Computing downward closures for stacked counter automata

Georg Zetzsche

Technische Universität Kaiserslautern

STACS 2015
System Observer

Downward closures

Observer sees precisely

Georg Zetzsche (TU KL)

STACS 2015
Downward closures

$u \subseteq v$: $u$ is a subsequence of $v$

Observer sees precisely $L$: $u \subseteq v \implies u \vdash \sigma$

Georg Zetzsche (TU KL)
Downward closures

$u \subseteq v$: $u$ is a subsequence of $v$

Observer sees precisely $L$: $u \subseteq v$

System Observer
LOSSY CHANNEL

aabcbbacbbaaab

Observer
abbbcba

aabcbbacbbaaab
Downward closures

- $u \preceq v$: $u$ is a subsequence of $v$
- $L \downarrow = \{ u \in X^* \mid \exists v \in L: u \preceq v \}$
- Observer sees precisely $L \downarrow$
Downward closures

Theorem (Higman/Haines)

For every language $L \subseteq X^*$, $L \downarrow$ is regular.
Downward closures

Theorem (Higman/Haines)

For every language $L \subseteq X^*$, $L\downarrow$ is regular.

Applications

Given an automaton for $L\downarrow$, many things are decidable:
**Downward closures**

**Theorem (Higman/Haines)**

*For every language $L \subseteq X^*$, $L \downarrow$ is regular.*

**Applications**

Given an automaton for $L \downarrow$, many things are decidable:

- Inclusion of behavior under lossy observation ($K \downarrow \subseteq L \downarrow$)
  
  *Ordinary inclusion almost always undecidable!*

Georg Zetzsche (TU KL)
Downward closures

Theorem (Higman/Haines)

For every language $L \subseteq X^*$, $L\downarrow$ is regular.

Applications

Given an automaton for $L\downarrow$, many things are decidable:

- Inclusion of behavior under lossy observation ($K\downarrow \subseteq L\downarrow$)
  Ordinary inclusion almost always undecidable!
- Which actions occur arbitrarily often? ($a^* \subseteq L\downarrow$)
Downward closures

Theorem (Higman/Haines)

For every language $L \subseteq X^*$, $L\downarrow$ is regular.

Applications

Given an automaton for $L\downarrow$, many things are decidable:

- Inclusion of behavior under lossy observation ($K\downarrow \subseteq L\downarrow$)
  Ordinary inclusion almost always undecidable!
- Which actions occur arbitrarily often? ($a^* \subseteq L\downarrow$)
- Is $a$ ever executed after $b$? ($ab \in L\downarrow$)
Theorem (Higman/Haines)

For every language $L \subseteq X^*$, $L\downarrow$ is regular.

Applications

Given an automaton for $L\downarrow$, many things are decidable:

- Inclusion of behavior under lossy observation ($K\downarrow \subseteq L\downarrow$)
  Ordinary inclusion almost always undecidable!
- Which actions occur arbitrarily often? ($a^* \subseteq L\downarrow$)
- Is $a$ ever executed after $b$? ($ab \in L\downarrow$)
- Can the system run arbitrarily long? ($L\downarrow$ infinite)
Downward closures

**Theorem (Higman/Haines)**

For every language $L \subseteq X^*$, $L \downarrow$ is regular.

**Applications**

Given an automaton for $L \downarrow$, many things are decidable:

- Inclusion of behavior under lossy observation ($K \downarrow \subseteq L \downarrow$)
  
  Ordinary inclusion almost always undecidable!

- Which actions occur arbitrarily often? ($a^* \subseteq L \downarrow$)

- Is $a$ ever executed after $b$? ($ab \in L \downarrow$)

- Can the system run arbitrarily long? ($L \downarrow$ infinite)

**Problem**

- Finite automaton for $L \downarrow$ exists for every $L$.

- How can we compute it?
State of the art

Very few known techniques.
State of the art

Very few known techniques.

Theorem (van Leeuwen 1978/Courcelle 1991)

Downward closures are computable for context-free languages.

Theorem (Abdulla, Boasson, Bouajjani 2001)

Downward closures are computable for context-free FIFO rewriting systems/0L-systems.

Theorem (Habermehl, Meyer, Wimmel 2010)

Downward closures are computable for Petri net languages.
State of the art

Very few known techniques.

