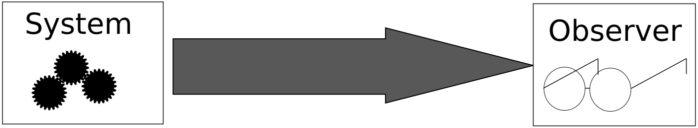
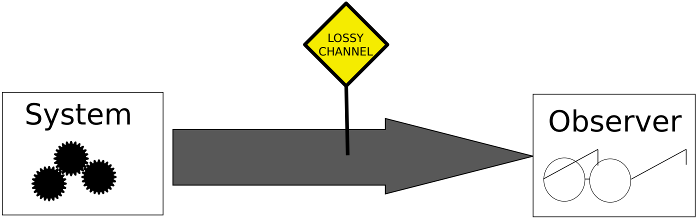
Effectively Regular Downward Closures

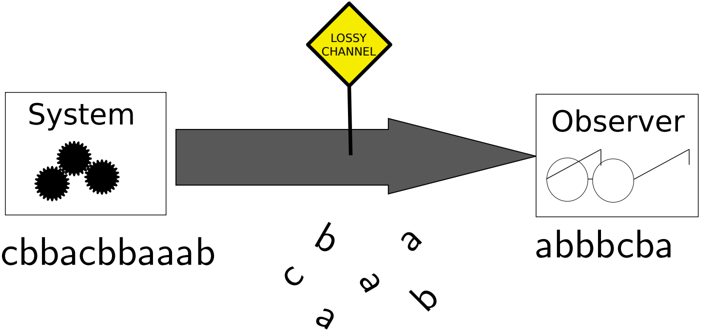
Georg Zetzsche

Technische Universität Kaiserslautern

LSV Cachan, 28 October 2014


System
Observer

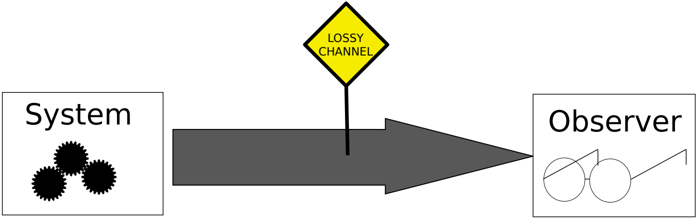

System
LOSSY
CHANNEL
Observer



aabcbbacbbaaab

abbbcba

$\underline{\mathsf{a}}\mathsf{ab}\underline{\mathsf{c}}\mathsf{bb}\underline{\mathsf{a}}\mathsf{cb}\underline{\mathsf{b}}\mathsf{a}\underline{\mathsf{aab}}$



aabcbbacbbaaab

abbbcba

$\underline{\text{a}}\text{ab}\underline{\text{c}}\text{bb}\underline{\text{a}}\text{cb}\underline{\text{b}}\text{a}\underline{\text{aab}}$

Downward Closures

- $u \preccurlyeq v$: u is a subsequence of v
- $L{\downarrow} = \{u \in X^* \mid \exists v \in L\colon u \preccurlyeq v\}$
- Oberver 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 $A \rightarrow w$
- 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 maps $\alpha_1, \ldots, \alpha_n \colon S \to S$
- 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 maps $\alpha_1, \ldots, \alpha_n \colon S \to S$
- 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 maps $\alpha_1, \dots, \alpha_n \colon S \to S$
- 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

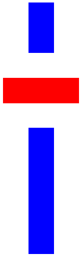
$S(M)$: Building stacks

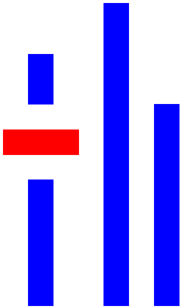
- States: sequences $\square c_1 \square c_2 \square \cdots \square c_n$, c_i old states
- Operations: push separator, pop if empty, manipulate topmost entry
- Initial and final state: Empty sequence

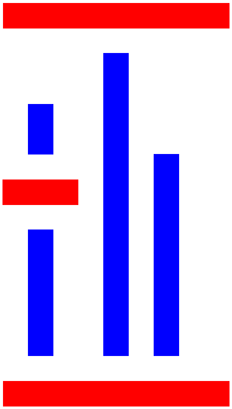
Stacked counters

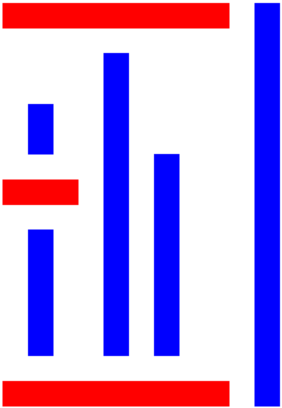
Mechanisms obtained from the trivial one by

