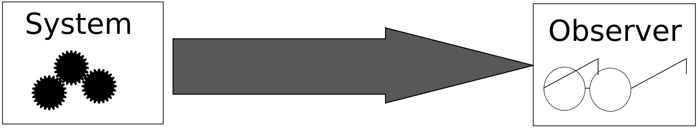
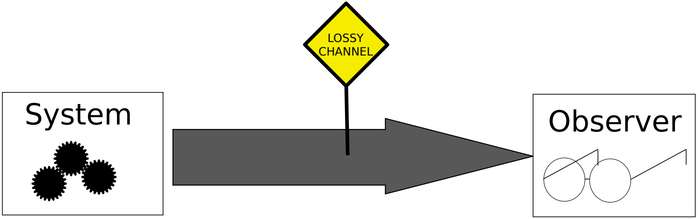
Computing downward closures for stacked counter automata

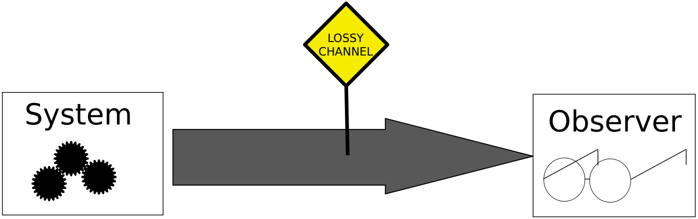
Georg Zetzsche

Technische Universität Kaiserslautern

STACS 2015


System
Observer

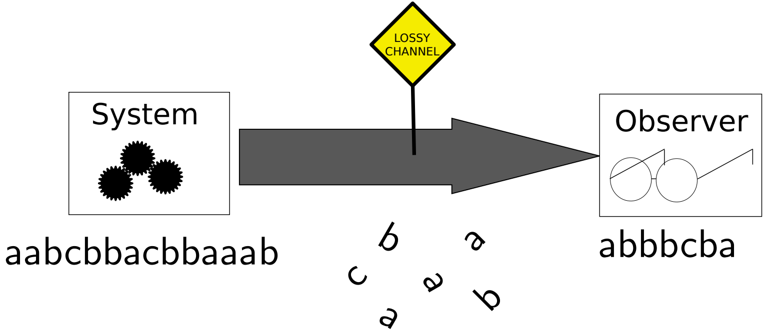

LOSSY CHANNEL
System
Observer



aabcbbacbbaaab

abbbcba

aabcbbacbbaaab



aabcbbacbbaaab

Downward closures

- $u \preccurlyeq v$: u is a subsequence of v
- $L{\downarrow} = \{u \in X^* \mid \exists v \in L\colon u \preccurlyeq v\}$
- 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!

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? ($ab \in L\downarrow$)

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)

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.

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 $A \to w$
- 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 $A \to w$
- 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 $\alpha_1, \ldots, \alpha_n \colon S \to S$
- Initial state: $s_0 \in S$
- Final states: $F \subseteq S$

Stacked counter automata

A storage mechanism M consists of:

- States: set S of states
- Operations: partial functions $\alpha_1, \ldots, \alpha_n \colon S \to S$
- 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 $\alpha_1, \ldots, \alpha_n \colon S \to S$
- 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

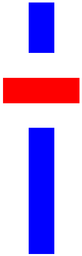
$S(M)$: Building stacks

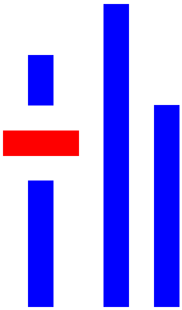
- States: sequences $\Box c_1 \Box c_2 \Box \cdots \Box c_n$, c_i old states
- Operations: push separator, pop if empty, manipulate topmost entry
- Initial and final state: Empty sequence

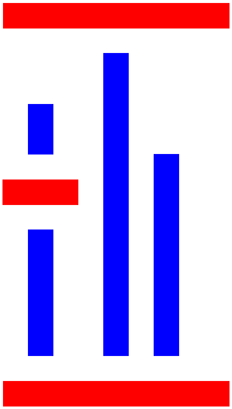
Stacked counters

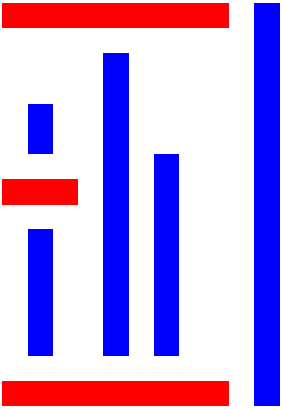
Mechanisms obtained from the trivial one by