Theorem (van Leeuwen 1978/Courcelle 1991)

*Downward closures are computable for context-free languages.*

Theorem (Abdulla, Boasson, Bouajjani 2001)

*Downward closures are computable for context-free FIFO rewriting systems/0L-systems.*
State of the art

Very few known techniques.

Theorem (van Leeuwen 1978/Courcelle 1991)

Downward closures are computable for context-free languages.

Theorem (Abdulla, Boasson, Bouajjani 2001)

Downward closures are computable for context-free FIFO rewriting systems/0L-systems.

- Context-free rules $A \rightarrow w$
- Applied as: $Au \Rightarrow uw$
State of the art

Very few known techniques.

Theorem (van Leeuwen 1978/Courcelle 1991)

*Downward closures are computable for context-free languages.*

Theorem (Abdulla, Boasson, Bouajjani 2001)

*Downward closures are computable for context-free FIFO rewriting systems/0L-systems.*

- Context-free rules $A \rightarrow w$
- Applied as: $Au \Rightarrow uw$

Theorem (Habermehl, Meyer, Wimmel 2010)

*Downward closures are computable for Petri net languages.*
Stacked counter automata

A storage mechanism $M$ consists of:
- States: set $S$ of states
- Operations: partial functions $\alpha_1, \ldots, \alpha_n : S \to S$
- Initial state: $s_0 \in S$
- Final states: $F \subseteq S$

Counter States:
- Operations: increment, decrement, zero test
- Initial and final state: 0

Trivial mechanism
- Consists of one state and no operations.

Georg Zetzsche (TU KL)
Stacked counter automata

A storage mechanism $M$ consists of:

- States: set $S$ of states
- Operations: partial functions $\alpha_1, \ldots, \alpha_n: S \to S$
- Initial state: $s_0 \in S$
- Final states: $F \subseteq S$

Counter

- States: $\mathbb{N}$
- Operations: increment, decrement, zero test
- Initial and final state: 0
Stacked counter automata

A storage mechanism $M$ consists of:

- **States**: set $S$ of states
- **Operations**: partial functions $\alpha_1, \ldots, \alpha_n : S \to S$
- **Initial state**: $s_0 \in S$
- **Final states**: $F \subseteq S$

**Counter**

- **States**: $\mathbb{N}$
- **Operations**: increment, decrement, zero test
- **Initial and final state**: 0

**Trivial mechanism**

Consists of one state and no operations.
$C(M)$: Adding a blind counter

- States: $(s, z)$, $s$ an old state, $z \in \mathbb{Z}$.
- Operations: old operations; increment, decrement for counter
- Initial state: $(s_0, 0)$
- Final states: $(f, 0)$, $f$ final in old mechanism
$C(M)$: Adding a blind counter

- States: $(s, z)$, $s$ an old state, $z \in \mathbb{Z}$.
- Operations: old operations; increment, decrement for counter
- Initial state: $(s_0, 0)$
- Final states: $(f, 0)$, $f$ final in old mechanism

$S(M)$: Building stacks

- States: sequences $\square c_1 \square c_2 \square \cdots \square c_n$, $c_i$ old states
- Operations: push separator, pop if empty, manipulate topmost entry
- Initial and final state: Empty sequence
**C(M): Adding a blind counter**

- States: \((s, z), s\) an old state, \(z \in \mathbb{Z}\).
- Operations: old operations; increment, decrement for counter
- Initial state: \((s_0, 0)\)
- Final states: \((f, 0), f\) final in old mechanism

**S(M): Building stacks**

- States: sequences \(\square c_1 \square c_2 \square \cdots \square c_n, c_i\) old states
- Operations: push separator, pop if empty, manipulate topmost entry
- Initial and final state: Empty sequence

**Stacked counters**

Mechanisms obtained from the trivial one by

- adding blind counters,
- building stacks.
Modeling capabilities

- Generalize both pushdown automata and blind counter automata
Modeling capabilities

- Generalize both pushdown automata and blind counter automata
- Recursive programs with access to private/shared counters
Modeling capabilities

- Generalize both pushdown automata and blind counter automata
- Recursive programs with access to private/shared counters
- Connections to group theory
Modeling capabilities

- Generalize both pushdown automata and blind counter automata
- Recursive programs with access to private/shared counters
- Connections to group theory

Theorem (Main result)

