Leftist Grammars

Tomasz Jurdziński

Institute of Computer Science
University of Wrocław
Poland

ABCD Workshop 2009, Prague

Outline

1. Motivations and definitions
2. Accessibility vs Grammars
3. Complexity of General Leftist Grammars
4. Restricted leftist grammars
5. Alphabet size restriction

Outline

1. Motivations and definitions
2. Accessibility vs Grammars
3. Complexity of General Leftist Grammars
4. Restricted leftist grammars
5. Alphabet size restriction

Accessibility Problems

Protection system:

- Objects: users, processes, agents, etc.

Accessibility Problems

Protection system:

- Objects: users, processes, agents, etc.
- Interactions: access rights, information sharing privileges, etc.

Accessibility Problems

Protection system:

- Objects: users, processes, agents, etc.
- Interactions: access rights, information sharing privileges, etc.
- Policies: rules, which determine the ways in which objects can get access to other objects.

Accessibility Problems

Protection system:

- Objects: users, processes, agents, etc.
- Interactions: access rights, information sharing privileges, etc.
- Policies: rules, which determine the ways in which objects can get access to other objects.

Accessibility problem

Can object p gain (illegal) access to object q by a series of legal moves?

Models for Accessibility Problems

1. access-matrix model: two-dimensional table of rights/priviliges; general rules for operations (undecidable);
2. grammatical protection systems: without creation of new objects;
3. take-grant model (limited expressive power).

The Model for Java virtual worlds (Cheiner, Saraswat)

The system described by:

- a directed graph $G(V, E)$: V represents objects, E access rights;

The Model for Java virtual worlds (Cheiner, Saraswat)

The system described by:

- a directed graph $G(V, E)$: V represents objects, E access rights;
- a type mapping $t : V \to T$: each vertex (object) has some label (type);

The Model for Java virtual worlds (Cheiner, Saraswat)

The system described by:

- a directed graph $G(V, E)$: V represents objects, E access rights;
- a type mapping $t : V \rightarrow T$: each vertex (object) has some label (type);
- binary relations $R_i, R_e \subseteq T \times T$: determine which operations are allowed.

The Cheiner/Saraswat Model

Operations

- Insert(v, x): inserts a new vertex x of any type and an edge (v, x).

The Cheiner/Saraswat Model

Operations

- Insert(v, x): inserts a new vertex x of any type and an edge (v, x).
- Give(a, b, c):
  inserts an edge (b, c), provided $(a, b), (a, c) \in E$ and $(t(b), t(c)) \in R_i$.

The Cheiner/Saraswat Model

Operations

- Insert(v, x): inserts a new vertex x of any type and an edge (v, x).
- Give(a, b, c):
  inserts an edge (b, c), provided $(a, b), (a, c) \in E$ and $(t(b), t(c)) \in R_i$.
- Get(a, b, c):
  inserts an edge (a, c), provided $(a, b), (b, c) \in E$ and $(t(b), t(c)) \in R_e$.

The Cheiner/Saraswat Model

Operations

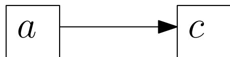
- Insert(v, x): inserts a new vertex x of any type and an edge (v, x).
- Give(a, b, c):
  inserts an edge (b, c), provided $(a, b), (a, c) \in E$ and $(t(b), t(c)) \in R_i$.
- Get(a, b, c):
  inserts an edge (a, c), provided $(a, b), (b, c) \in E$ and $(t(b), t(c)) \in R_e$.

Restricted set of operations:

- Insert(a, x, c): combines original operations Insert and Give;
- Get(a, b, c): as before.

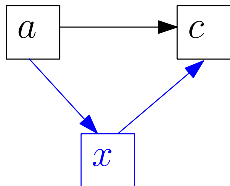
Operations (restricted)

$\mathrm{Insert}(a, x, c)$



Operations (restricted)

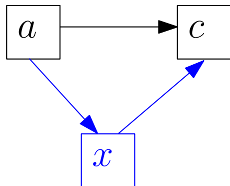
$\text{Insert}(a, x, c)$



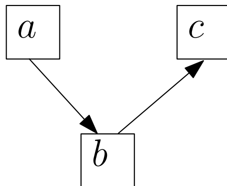
If $(t(x), t(c)) \in R_i$

Operations (restricted)

Insert(a, x, c)



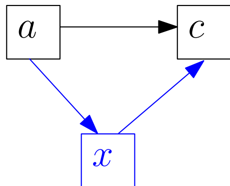
Get(a, b, c)



If $(t(x), t(c)) \in R_i$

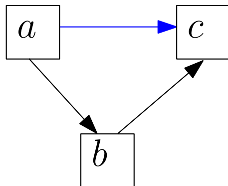
Operations (restricted)

Insert(a, x, c)



If $(t(x), t(c)) \in R_i$

Get(a, b, c)



If $(t(b), t(c)) \in R_e$

Accessibility in the Cheiner/Saraswat Model

Accessibility Problem

Input:

- a directed graph $G(V, E)$;
- a type mapping $t : V \to T$;
- $R_i, R_e \subseteq T \times T$.
- vertices $p, q \in V$.

Accessibility in the Cheiner/Saraswat Model

Accessibility Problem

Input:

- a directed graph $G(V, E)$;
- a type mapping $t : V \to T$;
- $R_i, R_e \subseteq T \times T$.
- vertices $p, q \in V$.

Output:

YES iff there exists a sequence S of operations Insert, Get, Give such that the graph obtained from G after applying S contains the edge (p, q).

Leftist Grammars (Motwani et al.)

A grammar $\mathcal{G} = (\Sigma, P, x)$ consists of:

Leftist Grammars (Motwani et al.)

A grammar $\mathcal{G} = (\Sigma, P, x)$ consists of:

- a finite alphabet Σ,

Leftist Grammars (Motwani et al.)

A grammar $\mathcal{G} = (\Sigma, P, x)$ consists of:

- a finite alphabet Σ,
- a final symbol $x \in \Sigma$,

Leftist Grammars (Motwani et al.)

A grammar $\mathcal{G} = (\Sigma, P, x)$ consists of:

- a finite alphabet Σ,
- a final symbol $x \in \Sigma$,
- production rules P of the following two types,

$$
\begin{array}{rcll}
ab & \to & b & \text{(Delete Rule, corresponds to Get)} \\
c & \to & dc & \text{(Insert Rule, corresponds to Insert)}
\end{array}
$$

where $a, b, c, d \in \Sigma$.

Leftist Grammars (Motwani et al.)

A grammar $\mathcal{G} = (\Sigma, P, x)$ consists of:

- a finite alphabet Σ,
- a final symbol $x \in \Sigma$,
- production rules P of the following two types,

$$\begin{array}{rcll} ab & \rightarrow & b & \text{(Delete Rule, corresponds to Get)} \\ c & \rightarrow & dc & \text{(Insert Rule, corresponds to Insert)} \end{array}$$

where $a, b, c, d \in \Sigma$.

