iteration:

Nr.	F_X	F_Y
0	*false*	*false*
1	$[\varepsilon]$	*false*
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

$$X_{\bigcirc} \rightarrow \quad aY \qquad | \quad \varepsilon$$
$$Y_{\Box} \rightarrow \quad bX$$

System of equations

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

Iteration:

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

Fixed point iteration

Theorem

For every sentential form:

The (finite) formula obtained from LFP is logically equivalent to the (infinite) formula obtained from the tree of plays.

Winning Regions

Rejecting

Define the evaluation φ so that

Rejecting

Define the evaluation φ so that

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

Rejecting

Define the evaluation φ 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 φ 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

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

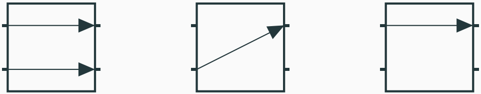
Rejecting

Define the evaluation φ 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

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



$\varphi([\varepsilon]) = 0$ $\varphi([b]) = 1$ $\varphi([ab]) = 0$

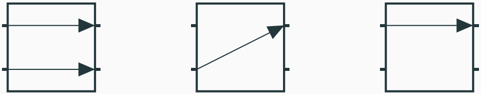
Rejecting

Define the evaluation φ 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

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



$\varphi([\varepsilon]) = 0$ $\quad\quad$ $\varphi([b]) = 1$ $\quad\quad$ $\varphi([ab]) = 0$

Sentential form $\alpha \in \vartheta$ 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 $\Box$ 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 $\Box$ 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 $\Box$ for the inclusion game.

In the example, starting from X:

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

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

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)

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

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

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)

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

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

↳ 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 $\Box$ for the inclusion game.

In the example, starting from X:

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

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

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

↳ X is non-rejecting

Indeed, prover wins inclusion from X

Winning region of refuter

Theorem

The set of rejecting positions

$$W^{\not\subseteq} \;=\; \{\alpha \in \vartheta \mid \varphi(F_\alpha) = 1\}$$

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

Winning region of refuter

Theorem

The set of rejecting positions

$$W^{\not\subseteq} \; = \; \{\alpha \in \vartheta \mid \varphi(F_\alpha) = 1\}$$

is the winning region of refuter ○ 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

$$W^{\not\subseteq} \;=\; \{\alpha \in \vartheta \mid \varphi(F_\alpha) = 1\}$$

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)

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

Winning region of refuter

Theorem

The set of rejecting positions

$$W^{\not\subseteq} \;=\; \{\alpha \in \vartheta \mid \varphi(F_\alpha) = 1\}$$

is the winning region of refuter ○ 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

$$W^{\not\subseteq} \;=\; \{\alpha \in \vartheta \mid \varphi(F_\alpha) = 1\}$$

is the winning region of refuter ○ 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

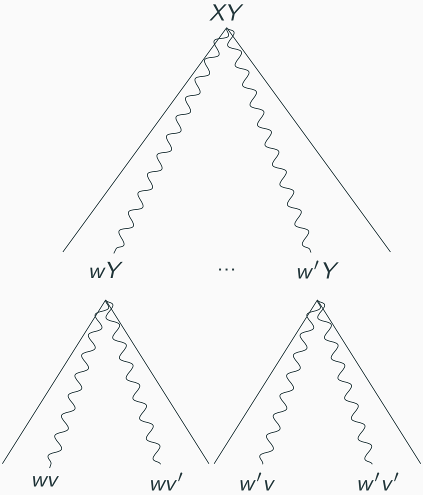
Composition

Composition

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

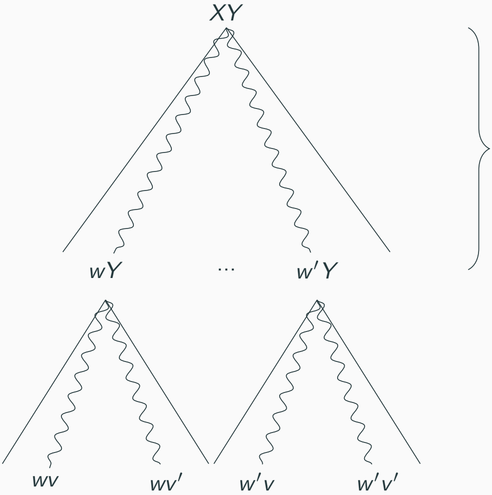
Composition

Plays from XY decompose:



Composition

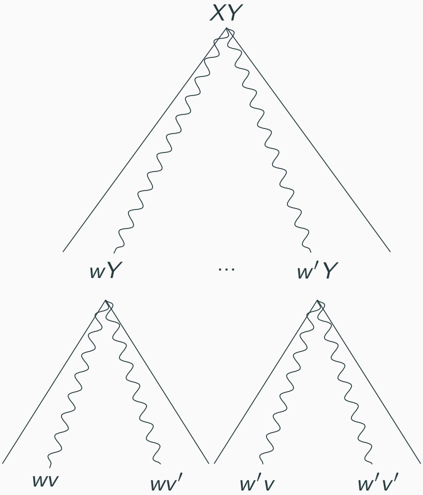
Plays from XY decompose:



play from X
(with suffix Y)

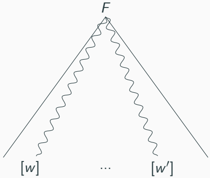
Composition

Plays from XY decompose:

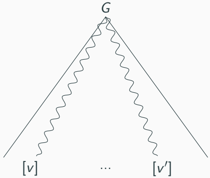


play from X
(with suffix Y)

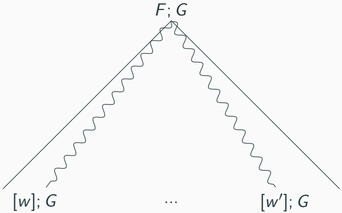
play from Y
(with prefix w/w')



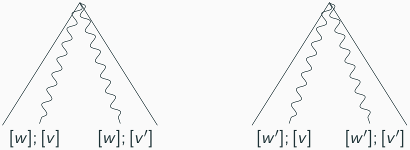
;



=



$(F * F'); G = F; G * F'; G$



$[w]; (G * G') = [w]; G * [w]; G'$

Complexity & Performance

Algorithm

Given: Game G, A and initial position α
Algorithm for solving non-inclusion:

Algorithm

Given: Game G, A and initial position α
Algorithm for solving non-inclusion:

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

Algorithm

Given: Game G, A and initial position α

Algorithm for solving non-inclusion:

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

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

$$F = rhs(F)$$

Algorithm

Given: Game G, A and initial position α
Algorithm for solving non-inclusion:

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

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

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

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

Complexity

Theorem

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

Complexity

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.*

Complexity

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 [MSS05].*

Performance

Comparison of 2EXPTIME *algorithms:*

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

$\underbrace{\hspace{6em}}_{\text{guaranteed blow-up}}$ $\underbrace{\hspace{6em}}_{\text{may be lucky}}$

Performance

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

	naive Kleene		worklist Kleene		Cachat	
$\lvert Q\rvert/\lvert N\rvert/\lvert T\rvert$	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

Future work

Liveness synthesis (infinite words)

Synthesis for systems with branching behavior (trees)

Synthesis for higher-order systems

Future work

Liveness synthesis (infinite words)

Synthesis for systems with branching behavior (trees)

Synthesis for higher-order systems

Solver technology for systems of equations (Newton iteration)

Future work

Liveness synthesis (infinite words)

Synthesis for systems with branching behavior (trees)

Synthesis for higher-order systems

Solver technology for systems of equations (Newton iteration)

Applications, e.g. in hardware synthesis

Thank you!

Questions?