*Downward closures are computable for stacked counter automata.*
Expressiveness

Algebraic extensions

Let $\mathcal{C}$ be a language class. A $\mathcal{C}$-grammar $G$ consists of

- Nonterminals $N$, terminals $T$, start symbol $S \in N$
- Productions $A \to L$ with $L \subseteq (N \cup T)^*$, $L \in \mathcal{C}$
Expressiveness

Algebraic extensions

Let $\mathcal{C}$ be a language class. A $\mathcal{C}$-grammar $G$ consists of

- Nonterminals $N$, terminals $T$, start symbol $S \in N$
- Productions $A \rightarrow L$ with $L \subseteq (N \cup T)^*$, $L \in \mathcal{C}$

$$uAv \Rightarrow uvw \text{ whenever } w \in L.$$
Expressiveness

Algebraic extensions

Let $\mathcal{C}$ be a language class. A $\mathcal{C}$-grammar $G$ consists of

- Nonterminals $N$, terminals $T$, start symbol $S \in N$
- Productions $A \rightarrow L$ with $L \subseteq (N \cup T)^*$, $L \in \mathcal{C}$
  
  $uAv \Rightarrow uwv$ whenever $w \in L$.
- Generated language: $\{ w \in T^* \mid S \Rightarrow^* w \}$. 

Such languages are algebraic over $\mathcal{C}$, class denoted $\text{Alg}_{\mathcal{C}}$. Example $\text{Alg}_{\text{FIN}} \supseteq \text{Alg}_{\text{REG}} = \text{CF}$.
### Algebraic extensions

Let $\mathcal{C}$ be a language class. A $\mathcal{C}$-grammar $G$ consists of

- Nonterminals $N$, terminals $T$, start symbol $S \in N$
- Productions $A \rightarrow L$ with $L \subseteq (N \cup T)^*$, $L \in \mathcal{C}$

$$uAv \Rightarrow uwv \quad \text{whenever } w \in L.$$  

- Generated language: $\{w \in T^* \mid S \Rightarrow^* w\}$.
- Such languages are *algebraic over* $\mathcal{C}$, class denoted $\text{Alg}(\mathcal{C})$. 

---

Expressiveness
Expressiveness

Algebraic extensions

Let $\mathcal{C}$ be a language class. A $\mathcal{C}$-grammar $G$ consists of

- Nonterminals $N$, terminals $T$, start symbol $S \in N$
- Productions $A \rightarrow L$ with $L \subseteq (N \cup T)^*$, $L \in \mathcal{C}$

  $$uAv \Rightarrow uwv \quad \text{whenever } w \in L.$$

- Generated language: $\{w \in T^* \mid S \Rightarrow^* w\}$.
- Such languages are algebraic over $\mathcal{C}$, class denoted $\text{Alg}(\mathcal{C})$.

Example

$\text{Alg}(\text{FIN}) = \text{Alg}(\text{REG}) = \text{CF}$
Definition

Let $X$ be an alphabet.

- $X^\oplus = \{ \mu \mid \mu : X \rightarrow \mathbb{N} \}$, multisets.
Definition

Let $X$ be an alphabet.

- $X^\oplus = \{ \mu \mid \mu : X \to \mathbb{N} \}$, multisets.
- $\Psi : X^* \to X^\oplus$, $\Psi(w)(x) = |w|_x$ is the Parikh map.

Sets of the form $\mu_0\ldots\mu_n$ are called linear.

Finite unions of linear sets are called semilinear.

Semilinear constraints

Let $C$ be a language class. $\text{SLI}_p^C$ denotes the class of languages $\Phi(L)(X)$ for some $L \in C$, a homomorphism $\Phi$, and a semilinear set $S$.

Example

$h : a\ast bc\ast X_\Psi \to a^n b^n c^n$ with $h(a) = a$, $h(c) = a$, $h(b) = b$.
Definition

Let $X$ be an alphabet.

- $X^\oplus = \{\mu \mid \mu : X \to \mathbb{N}\}$, multisets.
- $\Psi : X^* \to X^\oplus$, $\Psi(w)(x) = |w|_x$ is the Parikh map.
- For $F = \{\mu_1, \ldots, \mu_n\} \subseteq X^\oplus$, let $F^\oplus = \{\sum_{i=1}^n a_i \mu_i \mid a_1, \ldots, a_n \in \mathbb{N}\}$.
Definition

Let $X$ be an alphabet.