A word $w \in \Sigma^*$ belongs to the language $L(\mathcal{G})$ iff $wx \Rightarrow^* x$.

Notations and Definitions

Shorter description

$$\begin{array}{lcllccl} a & \to & ba & \text{equivalent to} & a & \xrightarrow{\text{ins}} & b \\ ba & \to & a & \text{equivalent to} & a & \xrightarrow{\text{del}} & b. \end{array}$$

Notations and Definitions

Shorter description

$$\begin{array}{cccccc} a & \to & ba & \text{equivalent to} & a \stackrel{\text{ins}}{\to} b \\ ba & \to & a & \text{equivalent to} & a \stackrel{\text{del}}{\to} b. \end{array}$$

Active symbol

Let

$$\begin{array}{l} u\ a\ v \Rightarrow u\ ba\ v \quad \text{or} \\ u\ ba\ v \Rightarrow u\ a\ v \end{array}$$

be a derivation step which applies a production $a \to ba$ (or $ba \to a$, resp.). Then, a is **active** in this step.

Insert/delete graph

Insert Graph for a grammar (Σ, P, x):

- vertices: elements of Σ;

Insert/delete graph

Insert Graph for a grammar (Σ, P, x):

- vertices: elements of Σ;
- edges: (a, b) in the graph iff P contains

$$a \overset{\text{ins}}{\longrightarrow} b$$

Insert/delete graph

Insert Graph for a grammar (Σ, P, x):

- vertices: elements of Σ;
- edges: (a, b) in the graph iff P contains

$$a \overset{\text{ins}}{\longrightarrow} b$$

Delete Graph for a grammar (Σ, P, x):

- vertices: elements of Σ;

Insert/delete graph

Insert Graph for a grammar (Σ, P, x):

- vertices: elements of Σ;
- edges: (a, b) in the graph iff P contains

$$a \overset{\text{ins}}{\longrightarrow} b$$

Delete Graph for a grammar (Σ, P, x):

- vertices: elements of Σ;
- edges: (a, b) in the graph iff P contains

$$a \overset{\text{del}}{\longrightarrow} b$$

Example

Alphabet

$\Sigma = \{a_0, a_1, b_0, b_1, x\}$

Example

Alphabet

$\Sigma = \{a_0, a_1, b_0, b_1, x\}$

Production rules

$b_1 \xrightarrow{\text{del}} a_1$

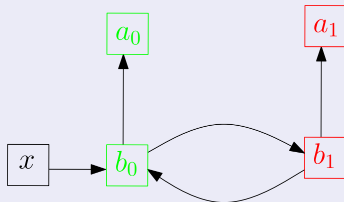
$b_0 \xrightarrow{\text{del}} a_0$

$b_1 \xrightarrow{\text{del}} b_0$

$b_0 \xrightarrow{\text{del}} b_1$

$x \xrightarrow{\text{del}} b_0$

Delete graph



Example

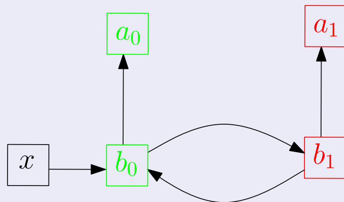
Alphabet

$\Sigma = \{a_0, a_1, b_0, b_1, x\}$

Production rules

$$b_1 \xrightarrow{\text{del}} a_1$$
$$b_0 \xrightarrow{\text{del}} a_0$$
$$b_1 \xrightarrow{\text{del}} b_0$$
$$b_0 \xrightarrow{\text{del}} b_1$$
$$x \xrightarrow{\text{del}} b_0$$

Delete graph



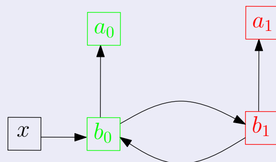
The language

$L(\mathcal{G}) \cap (a_0 a_1)^*(b_1 b_0)^* = \{(a_0 a_1)^n (b_1 b_0)^m \mid m \geq n\}$

Example cont.

a_0 a_1 a_0 $\underline{a_1}$ $\underline{b_1}$ b_0 b_1 b_0 x

Delete graph



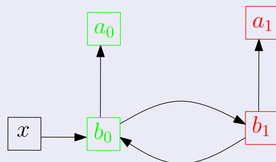
Example cont.

$a_0 \quad a_1 \quad a_0 \quad \underline{a_1} \quad \underline{b_1} \quad b_0 \quad b_1 \quad b_0 \quad x$

$\Downarrow$

$a_0 \quad a_1 \quad a_0 \quad \underline{b_1} \quad \underline{b_0} \quad b_1 \quad b_0 \quad x$

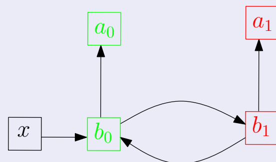
Delete graph



Example cont.

$$a_0 \quad a_1 \quad a_0 \quad \underline{a_1} \quad \underline{b_1} \quad b_0 \quad b_1 \quad b_0 \quad x$$
$$\Downarrow$$
$$a_0 \quad a_1 \quad a_0 \quad \underline{b_1} \quad \underline{b_0} \quad b_1 \quad b_0 \quad x$$
$$\Downarrow$$
$$a_0 \quad a_1 \quad \underline{a_0} \quad \underline{b_0} \quad b_1 \quad b_0 \quad x$$

Delete graph



Example cont.

$a_0 \quad a_1 \quad a_0 \quad \underline{a_1} \quad \underline{b_1} \quad b_0 \quad b_1 \quad b_0 \quad x$

$\Downarrow$

$a_0 \quad a_1 \quad a_0 \quad \underline{b_1} \quad \underline{b_0} \quad b_1 \quad b_0 \quad x$

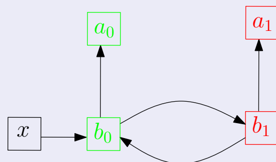
$\Downarrow$

$a_0 \quad a_1 \quad \underline{a_0} \quad \underline{b_0} \quad b_1 \quad b_0 \quad x$

$\Downarrow$

$a_0 \quad a_1 \quad \underline{b_0} \quad \underline{b_1} \quad b_0 \quad x$

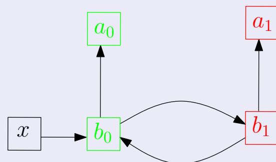
Delete graph



Example cont.

$$
\begin{array}{ccccccccc}
a_0 & a_1 & a_0 & \underline{a_1} & \underline{b_1} & b_0 & b_1 & b_0 & x \\
 & & & \Downarrow & & & & & \\
a_0 & a_1 & a_0 & & \underline{b_1} & \underline{b_0} & b_1 & b_0 & x \\
 & & & \Downarrow & & & & & \\
a_0 & a_1 & \underline{a_0} & & & \underline{b_0} & b_1 & b_0 & x \\
 & & & \Downarrow & & & & & \\
a_0 & a_1 & & & & \underline{b_0} & \underline{b_1} & b_0 & x \\
 & & & \Downarrow & & & & & \\
a_0 & \underline{a_1} & & & & & \underline{b_1} & b_0 & x
\end{array}
$$