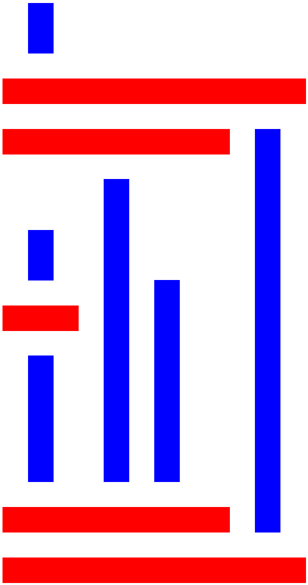
- adding blind counters,
- building stacks.











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 (Z. 2014)

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

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

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

A hierarchy of language classes

Hierarchy

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

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

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

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

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

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 $\mathrm{Alg}(\mathcal{C})$.

Consequence

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

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

Suppose there is only one nonterminal S.

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathsf{SF}(G){\downarrow}$.

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G){\downarrow}$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L{\downarrow}$ for $S \to L{\downarrow}$):

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G){\downarrow}$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L{\downarrow}$ for $S \to L{\downarrow}$):
  - $\mathrm{SF}(G){\downarrow} = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G){\downarrow}$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L{\downarrow}$ for $S \to L{\downarrow}$):
  - $\mathrm{SF}(G){\downarrow} = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $$x_1 x_2 S y_1 y_2$$

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G){\downarrow}$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L{\downarrow}$ for $S \to L{\downarrow}$):
  - $\mathrm{SF}(G){\downarrow} = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $\underline{S} \Rightarrow x_1 S$ $\qquad$ $x_1 x_2 S y_1 y_2$

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L\downarrow$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G)\downarrow$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L\downarrow$ for $S \to L\downarrow$):
  - $\mathrm{SF}(G)\downarrow = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $S \Rightarrow x_1\underline{S} \Rightarrow x_1x_2S$ $\qquad\qquad$ $x_1x_2Sy_1y_2$

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L\!\downarrow$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G)\!\downarrow$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L\!\downarrow$ for $S \to L\!\downarrow$):
  - $\mathrm{SF}(G)\!\downarrow = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $S \Rightarrow x_1 S \Rightarrow x_1 x_2 \underline{S} \Rightarrow x_1 x_2 S y_2 \qquad x_1 x_2 S y_1 y_2$

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G){\downarrow}$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L{\downarrow}$ for $S \to L{\downarrow}$):
  - $\mathrm{SF}(G){\downarrow} = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $S \Rightarrow x_1 S \Rightarrow x_1 x_2 S \Rightarrow x_1 x_2 \underline{S} y_2 \Rightarrow x_1 x_2 S y_1 y_2$

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G){\downarrow}$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L{\downarrow}$ for $S \to L{\downarrow}$):
  - $\mathrm{SF}(G){\downarrow} = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $S \Rightarrow x_1 S \Rightarrow x_1 x_2 S \Rightarrow x_1 x_2 S y_2 \Rightarrow x_1 x_2 S y_1 y_2$

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G){\downarrow}$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L{\downarrow}$ for $S \to L{\downarrow}$):
  - $\mathrm{SF}(G){\downarrow} = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $S \Rightarrow x_1 S \Rightarrow x_1 x_2 S \Rightarrow x_1 x_2 S y_2 \Rightarrow x_1 x_2 S y_1 y_2$
- If $\geqslant 2$ occurrences of S on some right-hand side:

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G){\downarrow}$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L{\downarrow}$ for $S \to L{\downarrow}$):
  - $\mathrm{SF}(G){\downarrow} = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $S \Rightarrow x_1 S \Rightarrow x_1 x_2 S \Rightarrow x_1 x_2 S y_2 \Rightarrow x_1 x_2 S y_1 y_2$
- If $\geqslant 2$ occurrences of S on some right-hand side:
  - $\mathrm{SF}(G){\downarrow} = (X \cup Y \cup \{S\})^*$