- $X^\oplus = \{\mu \mid \mu : X \to \mathbb{N}\}$, multisets.
- $\Psi : X^* \to X^\oplus$, $\Psi(w)(x) = |w|_x$ is the Parikh map.
- For $F = \{\mu_1, \ldots, \mu_n\} \subseteq X^\oplus$, let $F^\oplus = \{\sum_{i=1}^{n} a_i \mu_i \mid a_1, \ldots, a_n \in \mathbb{N}\}$
- Sets of the form $\mu_0 + F^\oplus$ are called linear.
Definition

Let $X$ be an alphabet.

- $X^\oplus = \{\mu \mid \mu : X \to \mathbb{N}\}$, multisets.
- $\Psi : X^* \to X^\oplus$, $\Psi(w)(x) = |w|_x$ is the Parikh map.
- For $F = \{\mu_1, \ldots, \mu_n\} \subseteq X^\oplus$, let $F^\oplus = \{\sum_{i=1}^n a_i \mu_i \mid a_1, \ldots, a_n \in \mathbb{N}\}$
- Sets of the form $\mu_0 + F^\oplus$ are called linear.
- Finite unions of linear sets are called semilinear.
Definition

Let $X$ be an alphabet.

- $X^\oplus = \{\mu \mid \mu : X \to \mathbb{N}\}$, *multisets*.
- $\Psi : X^* \to X^\oplus$, $\Psi(w)(x) = |w|_x$ is the *Parikh map*.
- For $F = \{\mu_1, \ldots, \mu_n\} \subseteq X^\oplus$, let $F^\oplus = \{\sum_{i=1}^n a_i \mu_i \mid a_1, \ldots, a_n \in \mathbb{N}\}$
- Sets of the form $\mu_0 + F^\oplus$ are called *linear*.
- Finite unions of linear sets are called *semilinear*.

Semilinear constraints

Let $C$ be a language class. $\text{SLI}(C)$ denotes the class of languages

$$h(L \cap \Psi^{-1}(S))$$

for some $L \in C$, a homomorphism $h$ and a semilinear set $S$. 

---

Georg Zetzsche (TU KL)
**Definition**

Let $X$ be an alphabet.

- $X^\oplus = \{\mu \mid \mu : X \to \mathbb{N}\}$, *multisets*.
- $\Psi : X^* \to X^\oplus$, $\Psi(w)(x) = \lvert w \rvert_x$ is the *Parikh map*.
- For $F = \{\mu_1, \ldots, \mu_n\} \subseteq X^\oplus$, let $F^\oplus = \{\sum_{i=1}^n a_i \mu_i \mid a_1, \ldots, a_n \in \mathbb{N}\}$
- Sets of the form $\mu_0 + F^\oplus$ are called *linear*.
- Finite unions of linear sets are called *semilinear*.

**Semilinear constraints**

Let $C$ be a language class. $\text{SLI}(C)$ denotes the class of languages

$$h(L \cap \Psi^{-1}(S))$$

for some $L \in C$, a homomorphism $h$ and a semilinear set $S$.

**Example**

$$b + (a + c)^\oplus$$
**Definition**

Let $X$ be an alphabet.

- $X^\oplus = \{\mu \mid \mu : X \to \mathbb{N}\}$, *multisets*.
- $\Psi : X^* \to X^\oplus$, $\Psi(w)(x) = |w|_x$ is the *Parikh map*.
- For $F = \{\mu_1, \ldots, \mu_n\} \subseteq X^\oplus$, let $F^\oplus = \{\sum_{i=1}^n a_i \mu_i \mid a_1, \ldots, a_n \in \mathbb{N}\}$
- Sets of the form $\mu_0 + F^\oplus$ are called *linear*.
- Finite unions of linear sets are called *semilinear*.

**Semilinear constraints**

Let $C$ be a language class. $\text{SLI}(C)$ denotes the class of languages

$$h(L \cap \Psi^{-1}(S))$$

for some $L \in C$, a homomorphism $h$ and a semilinear set $S$.

**Example**

$$\Psi^{-1}(b + (a + c)^\oplus)$$
**Definition**

Let $X$ be an alphabet.