Delete graph



Example cont.

$$a_0 \quad a_1 \quad a_0 \quad \underline{a_1} \quad \underline{b_1} \quad b_0 \quad b_1 \quad b_0 \quad x$$

$$\Downarrow$$

$$a_0 \quad a_1 \quad a_0 \quad \underline{b_1} \quad \underline{b_0} \quad b_1 \quad b_0 \quad x$$

$$\Downarrow$$

$$a_0 \quad a_1 \quad \underline{a_0} \quad \underline{b_0} \quad b_1 \quad b_0 \quad x$$

$$\Downarrow$$

$$a_0 \quad a_1 \quad \underline{b_0} \quad \underline{b_1} \quad b_0 \quad x$$

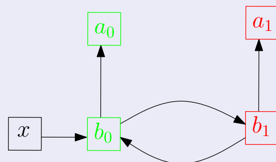
$$\Downarrow$$

$$a_0 \quad \underline{a_1} \quad \underline{b_1} \quad b_0 \quad x$$

$$\Downarrow$$

$$a_0 \quad \underline{b_1} \quad \underline{b_0} \quad x$$

Delete graph



Example cont.

$$a_0 \quad a_1 \quad a_0 \quad \underline{a_1} \quad \underline{b_1} \quad b_0 \quad b_1 \quad b_0 \quad x$$

$$\Downarrow$$

$$a_0 \quad a_1 \quad a_0 \quad \underline{b_1} \quad \underline{b_0} \quad b_1 \quad b_0 \quad x$$

$$\Downarrow$$

$$a_0 \quad a_1 \quad \underline{a_0} \quad \underline{b_0} \quad b_1 \quad b_0 \quad x$$

$$\Downarrow$$

$$a_0 \quad a_1 \quad \underline{b_0} \quad \underline{b_1} \quad b_0 \quad x$$

$$\Downarrow$$

$$a_0 \quad \underline{a_1} \quad \underline{b_1} \quad b_0 \quad x$$

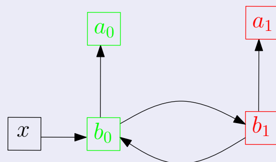
$$\Downarrow$$

$$a_0 \quad \underline{b_1} \quad \underline{b_0} \quad x$$

$$\Downarrow$$

$$\underline{a_0} \quad \underline{b_0} \quad x$$

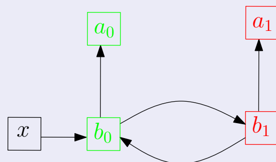
Delete graph



Example cont.

a_0	a_1	a_0	$\underline{a_1}$	$\underline{b_1}$	b_0	b_1	b_0	x
			$\Downarrow$					
a_0	a_1	a_0		$\underline{b_1}$	$\underline{b_0}$	b_1	b_0	x
			$\Downarrow$					
a_0	a_1	$\underline{a_0}$			$\underline{b_0}$	b_1	b_0	x
			$\Downarrow$					
a_0	a_1				$\underline{b_0}$	$\underline{b_1}$	b_0	x
			$\Downarrow$					
a_0	$\underline{a_1}$					$\underline{b_1}$	b_0	x
			$\Downarrow$					
a_0						$\underline{b_1}$	$\underline{b_0}$	x
			$\Downarrow$					
$\underline{a_0}$							$\underline{b_0}$	x
			$\Downarrow$					
							$\underline{b_0}$	$\underline{x}$

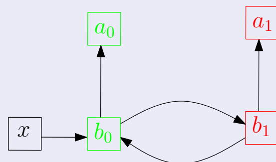
Delete graph



Example cont.

$$a_0 \quad a_1 \quad a_0 \quad \underline{a_1} \quad \underline{b_1} \quad b_0 \quad b_1 \quad b_0 \quad x$$
$$\Downarrow$$
$$a_0 \quad a_1 \quad a_0 \quad \underline{b_1} \quad \underline{b_0} \quad b_1 \quad b_0 \quad x$$
$$\Downarrow$$
$$a_0 \quad a_1 \quad \underline{a_0} \quad \underline{b_0} \quad b_1 \quad b_0 \quad x$$
$$\Downarrow$$
$$a_0 \quad a_1 \quad \underline{b_0} \quad \underline{b_1} \quad b_0 \quad x$$
$$\Downarrow$$
$$a_0 \quad \underline{a_1} \quad \underline{b_1} \quad b_0 \quad x$$
$$\Downarrow$$
$$a_0 \quad \underline{b_1} \quad \underline{b_0} \quad x$$
$$\Downarrow$$
$$\underline{a_0} \quad \underline{b_0} \quad x$$
$$\Downarrow$$
$$\underline{b_0} \quad \underline{x}$$
$$\Downarrow$$
$$x$$

Delete graph



Outline

1 Motivations and definitions

2 Accessibility vs Grammars

3 Complexity of General Leftist Grammars

4 Restricted leftist grammars

5 Alphabet size restriction

Accessibility Problem and Grammars

Intersection problem

Input: a (leftist) grammar $\mathcal{G}$ and a regular expression R.
Output: YES, iff $L(\mathcal{G}) \cap L(R) \neq \emptyset$.

Accessibility Problem and Grammars

Intersection problem

Input: a (leftist) grammar $\mathcal{G}$ and a regular expression R.
Output: YES, iff $L(\mathcal{G}) \cap L(R) \neq \emptyset$.

Variable membership problem

Input: a (leftist) grammar $\mathcal{G}$ and a word w.
Output: YES, iff $wx \Rightarrow_{\mathcal{G}}^{*} x$.

Accessibility Problem and Grammars

Intersection problem

Input: a (leftist) grammar $\mathcal{G}$ and a regular expression R.
Output: YES, iff $L(\mathcal{G}) \cap L(R) \neq \emptyset$.

Variable membership problem

Input: a (leftist) grammar $\mathcal{G}$ and a word w.
Output: YES, iff $wx \Rightarrow_{\mathcal{G}}^{*} x$.

Membership problem (a leftist grammar $\mathcal{G}$ is fixed)

Input: a word w.
Output: YES, iff $wx \Rightarrow_{\mathcal{G}}^{*} x$.

Accessibility, Grammars, ...

Theorem

The accessibility problem in the Cheiner/Saraswat model can be reduced in polynomial time to the intersection problem for leftist grammars.

Accessibility, Grammars, ...

Theorem

The accessibility problem in the Cheiner/Saraswat model can be reduced in polynomial time to the intersection problem for leftist grammars.

Theorem (MPSV00)

The intersection problem for leftist grammars is decidable.

Accessibility, Grammars, ...

Theorem

The accessibility problem in the Cheiner/Saraswat model can be reduced in polynomial time to the intersection problem for leftist grammars.

Theorem (MPSV00)

The intersection problem for leftist grammars is decidable.

Corollary

The membership problem for leftist grammars is decidable.