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G){\downarrow}$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L{\downarrow}$ for $S \to L{\downarrow}$):
  - $\mathrm{SF}(G){\downarrow} = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $S \Rightarrow x_1 S \Rightarrow x_1 x_2 S \Rightarrow x_1 x_2 S y_2 \Rightarrow x_1 x_2 S y_1 y_2$
- If $\geqslant 2$ occurrences of S on some right-hand side:
  - $\mathrm{SF}(G){\downarrow} = (X \cup Y \cup \{S\})^*$
  - Sx_1y_1

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G){\downarrow}$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L{\downarrow}$ for $S \to L{\downarrow}$):
  - $\mathrm{SF}(G){\downarrow} = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $S \Rightarrow x_1 S \Rightarrow x_1 x_2 S \Rightarrow x_1 x_2 S y_2 \Rightarrow x_1 x_2 S y_1 y_2$
- If $\geqslant 2$ occurrences of S on some right-hand side:
  - $\mathrm{SF}(G){\downarrow} = (X \cup Y \cup \{S\})^*$
  - $\underline{S} \Rightarrow SS$ $\qquad\qquad Sx_1y_1$

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L\downarrow$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G)\downarrow$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L\downarrow$ for $S \to L\downarrow$):
  - $\mathrm{SF}(G)\downarrow = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $S \Rightarrow x_1 S \Rightarrow x_1 x_2 S \Rightarrow x_1 x_2 S y_2 \Rightarrow x_1 x_2 S y_1 y_2$
- If $\geqslant 2$ occurrences of S on some right-hand side:
  - $\mathrm{SF}(G)\downarrow = (X \cup Y \cup \{S\})^*$
  - $S \Rightarrow S\underline{S} \Rightarrow Sx_1S$ $\qquad$ Sx_1y_1

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G){\downarrow}$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L{\downarrow}$ for $S \to L{\downarrow}$):
  - $\mathrm{SF}(G){\downarrow} = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $S \Rightarrow x_1 S \Rightarrow x_1 x_2 S \Rightarrow x_1 x_2 S y_2 \Rightarrow x_1 x_2 S y_1 y_2$
- If $\geqslant 2$ occurrences of S on some right-hand side:
  - $\mathrm{SF}(G){\downarrow} = (X \cup Y \cup \{S\})^*$
  - $S \Rightarrow SS \Rightarrow Sx_1\underline{S} \Rightarrow Sx_1SS$ $\qquad Sx_1y_1$

van Leeuwen's Algorithm

- Replace each production $A \to L$ with $A \to L\!\downarrow$.

Case 1

Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G)\!\downarrow$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L\!\downarrow$ for $S \to L\!\downarrow$):
  - $\mathrm{SF}(G)\!\downarrow = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $S \Rightarrow x_1 S \Rightarrow x_1 x_2 S \Rightarrow x_1 x_2 S y_2 \Rightarrow x_1 x_2 S y_1 y_2$
- If $\geqslant 2$ occurrences of S on some right-hand side:
  - $\mathrm{SF}(G)\!\downarrow = (X \cup Y \cup \{S\})^*$
  - $S \Rightarrow SS \Rightarrow Sx_1S \Rightarrow Sx_1\underline{S}S \Rightarrow Sx_1Sy_1S \succcurlyeq Sx_1y_1$

van Leeuwen's Algorithm

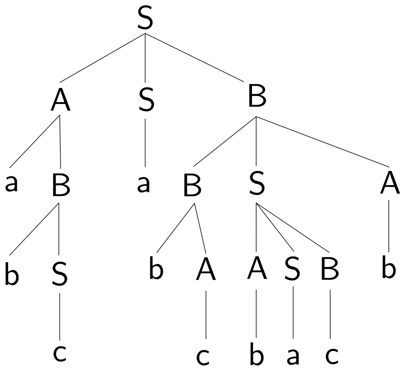
- Replace each production $A \to L$ with $A \to L{\downarrow}$.

Case 1

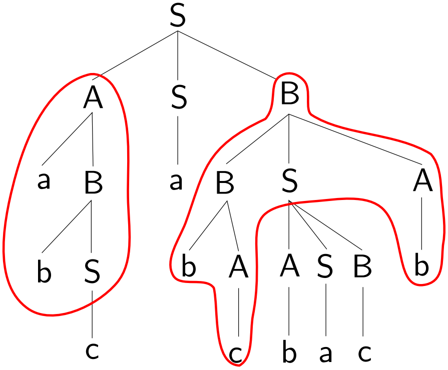
Suppose there is only one nonterminal S.