- $X^\oplus = \{\mu \mid \mu : X \to \mathbb{N}\}$, multisets.
- $\Psi : X^* \to X^\oplus$, $\Psi(w)(x) = |w|_x$ is the Parikh map.
- For $F = \{\mu_1, \ldots, \mu_n\} \subseteq X^\oplus$, let $F^\oplus = \{\sum_{i=1}^n a_i \mu_i \mid a_1, \ldots, a_n \in \mathbb{N}\}$
- Sets of the form $\mu_0 + F^\oplus$ are called linear.
- Finite unions of linear sets are called semilinear.

**Semilinear constraints**

Let $C$ be a language class. $\text{SLI}(C)$ denotes the class of languages

$$h(L \cap \Psi^{-1}(S))$$

for some $L \in C$, a homomorphism $h$ and a semilinear set $S$.

**Example**

$$a^* b c^* \cap \Psi^{-1}(b + (a + c)^\oplus)$$
Definition

Let $X$ be an alphabet.

- $X^\oplus = \{\mu \mid \mu : X \to \mathbb{N}\}$, multisets.
- $\Psi : X^* \to X^\oplus$, $\Psi(w)(x) = |w|_x$ is the Parikh map.
- For $F = \{\mu_1, \ldots, \mu_n\} \subseteq X^\oplus$, let $F^\oplus = \{\sum_{i=1}^n a_i \mu_i \mid a_1, \ldots, a_n \in \mathbb{N}\}$
- Sets of the form $\mu_0 + F^\oplus$ are called linear.
- Finite unions of linear sets are called semilinear.

Semilinear constraints

Let $C$ be a language class. $\text{SLI}(C)$ denotes the class of languages $h(L \cap \Psi^{-1}(S))$

for some $L \in C$, a homomorphism $h$ and a semilinear set $S$.

Example

$h(a^* b c^* \cap \Psi^{-1}(b + (a + c)^\oplus))$ \hspace{1cm} $h : a, c \mapsto a, \ b \mapsto b.$
Definition

Let $X$ be an alphabet.

- $X^\oplus = \{\mu \mid \mu : X \to \mathbb{N}\}$, multisets.
- $\Psi : X^* \to X^\oplus$, $\Psi(w)(x) = |w|_x$ is the Parikh map.
- For $F = \{\mu_1, \ldots, \mu_n\} \subseteq X^\oplus$, let $F^\oplus = \{\sum_{i=1}^n a_i\mu_i \mid a_1, \ldots, a_n \in \mathbb{N}\}$
- Sets of the form $\mu_0 + F^\oplus$ are called linear.
- Finite unions of linear sets are called semilinear.

Semilinear constraints

Let $C$ be a language class. $\text{SLI}(C)$ denotes the class of languages

$$h(L \cap \Psi^{-1}(S))$$

for some $L \in C$, a homomorphism $h$ and a semilinear set $S$.

Example

$$h(a^*bc^* \cap \Psi^{-1}(b + (a + c)^\oplus)) = \{a^nba^n \mid n \geq 0\}, \quad h: a, c \mapsto a, \quad b \mapsto b.$$
A hierarchy of language classes

Hierarchy

\[ F_0 = \text{finite languages}, \]
\[ G_i = \text{Alg}(F_i), \quad F_{i+1} = \text{SLI}(G_i), \quad F = \bigcup_{i \geq 0} F_i. \]
A hierarchy of language classes

Hierarchy

\[ F_0 = \text{finite languages}, \]
\[ G_i = \text{Alg}(F_i), \quad F_{i+1} = \text{SLI}(G_i), \quad F = \bigcup_{i \geq 0} F_i. \]

In particular: \( G_0 = \text{CF}. \)
A hierarchy of language classes

Hierarchy

\[ F_0 = \text{finite languages}, \]

\[ G_i = \text{Alg}(F_i), \quad F_{i+1} = \text{SLI}(G_i), \quad F = \bigcup_{i \geq 0} F_i. \]

In particular: \( G_0 = \text{CF}. \)

\[ F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \cdots \subseteq F \]
A hierarchy of language classes

Hierarchy

F₀ = finite languages,

Gᵢ = Alg(Fᵢ), \quad Fᵢ₊₁ = SLI(Gᵢ), \quad F = \bigcup_{i \geq 0} Fᵢ.

In particular: G₀ = CF.