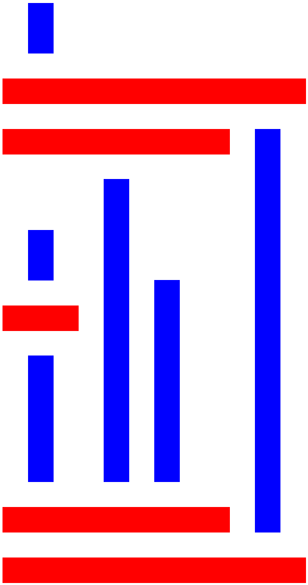
- adding blind counters,
- building stacks.











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 $A \to L$ with $L \subseteq (N \cup T)^*$, $L \in \mathcal{C}$

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

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 $A \to L$ with $L \subseteq (N \cup T)^*$, $L \in \mathcal{C}$

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

- Generated language: $\{w \in T^* \mid S \Rightarrow^* w\}$.

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 $L \subseteq (N \cup T)^*$, $L \in \mathcal{C}$

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

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

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 $A \to L$ 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 $\mathrm{Alg}(\mathcal{C})$.

Example

$\mathrm{Alg}(\mathrm{FIN}) = \mathrm{Alg}(\mathrm{REG}) = \mathrm{CF}$

Definition

Let X be an alphabet.

- $X^{\oplus} = \{\mu \mid \mu\colon X \to \mathbb{N}\}$, *multisets*.

Definition

Let X be an alphabet.

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

Definition

Let X be an alphabet.

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

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

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

- $X^{\oplus} = \{\mu \mid \mu\colon X \to \mathbb{N}\}$, *multisets*.
- $\Psi\colon X^* \to X^{\oplus}$, $\Psi(w)(x) = |w|_x$ 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 $\mathcal{C}$ be a language class. $\mathsf{SLI}(\mathcal{C})$ denotes the class of languages

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

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

Definition

Let X be an alphabet.

- $X^{\oplus} = \{\mu \mid \mu \colon X \to \mathbb{N}\}$, *multisets*.
- $\Psi \colon X^* \to X^{\oplus}$, $\Psi(w)(x) = |w|_x$ 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 $\mathcal{C}$ be a language class. $\mathsf{SLI}(\mathcal{C})$ denotes the class of languages

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

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

Example

$$b + (a + c)^{\oplus}$$

Definition

Let X be an alphabet.

- $X^{\oplus} = \{\mu \mid \mu\colon X \to \mathbb{N}\}$, *multisets*.
- $\Psi\colon X^* \to X^{\oplus}$, $\Psi(w)(x) = |w|_x$ 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 $\mathcal{C}$ be a language class. $\mathsf{SLI}(\mathcal{C})$ denotes the class of languages

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

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

Example

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

Definition

Let X be an alphabet.

- $X^{\oplus} = \{\mu \mid \mu\colon X \to \mathbb{N}\}$, *multisets*.
- $\Psi\colon X^* \to X^{\oplus}$, $\Psi(w)(x) = |w|_x$ 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 $\mathcal{C}$ be a language class. $\mathsf{SLI}(\mathcal{C})$ denotes the class of languages

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

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

Example

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

Definition

Let X be an alphabet.

- $X^{\oplus} = \{\mu \mid \mu\colon X \to \mathbb{N}\}$, *multisets*.
- $\Psi\colon X^* \to X^{\oplus}$, $\Psi(w)(x) = |w|_x$ 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 $\mathcal{C}$ be a language class. $\mathsf{SLI}(\mathcal{C})$ denotes the class of languages

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

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

Example

$h(a^*bc^* \cap \Psi^{-1}(b + (a+c)^{\oplus}))$ $\qquad h\colon a, c \mapsto a,\ b \mapsto b.$

Definition

Let X be an alphabet.