- Wlog: S occurs on every right-hand side; compute $\mathrm{SF}(G){\downarrow}$.
- If $\leqslant 1$ occurrences of S on right-hand sides ($SS \notin L{\downarrow}$ for $S \to L{\downarrow}$):
  - $\mathrm{SF}(G){\downarrow} = X^*\{S, \varepsilon\}Y^*$;
    X: occur to the left of S; Y: occur to the right of S
  - $S \Rightarrow x_1 S \Rightarrow x_1 x_2 S \Rightarrow x_1 x_2 S y_2 \Rightarrow x_1 x_2 S y_1 y_2$
- If $\geqslant 2$ occurrences of S on some right-hand side:
  - $\mathrm{SF}(G){\downarrow} = (X \cup Y \cup \{S\})^*$
  - $S \Rightarrow SS \Rightarrow Sx_1S \Rightarrow Sx_1SS \Rightarrow Sx_1Sy_1S \succcurlyeq Sx_1y_1$

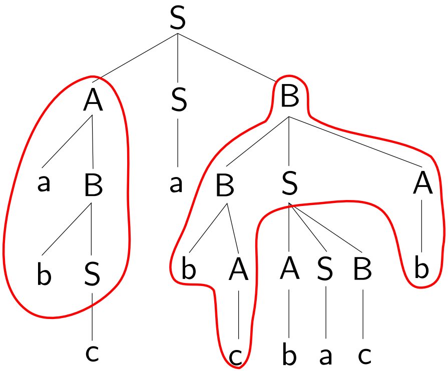
van Leeuwen's Algorithm



van Leeuwen's Algorithm



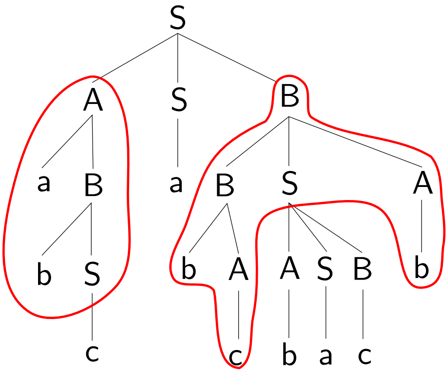
van Leeuwen's Algorithm



Algorithm

- For nonterminals $A \neq S$, construct grammar G_A:
  - Start symbol A
  - S is now terminal

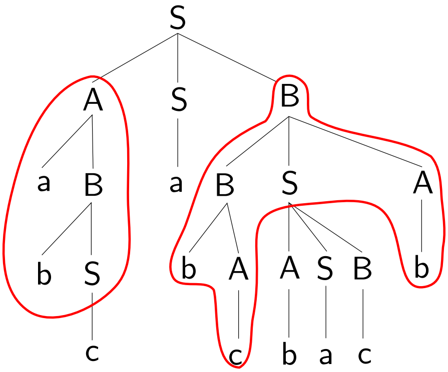
van Leeuwen's Algorithm



Algorithm

- For nonterminals $A \neq S$, construct grammar G_A:
  - Start symbol A
  - S is now terminal
- G_A has fewer nonterminals

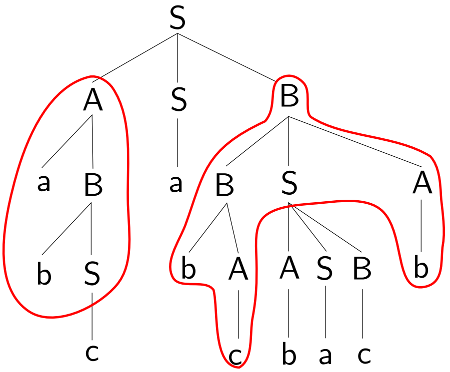
van Leeuwen's Algorithm



Algorithm

- For nonterminals $A \neq S$, construct grammar G_A:
  - Start symbol A
  - S is now terminal
- G_A has fewer nonterminals
- Compute $\mathsf{L}(G_A)\downarrow$