Theorem

Ł(S(S(M)))) = Alg(Ł(M))
A hierarchy of language classes

Hierarchy

\[ F_0 = \text{finite languages}, \]
\[ G_i = \text{Alg}(F_i), \quad F_{i+1} = \text{SLI}(G_i), \quad F = \bigcup_{i \geq 0} F_i. \]

In particular: \( G_0 = \text{CF}. \)

\[ F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \cdots \subseteq F \]

Theorem

\[ \mathcal{L}(S(S(M))) = \text{Alg}(\mathcal{L}(M)), \quad \bigcup_{i \geq 0} \mathcal{L}(C^i(M)) = \text{SLI}(\mathcal{L}(M)). \]
A hierarchy of language classes

**Hierarchy**

\[ F_0 = \text{finite languages}, \]
\[ G_i = \text{Alg}(F_i), \quad F_{i+1} = \text{SLI}(G_i), \quad F = \bigcup_{i \geq 0} F_i. \]

In particular: \( G_0 = \text{CF} \).

\[ F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \cdots \subseteq F \]

**Theorem**

\[ \mathcal{L}(S(S(M))) = \text{Alg}(\mathcal{L}(M)), \quad \bigcup_{i \geq 0} \mathcal{L}(C^i(M)) = \text{SLI}(\mathcal{L}(M)). \]

**Corollary**

*Stacked counter automata accept precisely the languages in \( F \).*
van Leeuwen proved a stronger statement:

**Theorem (van Leeuwen 1978)**

*If \( C \) is closed under regular intersections: Downward closures computable for \( C \) \( \implies \) computable for \( \text{Alg}(C) \).*
van Leeuwen proved a stronger statement:

**Theorem (van Leeuwen 1978)**

*If $C$ is closed under regular intersections: Downward closures computable for $C$ $\implies$ computable for $\text{Alg}(C)$.*

**Consequence**

Algorithm for $F_i$ $\implies$ Algorithm for $G_i = \text{Alg}(F_i)$. 
Ingredient II

\[ F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \cdots \subseteq F \]

Problem

- Computability preserved by \( \text{Alg}(\cdot) \)
Ingredient II

\[ F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \cdots \subseteq F \]

Problem

- Computability preserved by \( \text{Alg}(\cdot) \)
- No preservation for \( \text{SLI}(\cdot) \)
Ingredient II

\[ F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \cdots \subseteq F \]

Problem

- Computability preserved by Alg(\( \cdot \))
- No preservation for SLI(\( \cdot \))

Idea

- Given \( L \in F_{i+1} = \text{SLI}(G_i) \), construct \( L' \in G_i \) with \( L' \downarrow = L \downarrow \).
### Ingredient II

\[ F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \cdots \subseteq F \]

### Problem
- Computability preserved by Alg(\(\cdot\))
- No preservation for SLI(\(\cdot\))

### Idea
- Given \( L \in F_{i+1} = \text{SLI}(G_i) \), construct \( L' \in G_i \) with \( L' \downarrow = L \downarrow \).
- Wlog \( L = K \cap \psi^{-1}(S) \), \( K \in G_i \), \( S \) semilinear
Problem

- Computability preserved by Alg(·)
- No preservation for SLI(·)

Idea

- Given $L \in F_{i+1} = \text{SLI}(G_i)$, construct $L' \in G_i$ with $L' \downarrow = L \downarrow$.
- Wlog $L = K \cap \psi^{-1}(S)$, $K \in G_i$, $S$ semilinear
- Construct $K' \in G_i$ with $K \cap \psi^{-1}(S) \subseteq K' \subseteq (K \cap \psi^{-1}(S)) \downarrow$
Problem

- Computability preserved by Alg(·)
- No preservation for SLI(·)

Idea

- Given $L \in F_{i+1} = \text{SLI}(G_i)$, construct $L' \in G_i$ with $L' \downarrow = L \downarrow$.
- Wlog $L = K \cap \psi^{-1}(S)$, $K \in G_i$, $S$ semilinear
- Construct $K' \in G_i$ with $K \cap \psi^{-1}(S) \subseteq K' \subseteq (K \cap \psi^{-1}(S)) \downarrow$
- Plan: Use finite state transductions to stay within $G_i$
Problem

- Computability preserved by Alg(·)
- No preservation for SLI(·)

Idea

- Given \( L \in F_{i+1} = \text{SLI}(G_i) \), construct \( L' \in G_i \) with \( L'\downarrow = L\downarrow \).
- Wlog \( L = K \cap \Psi^{-1}(S) \), \( K \in G_i \), \( S \) semilinear
- Construct \( K' \in G_i \) with \( K \cap \Psi^{-1}(S) \subseteq K' \subseteq (K \cap \Psi^{-1}(S))\downarrow \)
- Plan: Use finite state transductions to stay within \( G_i \)
- Annotate words with additional information
Ingredient II