Outline

1. Motivations and definitions
2. Accessibility vs Grammars
3. Complexity of General Leftist Grammars
4. Restricted leftist grammars
5. Alphabet size restriction

Weakness of Leftist Grammars

Properties of leftist grammars

1. No closure under union, intersection, complementation.

Weakness of Leftist Grammars

Properties of leftist grammars

1. No closure under union, intersection, complementation.
2. "Limited closure under subsequences":

   if $xy \in L(\mathcal{G})$ and x deleted by y, then $x'y \in L(\mathcal{G})$ for each $x' \sqsubseteq x$.

Weakness of Leftist Grammars

Properties of leftist grammars

1. No closure under union, intersection, complementation.
2. "Limited closure under subsequences":

   if $xy \in L(\mathcal{G})$ and x deleted by y, then $x'y \in L(\mathcal{G})$ for each $x' \sqsubseteq x$.

3. Closure under stuttering:

   if $a_1 \cdots a_k \in L$, then $a_1^+ \cdots a_k^+ \subseteq L$ for $a_1, \ldots, a_k \in \Sigma$.

Weakness of Leftist Grammars

Properties of leftist grammars

1. No closure under union, intersection, complementation.
2. "Limited closure under subsequences":

$$\text{if } xy \in L(\mathcal{G}) \text{ and } x \text{ deleted by } y, \text{ then } x'y \in L(\mathcal{G}) \text{ for each } x' \sqsubseteq x.$$

3. Closure under stuttering:

$$\text{if } a_1 \cdots a_k \in L, \text{ then } a_1^+ \cdots a_k^+ \subseteq L \text{ for } a_1, \ldots, a_k \in \Sigma.$$

How to deal with stuttering?

For a word $w \in (\Sigma_0 \cup \Sigma_1)^*$, where $\Sigma_0 \cap \Sigma_1 = \emptyset$, measure the number of color alternations, instead of the number of symbols.

Complexity of Leftist Grammars

MPSV (STOC00)

Are all leftist languages context-free?

Complexity of Leftist Grammars

MPSV (STOC00)

Are all leftist languages context-free?

J., Loryś (FCT05)

Answer: No. But the witness language was simple (in LOGSPACE).

Complexity of Leftist Grammars

MPSV (STOC00)

Are all leftist languages context-free?

J., Loryś (FCT05)

Answer: No. But the witness language was simple (in LOGSPACE).

J. (ICALP06)

The membership problem for leftist grammars is PSPACE-hard.

Complexity of Leftist Grammars

MPSV (STOC00)

Are all leftist languages context-free?

J., Loryś (FCT05)

Answer: No. But the witness language was simple (in LOGSPACE).

J. (ICALP06)

The membership problem for leftist grammars is PSPACE-hard.

J. (ICALP08)

- variable membership problem: non-primitive recursive;
- ("static") membership problem: no primitive recursive upper bound.

Notations

Leftmost derivation

If

$$u \quad z \quad v \quad \Rightarrow \quad u \quad z' \quad v$$

is a derivation step of a leftmost derivation, then none of the symbols of u is active in the remaining part of the derivation.

Notations

Leftmost derivation

If

$$u \quad z \quad v \quad \Rightarrow \quad u \quad z' \quad v$$

is a derivation step of a leftmost derivation, then none of the symbols of u is active in the remaining part of the derivation.

Greedy derivation

- leftmost
- If $u\ z\ v \Rightarrow u\ z'\ v$ is a derivation step of a greedy derivation, then no delete production rule is applicable in u.

Notations

Leftmost derivation

If

$$u \quad z \quad v \quad \Rightarrow \quad u \quad z' \quad v$$

is a derivation step of a leftmost derivation, then none of the symbols of u is active in the remaining part of the derivation.

Greedy derivation

- leftmost
- If $u\ z\ v \Rightarrow u\ z'\,v$ is a derivation step of a greedy derivation, then no delete production rule is applicable in u.

Lemma

In order to verify wether $wx \Rightarrow^* x$, it is enough to consider greedy derivations.

Weak computation (simplified)

Categories of symbols

- input: I_0, I_1, disjoint, nonempty; $I = I_0 \cup I_1$;

Weak computation (simplified)

Categories of symbols

- input: I_0, I_1, disjoint, nonempty; $I = I_0 \cup I_1$;
- output: O_0, O_1, disjoint, nonempty; $O = O_0 \cup O_1$;

Weak computation (simplified)

Categories of symbols

- input: I_0, I_1, disjoint, nonempty; $I = I_0 \cup I_1$;
- output: O_0, O_1, disjoint, nonempty; $O = O_0 \cup O_1$;
- $g \notin I \cup O$.

Weak computation (simplified)

Categories of symbols

- input: I_0, I_1, disjoint, nonempty; $I = I_0 \cup I_1$;
- output: O_0, O_1, disjoint, nonempty; $O = O_0 \cup O_1$;
- $g \notin I \cup O$.

A grammar $\mathcal{G}$ weakly computes $\alpha : \mathbb{N} \to \mathbb{N}$ wrt I, O, g iff for each $w \in (I_1 I_0)^*$:

- there exists a derivation $wg \Rightarrow^* w'g$, where $w' \in (O_1 O_0)^*$ and $|w'| = \alpha(|w|)$;

Weak computation (simplified)

Categories of symbols

- input: I_0, I_1, disjoint, nonempty; $I = I_0 \cup I_1$;
- output: O_0, O_1, disjoint, nonempty; $O = O_0 \cup O_1$;
- $g \notin I \cup O$.

A grammar $\mathcal{G}$ weakly computes $\alpha : \mathbb{N} \to \mathbb{N}$ wrt I, O, g iff for each $w \in (I_1 I_0)^*$:

- there exists a derivation $wg \Rightarrow^* w'g$, where $w' \in (O_1 O_0)^*$ and $|w'| = \alpha(|w|)$;
- if $wg \Rightarrow^* w'g$, for $w' \in (O_0 \cup O_1)^*$, then $w' \in (O_1 O_0)^*$ and $|w'| \geq \alpha(|w|)$.

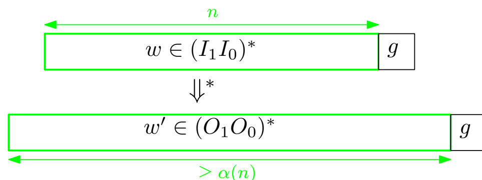
Weak computation (simplified)

Categories of symbols

- input: I_0, I_1, disjoint, nonempty; $I = I_0 \cup I_1$;
- output: O_0, O_1, disjoint, nonempty; $O = O_0 \cup O_1$;
- $g \notin I \cup O$.

A grammar $\mathcal{G}$ weakly computes $\alpha : \mathbb{N} \to \mathbb{N}$ wrt I, O, g iff for each $w \in (I_1 I_0)^*$:

- there exists a derivation $wg \Rightarrow^* w'g$, where $w' \in (O_1 O_0)^*$ and $|w'| = \alpha(|w|)$;
- if $wg \Rightarrow^* w'g$, for $w' \in (O_0 \cup O_1)^*$, then $w' \in (O_1 O_0)^*$ and $|w'| \geq \alpha(|w|)$.