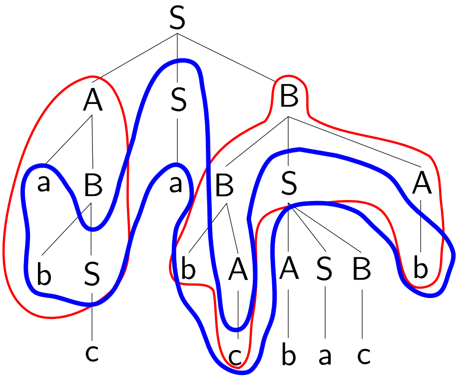
van Leeuwen's Algorithm



Algorithm

- For nonterminals $A \neq S$, construct grammar G_A:
  - Start symbol A
  - S is now terminal
- G_A has fewer nonterminals
- Compute $\mathrm{L}(G_A)\!\downarrow$
- In each $S \to L$, replace each A by $\mathrm{L}(G_A)\!\downarrow$

van Leeuwen's Algorithm

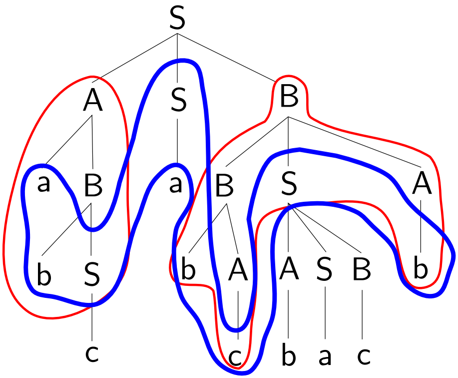


S → abSSbcSb

Algorithm

- For nonterminals $A \neq S$, construct grammar G_A:
  - Start symbol A
  - S is now terminal
- G_A has fewer nonterminals
- Compute $\mathrm{L}(G_A)\downarrow$
- In each $S \to L$, replace each A by $\mathrm{L}(G_A)\downarrow$

van Leeuwen's Algorithm



S → abSSbcSb

Algorithm

- For nonterminals $A \neq S$, construct grammar G_A:
  - Start symbol A
  - S is now terminal
- G_A has fewer nonterminals
- Compute $\mathrm{L}(G_A)\!\downarrow$
- In each $S \rightarrow L$, replace each A by $\mathrm{L}(G_A)\!\downarrow$
- Resulting grammar has one nonterminal

Ingredient II

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

Problem

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

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

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

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 $L'{\downarrow} = L{\downarrow}$.

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

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

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

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

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

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

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.

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

- μ_i: constant vector
- F_i: set of period vectors

Example

$$L = (ab)^*(ca^* \cup db^*)$$

Parikh image: $c + \{a+b, a\}^{\oplus} \quad \cup \quad d + \{a+b, b\}^{\oplus}$.

Example

$$L = (ab)^*(ca^* \cup db^*)$$

Parikh image: $\underset{\alpha}{\underset{\uparrow}{c}} + \{a \underset{\mu}{\underset{\uparrow}{+}} b, \underset{\nu}{\underset{\uparrow}{a}}\}^{\oplus} \quad \cup \quad \underset{\beta}{\underset{\uparrow}{d}} + \{a \underset{\sigma}{\underset{\uparrow}{+}} b, \underset{\tau}{\underset{\uparrow}{b}}\}^{\oplus}$.

$$\begin{aligned} C &= \{\alpha, \beta\}, \\ P &= \{\mu, \nu, \sigma, \tau\}, \\ P_\alpha &= \{\mu, \nu\}, \\ P_\beta &= \{\sigma, \tau\}, \end{aligned}$$

Example

$$L = (ab)^*(ca^* \cup db^*)$$

Parikh image: $\underset{\alpha}{\underset{\uparrow}{c}} + \{\underset{\mu}{a \underset{\uparrow}{+} b}, \underset{\nu}{\underset{\uparrow}{a}}\}^{\oplus} \quad \cup \quad \underset{\beta}{\underset{\uparrow}{d}} + \{\underset{\sigma}{a \underset{\uparrow}{+} b}, \underset{\tau}{\underset{\uparrow}{b}}\}^{\oplus}$.

$$\begin{aligned} C &= \{\alpha, \beta\}, & & & & \\ P &= \{\mu, \nu, \sigma, \tau\}, & \varphi(\alpha) &= c, & \varphi(\beta) &= d, \\ P_\alpha &= \{\mu, \nu\}, & \varphi(\mu) &= a + b, & \varphi(\nu) &= a, \\ P_\beta &= \{\sigma, \tau\}, & \varphi(\sigma) &= a + b, & \varphi(\tau) &= b, \end{aligned}$$