F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \cdots \subseteq F

Problem

- Computability preserved by Alg(·)
- No preservation for SLI(·)

Idea

- Given L \in F_{i+1} = SLI(G_i), construct L' \in G_i with L'\downarrow = L\downarrow.
- Wlog L = K \cap \psi^{-1}(S), K \in G_i, S semilinear
- Construct K' \in G_i with K \cap \psi^{-1}(S) \subseteq K' \subseteq (K \cap \psi^{-1}(S))\downarrow
- Plan: Use finite state transductions to stay within G_i
- Annotate words with additional information

Theorem (Parikh)

For context-free L, 
\psi(L) is semilinear.
Ingredient II

\[ F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \cdots \subseteq F \]

Problem
- Computability preserved by \( \text{Alg}(\cdot) \)
- No preservation for \( \text{SLI}(\cdot) \)

Idea
- Given \( L \in F_{i+1} = \text{SLI}(G_i) \), construct \( L' \in G_i \) with \( L'\downarrow = L\downarrow \).
- Without loss of generality \( L = K \cap \psi^{-1}(S) \), \( K \in G_i \), \( S \) semilinear.
- Construct \( K' \in G_i \) with \( K \cap \psi^{-1}(S) \subseteq K' \subseteq (K \cap \psi^{-1}(S))\downarrow \)
- Plan: Use finite state transductions to stay within \( G_i \)
- Annotate words with additional information

Theorem (Parikh)

For context-free \( L \), \( \psi(L) \) is semilinear.

\[ \psi(L) = \bigcup_{i=1}^{n} \mu_i + F_i^{\oplus} \]

- \( \mu_i \): constant vector
- \( F_i \): set of period vectors
Task

Use transducer to pick all words whose Parikh decomposition avoids a certain period vector.
Task
Use transducer to pick all words whose Parikh decomposition avoids a certain period vector.

Parikh annotation I

\[ L = \{a^n b^m \mid m = n \text{ or } m = 2n\}, \quad \Psi(L) = (a + b)^\oplus \cup (a + 2b)^\oplus. \]
Task

Use transducer to pick all words whose Parikh decomposition avoids a certain period vector.

Parikh annotation I

$L = \{a^n b^m \mid m = n \text{ or } m = 2n\}$, \[\Psi(L) = (a + b)^\oplus \cup (a + 2b)^\oplus.\]
Task
Use transducer to pick all words whose Parikh decomposition avoids a certain period vector.

Parikh annotation I
\[ L = \{ a^n b^m \mid m = n \text{ or } m = 2n \}, \quad \Psi(L) = (a + b)^\oplus \cup (a + 2b)^\oplus. \]
\[ K = \{ (\sigma a)^n b^n \mid n \geq 0 \} \cup \{ (\tau a)^n (2b)^n \mid n \geq 0 \} \]
Task
Use transducer to pick all words whose Parikh decomposition avoids a certain period vector.

Parikh annotation I

\[ L = \{a^n b^m \mid m = n \text{ or } m = 2n\}, \quad \Psi(L) = (a + b)^\top \cup (a + 2b)^\top. \]
\[ K = \{(\sigma a)^n b^n \mid n \geq 0\} \cup \{(\tau a)^n (2b)^n \mid n \geq 0\} \]

Parikh annotation II

\[ L = (ab)^*(ca^* \cup db^*), \quad \Psi(L) = c + \{a + b, a\}^\top \cup d + \{a + b, b\}^\top. \]
Task

Use transducer to pick all words whose Parikh decomposition avoids a certain period vector.

Parikh annotation I

\[ L = \{a^n b^m \mid m = n \text{ or } m = 2n\}, \quad \Psi(L) = (a + b)^\oplus \cup (a + 2b)^\oplus. \]

\[ K = \{(\sigma a)^n b^n \mid n \geq 0\} \cup \{(\tau a)^n (2b)^n \mid n \geq 0\} \]

Parikh annotation II

\[ L = (ab)^* (ca^* \cup db^*), \quad \Psi(L) = c + \{a + b, a\}^\oplus \cup d + \{a + b, b\}^\oplus. \]
Task

Use transducer to pick all words whose Parikh decomposition avoids a certain period vector.