Weak computation: example

A grammar for the function $\alpha(n) = 2n$:

- input symbols: $I_0 = \{a_0\}$, $I_1 = \{a_1\}$;
- output symbols: $O_0 = \{b_0, b_2\}$, $O_1 = \{b_1, b_3\}$

Weak computation: example

A grammar for the function $\alpha(n) = 2n$:

- input symbols: $I_0 = \{a_0\}$, $I_1 = \{a_1\}$;
- output symbols: $O_0 = \{b_0, b_2\}$, $O_1 = \{b_1, b_3\}$

Production rules

$$\begin{array}{ccl} b_i & \stackrel{\text{ins}}{\longrightarrow} & b_{(i+1) \bmod 4} \\ b_1 & \stackrel{\text{del}}{\longrightarrow} & a_0 \\ b_3 & \stackrel{\text{del}}{\longrightarrow} & a_1 \\ g & \stackrel{\text{ins}}{\longrightarrow} & b_0. \end{array}$$

Weak computation: example

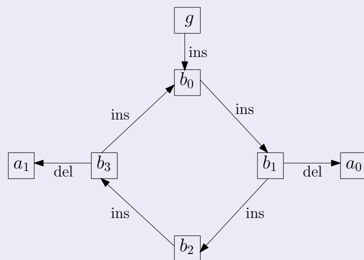
A grammar for the function $\alpha(n) = 2n$:

- input symbols: $I_0 = \{a_0\}$, $I_1 = \{a_1\}$;
- output symbols: $O_0 = \{b_0, b_2\}$, $O_1 = \{b_1, b_3\}$

Production rules

$$
\begin{array}{ccl}
b_i & \stackrel{\text{ins}}{\longrightarrow} & b_{(i+1) \bmod 4} \\
b_1 & \stackrel{\text{del}}{\longrightarrow} & a_0 \\
b_3 & \stackrel{\text{del}}{\longrightarrow} & a_1 \\
g & \stackrel{\text{ins}}{\longrightarrow} & b_0.
\end{array}
$$

Insert/delete graph

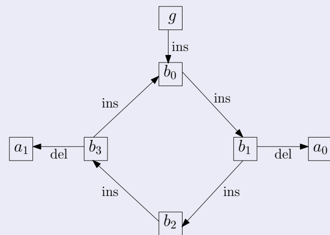


Example: $\alpha(n) = 2n$

$a_1 \quad a_0 \quad a_1 \quad a_0 \qquad \underline{g}$

$\Downarrow^*$

Insert/delete graph



Example: $\alpha(n) = 2n$

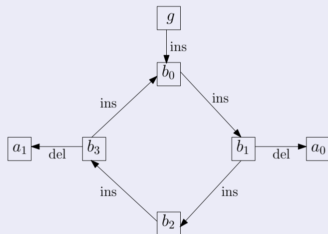
$$a_1 \quad a_0 \quad a_1 \quad a_0 \qquad\qquad \underline{g}$$

$$\Downarrow^*$$

$$a_1 \quad a_0 \quad a_1 \quad \underline{a_0} \qquad\qquad \underline{b_1} b_0 \quad g$$

$$\Downarrow$$

Insert/delete graph



Example: $\alpha(n) = 2n$

$$a_1 \quad a_0 \quad a_1 \quad a_0 \qquad\qquad\qquad\qquad \underline{g}$$

$$\Downarrow^*$$

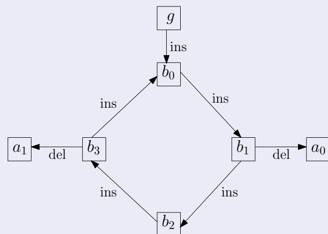
$$a_1 \quad a_0 \quad a_1 \quad \underline{a_0} \qquad\qquad \underline{b_1} b_0 \quad g$$

$$\Downarrow$$

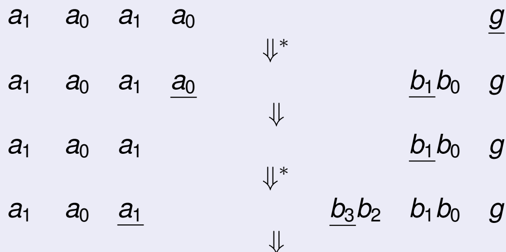
$$a_1 \quad a_0 \quad a_1 \qquad\qquad\qquad \underline{b_1} b_0 \quad g$$

$$\Downarrow^*$$

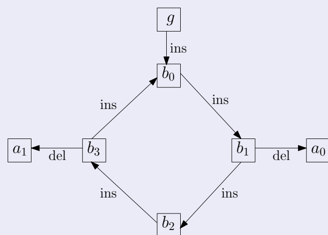
Insert/delete graph



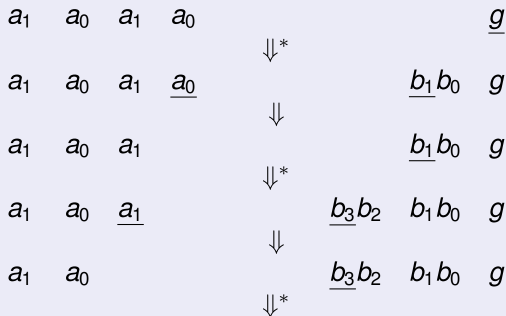
Example: $\alpha(n) = 2n$



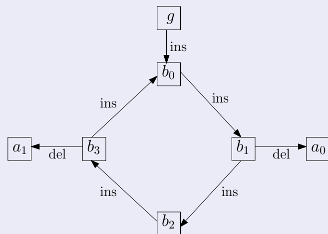
Insert/delete graph



Example: $\alpha(n) = 2n$



Insert/delete graph



Example: $\alpha(n) = 2n$

$$a_1 \quad a_0 \quad a_1 \quad a_0 \qquad\qquad \underline{g}$$

$$\Downarrow^*$$

$$a_1 \quad a_0 \quad a_1 \quad \underline{a_0} \qquad\qquad \underline{b_1} b_0 \quad g$$

$$\Downarrow$$

$$a_1 \quad a_0 \quad a_1 \qquad\qquad \underline{b_1} b_0 \quad g$$

$$\Downarrow^*$$

$$a_1 \quad a_0 \quad \underline{a_1} \qquad\qquad \underline{b_3} b_2 \quad b_1 b_0 \quad g$$

$$\Downarrow$$

$$a_1 \quad a_0 \qquad\qquad \underline{b_3} b_2 \quad b_1 b_0 \quad g$$

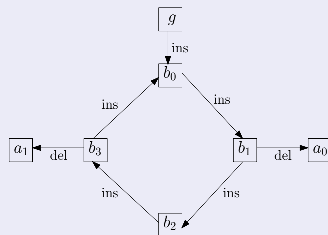
$$\Downarrow^*$$

$$a_1 \quad a_0 \qquad\qquad b_1 b_0 \quad b_3 b_2 \quad b_1 b_0 \quad g$$

$$\Downarrow$$

$\ldots$

Insert/delete graph



Exponentiation?