Example

$$L = (ab)^*(ca^* \cup db^*)$$

Parikh image: $\underset{\alpha}{\underset{\uparrow}{c}} + \{a \underset{\mu}{\underset{\uparrow}{+}} b, \underset{\nu}{\underset{\uparrow}{a}}\}^{\oplus} \quad \cup \quad \underset{\beta}{\underset{\uparrow}{d}} + \{a \underset{\sigma}{\underset{\uparrow}{+}} b, \underset{\tau}{\underset{\uparrow}{b}}\}^{\oplus}$.

$$\begin{aligned} C &= \{\alpha, \beta\}, \\ P &= \{\mu, \nu, \sigma, \tau\}, & \varphi(\alpha) &= c, & \varphi(\beta) &= d, \\ P_\alpha &= \{\mu, \nu\}, & \varphi(\mu) &= a + b, & \varphi(\nu) &= a, \\ P_\beta &= \{\sigma, \tau\}, & \varphi(\sigma) &= a + b, & \varphi(\tau) &= b, \end{aligned}$$

Example

$$L = (ab)^*(ca^* \cup db^*)$$

Parikh image: $\underset{\alpha}{\overset{\uparrow}{c}} + \{a \underset{\mu}{\overset{\uparrow}{+}} b, \underset{\nu}{\overset{\uparrow}{a}}\}^{\oplus} \qquad \cup \qquad \underset{\beta}{\overset{\uparrow}{d}} + \{a \underset{\sigma}{\overset{\uparrow}{+}} b, \underset{\tau}{\overset{\uparrow}{b}}\}^{\oplus}$.

$$\begin{aligned} C &= \{\alpha, \beta\}, & & & & \\ P &= \{\mu, \nu, \sigma, \tau\}, & \varphi(\alpha) &= c, & \varphi(\beta) &= d, \\ P_\alpha &= \{\mu, \nu\}, & \varphi(\mu) &= a + b, & \varphi(\nu) &= a, \\ P_\beta &= \{\sigma, \tau\}, & \varphi(\sigma) &= a + b, & \varphi(\tau) &= b, \end{aligned}$$

Example

$$L = (ab)^*(ca^* \cup db^*)$$

Parikh image: $\underset{\alpha}{\underset{\uparrow}{c}} + \{\underset{\mu}{a \underset{\uparrow}{+} b}, \underset{\nu}{\underset{\uparrow}{a}}\}^{\oplus} \quad \cup \quad \underset{\beta}{\underset{\uparrow}{d}} + \{\underset{\sigma}{a \underset{\uparrow}{+} b}, \underset{\tau}{\underset{\uparrow}{b}}\}^{\oplus}$.

$$\begin{aligned}
C &= \{\alpha, \beta\}, & & \\
P &= \{\mu, \nu, \sigma, \tau\}, & \varphi(\alpha) &= c, & \varphi(\beta) &= d, \\
P_\alpha &= \{\mu, \nu\}, & \varphi(\mu) &= a + b, & \varphi(\nu) &= a, \\
P_\beta &= \{\sigma, \tau\}, & \varphi(\sigma) &= a + b, & \varphi(\tau) &= b,
\end{aligned}$$

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

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 at designated positions

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 at designated positions

Definition (Parikh annotation)

Let $\mathcal{C}$ be a language class and $L \in \mathcal{C}$, $L \subseteq X^*$. A Parikh annotation for L in $\mathcal{C}$ is a tuple $(K, C, P, (P_c)_{c \in C}, \varphi)$

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 at designated positions

Definition (Parikh annotation)

Let $\mathcal{C}$ be a language class and $L \in \mathcal{C}$, $L \subseteq X^*$. A Parikh annotation for L in $\mathcal{C}$ is a tuple $(K, C, P, (P_c)_{c \in C}, \varphi)$, where

- $K \subseteq C(X \cup P)^*$ is in $\mathcal{C}$,

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 at designated positions

Definition (Parikh annotation)

Let $\mathcal{C}$ be a language class and $L \in \mathcal{C}$, $L \subseteq X^*$. A Parikh annotation for L in $\mathcal{C}$ is a tuple $(K, C, P, (P_c)_{c \in C}, \varphi)$, where