Parikh annotation I

\[ L = \{ a^n b^m \mid m = n \text{ or } m = 2n \}, \quad \Psi(L) = (a + b)^{\uparrow \sigma} \cup (a + 2b)^{\uparrow \tau} \]

\[ K = \{ (\sigma a)^n b^n \mid n \geq 0 \} \cup \{ (\tau a)^n (2b)^n \mid n \geq 0 \} \]

Parikh annotation II

\[ L = (ab)^*(ca^* \cup db^*) \]

\[ \Psi(L) = c + \{ a + b, a \}^{\uparrow \alpha} \cup d + \{ a + b, b \}^{\uparrow \beta} \]

\[ K = \alpha(\mu ab)^* c(\nu a)^* \cup \beta(\sigma ab)^* d(\tau b)^* \]
Parikh annotations

- New language in the same class
- Additional symbols encode decomposition of Parikh image into constant and period vectors
- Adding period vectors by inserting words
Theorem

For each level of the hierarchy, one can construct Parikh annotations.
Theorem

For each level of the hierarchy, one can construct Parikh annotations.

Corollary

Given $L \in G_i$ and semilinear $S$, one can construct $L' \in G_i$ with $L \cap \psi^{-1}(S) \subseteq L' \subseteq (L \cap \psi^{-1}(S))\downarrow$. 
Theorem

For each level of the hierarchy, one can construct Parikh annotations.

Corollary

Given \( L \in G_i \) and semilinear \( S \), one can construct \( L' \in G_i \) with \( L \cap \Psi^{-1}(S) \subseteq L' \subseteq (L \cap \Psi^{-1}(S))\downarrow \).

- Select all words where adding period vectors leads into \( S \).
Theorem

For each level of the hierarchy, one can construct Parikh annotations.

Corollary

Given \( L \in G_i \) and semilinear \( S \), one can construct \( L' \in G_i \) with
\[
L \cap \psi^{-1}(S) \subseteq L' \subseteq (L \cap \psi^{-1}(S))\downarrow.
\]

- Select all words where adding period vectors leads into \( S \)
- Downward closed set of multisets of period vectors
Theorem

For each level of the hierarchy, one can construct Parikh annotations.

Corollary

Given \( L \in G_i \) and semilinear \( S \), one can construct \( L' \in G_i \) with
\[
L \cap \Psi^{-1}(S) \subseteq L' \subseteq (L \cap \Psi^{-1}(S))\downarrow.
\]

- Select all words where adding period vectors leads into \( S \)
- Downward closed set of multisets of period vectors
  - Finitely many forbidden sub-multisets
Theorem

For each level of the hierarchy, one can construct Parikh annotations.

Corollary

Given \( L \in G_i \) and semilinear \( S \), one can construct \( L' \in G_i \) with
\[
L \cap \Psi^{-1}(S) \subseteq L' \subseteq (L \cap \Psi^{-1}(S)) \downarrow.
\]

- Select all words where adding period vectors leads into \( S \)
- Downward closed set of multisets of period vectors
  - Finitely many forbidden sub-multisets
  - Presburger-definable, hence computable
Theorem

For each level of the hierarchy, one can construct Parikh annotations.

Corollary

Given $L \in G_i$ and semilinear $S$, one can construct $L' \in G_i$ with $L \cap \Psi^{-1}(S) \subseteq L' \subseteq (L \cap \Psi^{-1}(S))\downarrow$.

- Select all words where adding period vectors leads into $S$
- Downward closed set of multisets of period vectors
  - Finitely many forbidden sub-multisets
  - Presburger-definable, hence computable
- Recognizable by finite automaton
Conclusion

- Downward closure: promising abstraction of languages
- Computability known for few language classes
- Computable for stacked counter automata

Future work

Applications of downward closures
- Downward closures for other WQOs
- Further classes of systems

Thank you for your attention!
Conclusion

- Downward closure: promising abstraction of languages
- Computability known for few language classes
- Computable for stacked counter automata

Future work

- Applications of downward closures
- Downward closures for other WQOs
- Further classes of systems
Conclusion

- Downward closure: promising abstraction of languages
- Computability known for few language classes
- Computable for stacked counter automata

Future work

- Applications of downward closures
- Downward closures for other WQOs
- Further classes of systems

Thank you for your attention!