Exponentiation for the input subalphabets I_0, I_1

Build grammars $\mathcal{G}_0, \mathcal{G}_1$ such that:

		input	output	
$\mathcal{G}_0$	weakly comp. $\alpha(n) = 2n$ wrt	O_1	O_0	g_0
$\mathcal{G}_1$	weakly comp. $\alpha(n) = 2n$ wrt	O_0	O_1	g_1

Exponentiation?

Exponentiation for the input subalphabets I_0, I_1

Build grammars $\mathcal{G}_0, \mathcal{G}_1$ such that:

		input	output	
$\mathcal{G}_0$	weakly comp. $\alpha(n) = 2n$ wrt	O_1	O_0	g_0
$\mathcal{G}_1$	weakly comp. $\alpha(n) = 2n$ wrt	O_0	O_1	g_1

Moreover

- elements of O_0 can delete elements of I_0;
- elements of O_1 can delete elements of I_1.

Exponentiation?

Exponentiation for the input subalphabets I_0, I_1

Build grammars $\mathcal{G}_0, \mathcal{G}_1$ such that:

		input	output	
$\mathcal{G}_0$	weakly comp. $\alpha(n) = 2n$ wrt	O_1	O_0	g_0
$\mathcal{G}_1$	weakly comp. $\alpha(n) = 2n$ wrt	O_0	O_1	g_1

Moreover

- elements of O_0 can delete elements of I_0;
- elements of O_1 can delete elements of I_1.

Finally, add productions

$$g \xrightarrow{\text{ins}} g_i \quad \text{and} \quad g \xrightarrow{\text{del}} g_i \text{ for } i \in [0, 3].$$

Exponentiation?

Exponentiation for the input subalphabets I_0, I_1

Build grammars $\mathcal{G}_0, \mathcal{G}_1$ such that:

		input	output	
$\mathcal{G}_0$	weakly comp. $\alpha(n) = 2n$ wrt	O_1	O_0	g_0
$\mathcal{G}_1$	weakly comp. $\alpha(n) = 2n$ wrt	O_0	O_1	g_1

Moreover

- elements of O_0 can delete elements of I_0;
- elements of O_1 can delete elements of I_1.

Finally, add productions

$$g \stackrel{\text{ins}}{\to} g_i \quad \text{and} \quad g \stackrel{\text{del}}{\to} g_i \text{ for } i \in [0,3].$$

Claim

This grammar weakly computes $\alpha(n) = 2^n$ wrt I, $O_0 \cup O_1, g$ where $I = I_0 \cup I_1$.

Exponentiation: how does it work

$\imath_1$	$\imath_0$	$\imath_1$	$\imath_0$	g

$\imath_0 \in I_0 \quad \imath_1 \in I_1$

Exponentiation: how does it work

$\imath_1$	$\imath_0$	$\imath_1$	$\imath_0$	g

$\imath_0 \in I_0 \quad \imath_1 \in I_1$

$G_0 \Downarrow^*$

$\imath_1$	$\imath_0$	$\imath_1$	w_1	g

$w_1 \in O_0^*$

Exponentiation: how does it work

$\imath_1$	$\imath_0$	$\imath_1$	$\imath_0$	g

$\imath_0 \in I_0$ $\imath_1 \in I_1$

$G_0 \Downarrow^*$

$\imath_1$	$\imath_0$	$\imath_1$	w_1	g

2^0

$w_1 \in O_0^*$

$G_1 \Downarrow^*$

$\imath_1$	$\imath_0$	$w_2 \in O_1^+$	g

2^1

Exponentiation: how does it work

$\imath_1$	$\imath_0$	$\imath_1$	$\imath_0$	g

$\imath_0 \in I_0 \quad \imath_1 \in I_1$

$G_0 \Downarrow^*$

$\imath_1$	$\imath_0$	$\imath_1$	w_1	g

2^0

$w_1 \in O_0^*$

$G_1 \Downarrow^*$

$\imath_1$	$\imath_0$	$w_2 \in O_1^+$	g

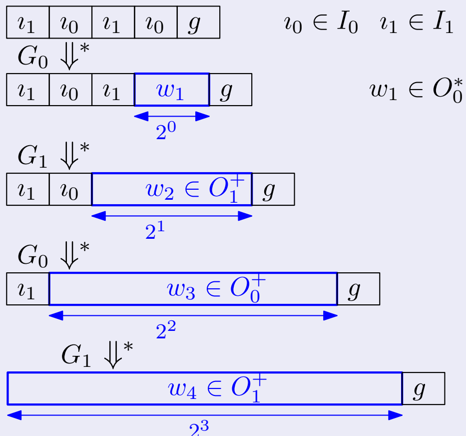
2^1

$G_0 \Downarrow^*$

$\imath_1$	$w_3 \in O_0^+$	g

2^2

Exponentiation: how does it work



More complex functions

Ackermann's function

Let $\phi_p : \mathbb{N} \rightarrow \mathbb{N}$ be as follows:

More complex functions

Ackermann's function

Let $\phi_p : \mathbb{N} \rightarrow \mathbb{N}$ be as follows:

$$\phi_p(0) = 1 \qquad \text{for each } p \geq 2,$$

More complex functions

Ackermann's function

Let $\phi_p : \mathbb{N} \rightarrow \mathbb{N}$ be as follows:

$$\begin{array}{lcll} \phi_p(0) & = & 1 & \text{for each } p \geq 2, \\ \phi_2(n) & = & 2 \cdot n & \text{for } n > 0, \end{array}$$

More complex functions

Ackermann's function

Let $\phi_p : \mathbb{N} \to \mathbb{N}$ be as follows:

$$
\begin{array}{lcll}
\phi_p(0) & = & 1 & \text{for each } p \geq 2, \\
\phi_2(n) & = & 2 \cdot n & \text{for } n > 0, \\
\phi_p(n) & = & \phi_{p-1}^{(n)}(1) & \text{for } p > 2, n > 0.
\end{array}
$$

More complex functions

Ackermann's function

Let $\phi_p : \mathbb{N} \rightarrow \mathbb{N}$ be as follows:

$$\begin{array}{lcll} \phi_p(0) & = & 1 & \text{for each } p \geq 2, \\ \phi_2(n) & = & 2 \cdot n & \text{for } n > 0, \\ \phi_p(n) & = & \phi_{p-1}^{(n)}(1) & \text{for } p > 2, n > 0. \end{array}$$

Ackermann's function:
$Ack(n) = \phi_n(3)$.