- $X^{\oplus} = \{\mu \mid \mu\colon X \to \mathbb{N}\}$, *multisets*.
- $\Psi\colon X^* \to X^{\oplus}$, $\Psi(w)(x) = |w|_x$ 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 $\mathcal{C}$ be a language class. $\mathsf{SLI}(\mathcal{C})$ denotes the class of languages

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

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

Example

$h(a^*bc^* \cap \Psi^{-1}(b + (a+c)^{\oplus})) = \{a^nba^n \mid n \geqslant 0\},\ h\colon a, c \mapsto a,\ b \mapsto b.$

A hierarchy of language classes

Hierarchy

$$F_0 = \text{finite languages},$$

$$G_i = \mathrm{Alg}(F_i), \qquad F_{i+1} = \mathrm{SLI}(G_i), \qquad F = \bigcup_{i \geq 0} F_i.$$

A hierarchy of language classes

Hierarchy

$$F_0 = \text{finite languages},$$

$$G_i = \mathrm{Alg}(F_i), \qquad F_{i+1} = \mathrm{SLI}(G_i), \qquad F = \bigcup_{i \geqslant 0} F_i.$$

In particular: $G_0 = \mathrm{CF}$.

A hierarchy of language classes

Hierarchy

$$\mathsf{F}_0 = \text{finite languages},$$

$$\mathsf{G}_i = \mathsf{Alg}(\mathsf{F}_i), \qquad \mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{G}_i), \qquad \mathsf{F} = \bigcup_{i \geqslant 0} \mathsf{F}_i.$$

In particular: $\mathsf{G}_0 = \mathsf{CF}$.

$$\mathsf{F}_0 \subseteq \mathsf{G}_0 \subseteq \mathsf{F}_1 \subseteq \mathsf{G}_1 \subseteq \cdots \subseteq \mathsf{F}$$

A hierarchy of language classes

Hierarchy

$$\mathsf{F}_0 = \text{finite languages},$$

$$\mathsf{G}_i = \mathsf{Alg}(\mathsf{F}_i), \qquad \mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{G}_i), \qquad \mathsf{F} = \bigcup_{i\geqslant 0} \mathsf{F}_i.$$

In particular: $\mathsf{G}_0 = \mathsf{CF}$.

$$\mathsf{F}_0 \subseteq \mathsf{G}_0 \subseteq \mathsf{F}_1 \subseteq \mathsf{G}_1 \subseteq \cdots \subseteq \mathsf{F}$$

Theorem

$\mathcal{L}(S(S(M))) = \mathsf{Alg}(\mathcal{L}(M))$

A hierarchy of language classes

Hierarchy

$$\mathsf{F}_0 = \text{finite languages},$$

$$\mathsf{G}_i = \mathsf{Alg}(\mathsf{F}_i), \qquad \mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{G}_i), \qquad \mathsf{F} = \bigcup_{i \geqslant 0} \mathsf{F}_i.$$

In particular: $\mathsf{G}_0 = \mathsf{CF}$.

$$\mathsf{F}_0 \subseteq \mathsf{G}_0 \subseteq \mathsf{F}_1 \subseteq \mathsf{G}_1 \subseteq \cdots \subseteq \mathsf{F}$$

Theorem

$\mathcal{L}(S(S(M))) = \mathsf{Alg}(\mathcal{L}(M))$, $\qquad \bigcup_{i \geqslant 0} \mathcal{L}(C^i(M)) = \mathsf{SLI}(\mathcal{L}(M))$.

A hierarchy of language classes

Hierarchy

$$\mathsf{F}_0 = \text{finite languages},$$

$$\mathsf{G}_i = \mathsf{Alg}(\mathsf{F}_i), \qquad \mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{G}_i), \qquad \mathsf{F} = \bigcup_{i\geqslant 0} \mathsf{F}_i.$$

In particular: $\mathsf{G}_0 = \mathsf{CF}$.

$$\mathsf{F}_0 \subseteq \mathsf{G}_0 \subseteq \mathsf{F}_1 \subseteq \mathsf{G}_1 \subseteq \cdots \subseteq \mathsf{F}$$

Theorem

$\mathcal{L}(S(S(M))) = \mathsf{Alg}(\mathcal{L}(M))$, $\quad \bigcup_{i\geqslant 0} \mathcal{L}(C^i(M)) = \mathsf{SLI}(\mathcal{L}(M))$.