- $K \subseteq C(X \cup P)^*$ is in $\mathcal{C}$,
- $\pi_X(K) = L$,

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 at designated positions

Definition (Parikh annotation)

Let $\mathcal{C}$ be a language class and $L \in \mathcal{C}$, $L \subseteq X^*$. A Parikh annotation for L in $\mathcal{C}$ is a tuple $(K, C, P, (P_c)_{c \in C}, \varphi)$, where

- $K \subseteq C(X \cup P)^*$ is in $\mathcal{C}$,
- $\pi_X(K) = L$,
- $\Psi(\pi_X(w)) = \varphi(\pi_{C \cup P}(w))$ for each $w \in K$,

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 at designated positions

Definition (Parikh annotation)

Let $\mathcal{C}$ be a language class and $L \in \mathcal{C}$, $L \subseteq X^*$. A Parikh annotation for L in $\mathcal{C}$ is a tuple $(K, C, P, (P_c)_{c\in C}, \varphi)$, where

- $K \subseteq C(X \cup P)^*$ is in $\mathcal{C}$,
- $\pi_X(K) = L$,
- $\Psi(\pi_X(w)) = \varphi(\pi_{C\cup P}(w))$ for each $w \in K$,
- $\Psi(\pi_{C\cup P}(K)) = \bigcup_{c\in C} c + P_c^{\oplus}$.

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 at designated positions

Definition (Parikh annotation)

Let $\mathcal{C}$ be a language class and $L \in \mathcal{C}$, $L \subseteq X^*$. A Parikh annotation for L in $\mathcal{C}$ is a tuple $(K, C, P, (P_c)_{c \in C}, \varphi)$, where

- $K \subseteq C(X \cup P)^*$ is in $\mathcal{C}$,
- $\pi_X(K) = L$,
- $\Psi(\pi_X(w)) = \varphi(\pi_{C \cup P}(w))$ for each $w \in K$,
- $\Psi(\pi_{C \cup P}(K)) = \bigcup_{c \in C} c + P_c^{\oplus}$.
- For $cw \in K$ and $\kappa \in P_c^{\oplus}$

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 at designated positions

Definition (Parikh annotation)

Let $\mathcal{C}$ be a language class and $L \in \mathcal{C}$, $L \subseteq X^*$. A Parikh annotation for L in $\mathcal{C}$ is a tuple $(K, C, P, (P_c)_{c \in C}, \varphi)$, where

- $K \subseteq C(X \cup P)^*$ is in $\mathcal{C}$,
- $\pi_X(K) = L$,
- $\Psi(\pi_X(w)) = \varphi(\pi_{C \cup P}(w))$ for each $w \in K$,
- $\Psi(\pi_{C \cup P}(K)) = \bigcup_{c \in C} c + P_c^{\oplus}$.
- For $cw \in K$ and $\kappa \in P_c^{\oplus}$, there is a $v \in L$ with

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 at designated positions

Definition (Parikh annotation)

Let $\mathcal{C}$ be a language class and $L \in \mathcal{C}$, $L \subseteq X^*$. A Parikh annotation for L in $\mathcal{C}$ is a tuple $(K, C, P, (P_c)_{c \in C}, \varphi)$, where

- $K \subseteq C(X \cup P)^*$ is in $\mathcal{C}$,
- $\pi_X(K) = L$,
- $\Psi(\pi_X(w)) = \varphi(\pi_{C \cup P}(w))$ for each $w \in K$,
- $\Psi(\pi_{C \cup P}(K)) = \bigcup_{c \in C} c + P_c^{\oplus}$.
- For $cw \in K$ and $\kappa \in P_c^{\oplus}$, there is a $v \in L$ with

$$\Psi(v) = \Psi(\pi_X(cw)) + \varphi(\kappa)$$

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 at designated positions

Definition (Parikh annotation)

Let $\mathcal{C}$ be a language class and $L \in \mathcal{C}$, $L \subseteq X^*$. A Parikh annotation for L in $\mathcal{C}$ is a tuple $(K, C, P, (P_c)_{c \in C}, \varphi)$, where

- $K \subseteq C(X \cup P)^*$ is in $\mathcal{C}$,
- $\pi_X(K) = L$,
- $\Psi(\pi_X(w)) = \varphi(\pi_{C \cup P}(w))$ for each $w \in K$,
- $\Psi(\pi_{C \cup P}(K)) = \bigcup_{c \in C} c + P_c^{\oplus}$.
- For $cw \in K$ and $\kappa \in P_c^{\oplus}$, there is a $v \in L$ with

$$\Psi(v) = \Psi(\pi_X(cw)) + \varphi(\kappa), \qquad \pi_X(cw) \preccurlyeq v.$$