More complex functions

Ackermann's function

Let $\phi_p : \mathbb{N} \to \mathbb{N}$ be as follows:

$$\begin{array}{lcll} \phi_p(0) & = & 1 & \text{for each } p \geq 2, \\ \phi_2(n) & = & 2 \cdot n & \text{for } n > 0, \\ \phi_p(n) & = & \phi_{p-1}^{(n)}(1) & \text{for } p > 2, n > 0. \end{array}$$

Ackermann's function:
$Ack(n) = \phi_n(3)$.

Fact

$Ack : \mathbb{N} \to \mathbb{N}$ dominates any primitive recursive function $\alpha : \mathbb{N} \to \mathbb{N}$.

More complex functions

Ackermann's function

Let $\phi_p : \mathbb{N} \to \mathbb{N}$ be as follows:

$$
\begin{array}{lcll}
\phi_p(0) & = & 1 & \text{for each } p \geq 2, \\
\phi_2(n) & = & 2 \cdot n & \text{for } n > 0, \\
\phi_p(n) & = & \phi_{p-1}^{(n)}(1) & \text{for } p > 2, n > 0.
\end{array}
$$

Ackermann's function:
$Ack(n) = \phi_n(3)$.

Fact

$Ack : \mathbb{N} \to \mathbb{N}$ dominates any primitive recursive function $\alpha : \mathbb{N} \to \mathbb{N}$.

Inverse functions $(\phi_p^{-1})_{p \geq 2}$

$$\phi_p^{-1}(m) := \min\{n \mid \phi_p(n) \geq m\}.$$

Ackermann's Function

"Theorem"

For each $p \geq 2$, there exists a leftist grammar which weakly computes ϕ_p.

Ackermann's Function

"Theorem"

For each $p \geq 2$, there exists a leftist grammar which weakly computes ϕ_p.

"Theorem"

For each $p \geq 2$, there exists a leftist grammar which weakly computes ϕ_p^{-1}.

Main reduction

Fact

The problem whether a one-tape Turing machine M halts on an empty input in $Ack(|M|)$ space is non-primitive recursive.

Main reduction

Fact

The problem whether a one-tape Turing machine M halts on an empty input in $Ack(|M|)$ space is non-primitive recursive.

Reduction

Problem

- Input: a Turing machine M.
- Output: yes iff M halts on an empty input in $Ack(|M|)$ space.

Main reduction

Fact

The problem whether a one-tape Turing machine M halts on an empty input in $Ack(|M|)$ space is non-primitive recursive.

Reduction

Problem

- Input: a Turing machine M.
- Output: yes iff M halts on an empty input in $Ack(|M|)$ space.

is reduced to the problem

- Input: a leftist grammar $G(M)$ and an input word $w(M)$
- Output: yes iff $w(M) \in L(G(M))$.

Main reduction cont.

Reduction

Combine grammars:

Main reduction cont.

Reduction

Combine grammars:

- $\mathcal{G}_1$ which weakly computes $\phi_p(n)$

Main reduction cont.

Reduction

Combine grammars:

- $\mathcal{G}_1$ which weakly computes $\phi_p(n)$
- $\mathcal{G}_2$ which "weakly simulates" a computation of M in space equal to the "length" of an input word (see J., 06)

Main reduction cont.

Reduction

Combine grammars:

- $\mathcal{G}_1$ which weakly computes $\phi_p(n)$
- $\mathcal{G}_2$ which "weakly simulates" a computation of M in space equal to the "length" of an input word (see J., 06)
- $\mathcal{G}_3$ which weakly computes ϕ_p^{-1};

Main reduction cont.

Reduction

Combine grammars:

- $\mathcal{G}_1$ which weakly computes $\phi_p(n)$
- $\mathcal{G}_2$ which "weakly simulates" a computation of M in space equal to the "length" of an input word (see J., 06)
- $\mathcal{G}_3$ which weakly computes ϕ_p^{-1};
- $\mathcal{G}_4$ which verifies whether the length of the word left by $\mathcal{G}_3$ is $\leq n$;

where $p := |M|$.

Main reduction cont.

Reduction

Combine grammars:

- $\mathcal{G}_1$ which weakly computes $\phi_p(n)$
- $\mathcal{G}_2$ which "weakly simulates" a computation of M in space equal to the "length" of an input word (see J., 06)
- $\mathcal{G}_3$ which weakly computes ϕ_p^{-1};
- $\mathcal{G}_4$ which verifies whether the length of the word left by $\mathcal{G}_3$ is $\leq n$;

where $p := |M|$.
Check membership for $n = 3$.

Main reduction cont.

How does it work

The following conditions

Main reduction cont.

How does it work

The following conditions

- $\mathcal{G}_1$: $n_1 \geq \phi_p(n)$

Main reduction cont.

How does it work

The following conditions

- $\mathcal{G}_1$: $n_1 \geq \phi_p(n)$
- $\mathcal{G}_2$: "weak simulation" of M in SPACE n,
  result: $n_2 \geq n_1$;

Main reduction cont.

How does it work

The following conditions

- $\mathcal{G}_1$: $n_1 \geq \phi_p(n)$
- $\mathcal{G}_2$: "weak simulation" of M in SPACE n,
  result: $n_2 \geq n_1$;
- $\mathcal{G}_3$: $n_3 \geq \phi_p^{-1}(n_2)$;

Main reduction cont.

How does it work

The following conditions

- $\mathcal{G}_1$: $n_1 \geq \phi_p(n)$
- $\mathcal{G}_2$: "weak simulation" of M in SPACE n,
  result: $n_2 \geq n_1$;
- $\mathcal{G}_3$: $n_3 \geq \phi_p^{-1}(n_2)$;
- $\mathcal{G}_4$: $n_3 \leq n$;

Main reduction cont.

How does it work

The following conditions

- $\mathcal{G}_1$: $n_1 \geq \phi_p(n)$
- $\mathcal{G}_2$: “weak simulation” of M in SPACE n,
  result: $n_2 \geq n_1$;
- $\mathcal{G}_3$: $n_3 \geq \phi_p^{-1}(n_2)$;
- $\mathcal{G}_4$: $n_3 \leq n$;

hold iff $n = n_1 = n_2 = n_3$, i.e., the grammar correctly “simulates” computation of M in space $\phi_p(n)$.

Main reduction cont.

How does it work

The following conditions

- $\mathcal{G}_1$: $n_1 \geq \phi_p(n)$
- $\mathcal{G}_2$: "weak simulation" of M in SPACE n,
  result: $n_2 \geq n_1$;
- $\mathcal{G}_3$: $n_3 \geq \phi_p^{-1}(n_2)$;
- $\mathcal{G}_4$: $n_3 \leq n$;

hold iff $n = n_1 = n_2 = n_3$, i.e., the grammar correctly "simulates" computation of M in space $\phi_p(n)$.

Theorem

The variable membership problem for leftist grammars is non-primitive recursive.