Corollary

Stacked counter automata accept precisely the languages in F.

Ingredient I

van Leeuwen proved a stronger statement:

Theorem (van Leeuwen 1978)

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

Ingredient I

van Leeuwen proved a stronger statement:

Theorem (van Leeuwen 1978)

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

Consequence

Algorithm for $\mathsf{F}_i \implies$ Algorithm for $\mathsf{G}_i = \mathsf{Alg}(\mathsf{F}_i)$.

Ingredient II

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

Problem

- Computability preserved by $\mathsf{Alg}(\cdot)$

Ingredient II

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

Problem

- Computability preserved by $\mathsf{Alg}(\cdot)$
- No preservation for $\mathsf{SLI}(\cdot)$

Ingredient II

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

Problem

- Computability preserved by $\mathsf{Alg}(\cdot)$
- No preservation for $\mathsf{SLI}(\cdot)$

Idea

- Given $L \in F_{i+1} = \mathsf{SLI}(G_i)$, 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 $\mathrm{Alg}(\cdot)$
- No preservation for $\mathrm{SLI}(\cdot)$

Idea

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

Ingredient II

$$\mathsf{F}_0 \subseteq \mathsf{G}_0 \subseteq \mathsf{F}_1 \subseteq \mathsf{G}_1 \subseteq \cdots \subseteq \mathsf{F}$$

Problem

- Computability preserved by $\mathsf{Alg}(\cdot)$
- No preservation for $\mathsf{SLI}(\cdot)$

Idea

- Given $L \in \mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{G}_i)$, construct $L' \in \mathsf{G}_i$ with $L'{\downarrow} = L{\downarrow}$.
- Wlog $L = K \cap \Psi^{-1}(S)$, $K \in \mathsf{G}_i$, S semilinear
- Construct $K' \in \mathsf{G}_i$ with $K \cap \Psi^{-1}(S) \subseteq K' \subseteq (K \cap \Psi^{-1}(S)){\downarrow}$

Ingredient II

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

Problem

- Computability preserved by $\mathsf{Alg}(\cdot)$
- No preservation for $\mathsf{SLI}(\cdot)$

Idea

- Given $L \in F_{i+1} = \mathsf{SLI}(G_i)$, 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

Ingredient II

$$\mathsf{F}_0 \subseteq \mathsf{G}_0 \subseteq \mathsf{F}_1 \subseteq \mathsf{G}_1 \subseteq \cdots \subseteq \mathsf{F}$$

Problem

- Computability preserved by $\mathsf{Alg}(\cdot)$
- No preservation for $\mathsf{SLI}(\cdot)$

Idea

- Given $L \in \mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{G}_i)$, construct $L' \in \mathsf{G}_i$ with $L'\!\downarrow = L\!\downarrow$.
- Wlog $L = K \cap \Psi^{-1}(S)$, $K \in \mathsf{G}_i$, S semilinear
- Construct $K' \in \mathsf{G}_i$ 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

$$\mathsf{F}_0 \subseteq \mathsf{G}_0 \subseteq \mathsf{F}_1 \subseteq \mathsf{G}_1 \subseteq \cdots \subseteq \mathsf{F}$$

Problem

- Computability preserved by $\mathsf{Alg}(\cdot)$
- No preservation for $\mathsf{SLI}(\cdot)$

Idea

- Given $L \in \mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{G}_i)$, construct $L' \in \mathsf{G}_i$ with $L'\!\downarrow = L\!\downarrow$.
- Wlog $L = K \cap \Psi^{-1}(S)$, $K \in \mathsf{G}_i$, S semilinear
- Construct $K' \in \mathsf{G}_i$ 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 $\mathsf{Alg}(\cdot)$
- No preservation for $\mathsf{SLI}(\cdot)$

Idea

- Given $L \in F_{i+1} = \mathsf{SLI}(G_i)$, 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.

$$\Psi(L) = \bigcup_{i=1}^{n} \mu_i + F_i^{\oplus}$$

- μ_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

$$L = \{a^n b^m \mid m = n \text{ or } m = 2n\}, \qquad \Psi(L) = (a \underset{\sigma}{\underset{\uparrow}{+}} b)^{\oplus} \quad \cup \quad (a \underset{\tau}{\underset{\uparrow}{+}} 2b)^{\oplus}.$$