Theorem

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

- Refinement of Parikh's theorem

Theorem

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

- Refinement of Parikh's theorem
- Direct construction for $\mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{G}_i)$

Theorem

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

- Refinement of Parikh's theorem
- Direct construction for $\mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{G}_i)$
- Series of steps for $\mathsf{G}_i = \mathsf{Alg}(\mathsf{F}_i)$
  - Decomposition similar to van Leeuwen's algorithm
  - Most involved step: substitute a by $\{a, b\}$
  - Replace annotation symbols and ordinary symbols consistently

Theorem

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

- Refinement of Parikh's theorem
- Direct construction for $\mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{G}_i)$
- Series of steps for $\mathsf{G}_i = \mathsf{Alg}(\mathsf{F}_i)$
  - Decomposition similar to van Leeuwen's algorithm
  - Most involved step: substitute a by $\{a, b\}$
  - Replace annotation symbols and ordinary symbols consistently

A morphism $\psi\colon (N \cup T)^* \to \mathbb{Z}$ is *G-compatible* if $A \Rightarrow^* w$ implies $\psi(A) = \psi(w)$, for $A \in N$, $w \in T^*$.

Theorem

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

- Refinement of Parikh's theorem
- Direct construction for $\mathsf{F}_{i+1} = \mathsf{SLI}(\mathsf{G}_i)$
- Series of steps for $\mathsf{G}_i = \mathsf{Alg}(\mathsf{F}_i)$
  - Decomposition similar to van Leeuwen's algorithm
  - Most involved step: substitute a by $\{a, b\}$
  - Replace annotation symbols and ordinary symbols consistently

A morphism $\psi\colon (N \cup T)^* \to \mathbb{Z}$ is *G-compatible* if $A \Rightarrow^* w$ implies $\psi(A) = \psi(w)$, for $A \in N$, $w \in T^*$.

Lemma

Let G be a reduced $\mathcal{C}$-grammar and $\psi\colon T^* \to \mathbb{Z}$ a morphism such that $\psi(w) = 0$ for every $w \in L(G)$. Then ψ extends uniquely to a G-compatible morphism $\psi\colon (N \cup T)^* \to \mathbb{Z}$.

Using Parikh annotations, one can show:

Corollary

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

Using Parikh annotations, one can show:

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

Using Parikh annotations, one can show:

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

Using Parikh annotations, one can show:

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
- 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!

Why not $K \subseteq CX^*P^*$?
Then there would be no Parikh annotations for context-free languages!

Why not $K \subseteq CX^*P^*$?

Then there would be no Parikh annotations for context-free languages!

- Suppose $L = \{a^n b^n \mid n \geqslant 0\}$. Then $L \in \mathrm{CF}$.

Why not $K \subseteq CX^*P^*$?

Then there would be no Parikh annotations for context-free languages!

- Suppose $L = \{a^n b^n \mid n \geqslant 0\}$. Then $L \in \mathrm{CF}$.
- If L has a PA with $K \subseteq CX^*P^*$ in CF, then there is one with $K \subseteq X^*P^*C$.

Why not $K \subseteq CX^*P^*$?
Then there would be no Parikh annotations for context-free languages!

- Suppose $L = \{a^n b^n \mid n \geqslant 0\}$. Then $L \in \mathrm{CF}$.
- If L has a PA with $K \subseteq CX^*P^*$ in CF, then there is one with $K \subseteq X^*P^*C$.
- Let L' be obtained from K by replacing every $x \in C \cup P$ by $a^{\varphi(x)(a)}$.

Why not $K \subseteq CX^*P^*$?

Then there would be no Parikh annotations for context-free languages!

- Suppose $L = \{a^n b^n \mid n \geqslant 0\}$. Then $L \in \mathrm{CF}$.
- If L has a PA with $K \subseteq CX^*P^*$ in CF, then there is one with $K \subseteq X^*P^*C$.
- Let L' be obtained from K by replacing every $x \in C \cup P$ by $a^{\varphi(x)(a)}$.
- Then $L' = \{a^n b^n a^n \mid n \geqslant 0\}$, which is not context-free.