Open problems – generalizations

Extremist grammars

Open problems – generalizations

Extremist grammars

- a finite alphabet Σ,

Open problems – generalizations

Extremist grammars

- a finite alphabet Σ,
- a final symbol $x \in \Sigma$,

Open problems – generalizations

Extremist grammars

- a finite alphabet Σ,
- a final symbol $x \in \Sigma$,
- production rules P:

$$
\begin{aligned}
ab &\rightarrow b && \text{(Left Delete Rule)} && (1)\\
c &\rightarrow dc && \text{(Left Insert Rule)} && (2)\\
ba &\rightarrow b && \text{(Right Delete Rule)} && (3)\\
c &\rightarrow cd && \text{(Right Insert Rule)} && (4)
\end{aligned}
$$

where $a, b, c, d \in \Sigma$.

Open problems – generalizations

Extremist grammars

- a finite alphabet Σ,
- a final symbol $x \in \Sigma$,
- production rules P:

$$\begin{array}{rcll} ab & \to & b & \text{(Left Delete Rule)} \quad (1) \\ c & \to & dc & \text{(Left Insert Rule)} \quad (2) \\ ba & \to & b & \text{(Right Delete Rule)} \quad (3) \\ c & \to & cd & \text{(Right Insert Rule)} \quad (4) \end{array}$$

where $a, b, c, d \in \Sigma$.

Problems

- Decidability of membership problem for extremist grammars?

Open problems – generalizations

Extremist grammars

- a finite alphabet Σ,
- a final symbol $x \in \Sigma$,
- production rules P:

$$\begin{array}{rclll} ab & \to & b & \text{(Left Delete Rule)} & (1) \\ c & \to & dc & \text{(Left Insert Rule)} & (2) \\ ba & \to & b & \text{(Right Delete Rule)} & (3) \\ c & \to & cd & \text{(Right Insert Rule)} & (4) \end{array}$$

where $a, b, c, d \in \Sigma$.

Problems

- Decidability of membership problem for extremist grammars?
- Complexity of extremist grammars?

Open problems – generalizations

Extremist grammars

- a finite alphabet Σ,
- a final symbol $x \in \Sigma$,
- production rules P:

$$
\begin{aligned}
ab &\rightarrow b && \text{(Left Delete Rule)} && (1)\\
c &\rightarrow dc && \text{(Left Insert Rule)} && (2)\\
ba &\rightarrow b && \text{(Right Delete Rule)} && (3)\\
c &\rightarrow cd && \text{(Right Insert Rule)} && (4)
\end{aligned}
$$

where $a, b, c, d \in \Sigma$.

Problems

- Decidability of membership problem for extremist grammars?
- Complexity of extremist grammars?
- Grammars with productions of types (1), (2), (3) or (1), (2), (4) ...?

Outline

1. Motivations and definitions
2. Accessibility vs Grammars
3. Complexity of General Leftist Grammars
4. Restricted leftist grammars
5. Alphabet size restriction

Restricted leftist grammars vs the Chomsky hierarchy

Restricted leftist grammars (J., Loryś ’05)

Delete graph	Insert graph	Included in	Not included in
acyclic	arbitrary	REG	FIN
arbitrary	empty	DCFL	REG
arbitrary	acyclic	CFL	DCFL
arbitrary	arbitrary	recursive*	SPACE(α)**

$\star$ – proved by MPSV

$\star\star$ – α is a primitive recursive function.

Open problems – restricted leftist grammars

Notation

LG(A, B):
languages defined by leftist grammars with delete graphs of type A and insert graphs of type B.

Open problems – restricted leftist grammars

Notation

LG(A, B):
languages defined by leftist grammars with delete graphs of type A and insert graphs of type B.

Problems

1. Characterizations of classes
   LG(*acyclic*, *arbitrary*)
   LG(*arbitrary*, *empty*)
   LG(*arbitrary*, *acyclic*).

Open problems – restricted leftist grammars

Notation

LG(A, B):
languages defined by leftist grammars with delete graphs of type A and insert graphs of type B.

Problems

1. Characterizations of classes
   LG(*acyclic*, *arbitrary*)
   LG(*arbitrary*, *empty*)
   LG(*arbitrary*, *acyclic*).
2. Comparison of these classes with REG_+, CFL_+, DCFL_+, where X_+ is the closure of the class X under stuttering.

Variable membership problem for restricted leftist grammars

Complexity of variable membership problem (J., '06, '08)

Delete graph	Insert graph	Lower bound	Upper bound
arbitrary	empty	?	EXPSPACE
acyclic	acyclic	PSPACE	EXPSPACE
acyclic	arbitrary	PSPACE	EXPSPACE
arbitrary	acyclic	PSPACE	EXPSPACE
arbitrary	arbitrary	non-prim.-rec.	recursive*

$\star$ – proved by MPSV

Variable membership problem for restricted leftist grammars

Complexity of variable membership problem (J., ’06, ’08)

Delete graph	Insert graph	Lower bound	Upper bound
arbitrary	empty	?	EXPSPACE
acyclic	acyclic	PSPACE	EXPSPACE
acyclic	arbitrary	PSPACE	EXPSPACE
arbitrary	acyclic	PSPACE	EXPSPACE
arbitrary	arbitrary	non-prim.-rec.	recursive *

$\star$ – proved by MPSV

Problems

1. Close gaps between lower and upper bounds.

Outline

1. Motivations and definitions
2. Accessibility vs Grammars
3. Complexity of General Leftist Grammars
4. Restricted leftist grammars
5. Alphabet size restriction

Leftist grammars with small alphabets

Assumption

For a grammar $\mathcal{G} = (\Sigma, P, x)$, productions of types

$$a \xrightarrow{\text{ins}} x$$
$$a \xrightarrow{\text{del}} x$$

are not allowed.

Leftist grammars with small alphabets

Assumption

For a grammar $\mathcal{G} = (\Sigma, P, x)$, productions of types

$$a \overset{\text{ins}}{\longrightarrow} x$$
$$a \overset{\text{del}}{\longrightarrow} x$$

are not allowed.

S. Bandyopadhyay, M. Mahajan and K. N. Kumar (05)

- Languages over two-letter alphabets defined by leftist grammars are regular.

Leftist grammars with small alphabets

Assumption

For a grammar $\mathcal{G} = (\Sigma, P, x)$, productions of types

$$a \overset{\text{ins}}{\longrightarrow} x$$
$$a \overset{\text{del}}{\longrightarrow} x$$

are not allowed.

S. Bandyopadhyay, M. Mahajan and K. N. Kumar (05)

- Languages over two-letter alphabets defined by leftist grammars are regular.
- There exists a non-regular language over three-letter alphabet defined by a leftist grammar.

Small alphabets – problems

Conjecture

Languages over three-letter alphabets defined by leftist grammars are context-free.

Small alphabets – problems

Conjecture

Languages over three-letter alphabets defined by leftist grammars are context-free.

Problems

1. Relationships between complexity of grammars and the size of the alphabet?

Small alphabets – problems

Conjecture

Languages over three-letter alphabets defined by leftist grammars are context-free.

Problems

1. Relationships between complexity of grammars and the size of the alphabet?
2. Exact characterizations of leftist languages over 2-letter, 3-letter, ... alphabets?

The End

Thank you for your attention!