Task

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

Parikh annotation I

$L = \{a^n b^m \mid m = n \text{ or } m = 2n\}, \qquad \Psi(L) = (a \underset{\sigma}{\overset{\uparrow}{+}} b)^{\oplus} \quad \cup \quad (a \underset{\tau}{\overset{\uparrow}{+}} 2b)^{\oplus}.$

$K = \{(\sigma a)^n b^n \mid n \geqslant 0\} \cup \{(\tau a)^n (2b)^n \mid n \geqslant 0\}$

Task

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

Parikh annotation I

$$L = \{a^n b^m \mid m = n \text{ or } m = 2n\}, \qquad \Psi(L) = (a \underset{\substack{\uparrow \\ \sigma}}{+} b)^{\oplus} \quad \cup \quad (a \underset{\substack{\uparrow \\ \tau}}{+} 2b)^{\oplus}.$$

$$K = \{(\sigma a)^n b^n \mid n \geqslant 0\} \cup \{(\tau a)^n (2b)^n \mid n \geqslant 0\}$$

Parikh annotation II

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

Task

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

Parikh annotation I

$L = \{a^n b^m \mid m = n \text{ or } m = 2n\}, \qquad \Psi(L) = (a \underset{\sigma}{\underset{\uparrow}{+}} b)^{\oplus} \quad \cup \quad (a \underset{\tau}{\underset{\uparrow}{+}} 2b)^{\oplus}.$

$K = \{(\sigma a)^n b^n \mid n \geqslant 0\} \cup \{(\tau a)^n (2b)^n \mid n \geqslant 0\}$

Parikh annotation II

$L = (ab)^*(ca^* \cup db^*), \qquad \Psi(L) = \underset{\alpha}{\underset{\uparrow}{c}} + \{a \underset{\mu}{\underset{\uparrow}{+}} b, \underset{\nu}{\underset{\uparrow}{a}}\}^{\oplus} \quad \cup \quad \underset{\beta}{\underset{\uparrow}{d}} + \{a \underset{\sigma}{\underset{\uparrow}{+}} b, \underset{\tau}{\underset{\uparrow}{b}}\}^{\oplus}.$

Task

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

Parikh annotation I

$$L = \{a^n b^m \mid m = n \text{ or } m = 2n\}, \qquad \Psi(L) = (a \underset{\overset{\uparrow}{\sigma}}{+} b)^{\oplus} \;\cup\; (a \underset{\overset{\uparrow}{\tau}}{+} 2b)^{\oplus}.$$

$$K = \{(\sigma a)^n b^n \mid n \geqslant 0\} \cup \{(\tau a)^n (2b)^n \mid n \geqslant 0\}$$

Parikh annotation II

$$L = (ab)^*(ca^* \cup db^*), \qquad \Psi(L) = \underset{\overset{\uparrow}{\alpha}}{c} + \{a \underset{\overset{\uparrow}{\mu}}{+} b, \underset{\overset{\uparrow}{\nu}}{a}\}^{\oplus} \;\cup\; \underset{\overset{\uparrow}{\beta}}{d} + \{a \underset{\overset{\uparrow}{\sigma}}{+} b, \underset{\overset{\uparrow}{\tau}}{b}\}^{\oplus}.$$

$$K = \alpha(\mu ab)^* c(\nu a)^* \;\cup\; \beta(\sigma ab)^* d(\tau b)^*$$

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 $L \in \mathsf{G}_i$ *and semilinear* S*, one can construct* $L' \in \mathsf{G}_i$ *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 $L \in \mathsf{G}_i$ *and semilinear* S, *one can construct* $L' \in \mathsf{G}_i$ *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 $L \in \mathsf{G}_i$ *and semilinear* S*, one can construct* $L' \in \mathsf{G}_i$ *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 $L \in \mathsf{G}_i$ and semilinear S, one can construct $L' \in \mathsf{G}_i$ 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 $L \in \mathsf{G}_i$ *and semilinear* S, *one can construct* $L' \in \mathsf{G}_i$ *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 $L \in \mathsf{G}_i$ *and semilinear* S*, one can construct* $L' \in \mathsf{G}_i$ *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

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!