Languages in Group Theory

Recent Past and Open Problems

Claas Röver

NUI Galway

Groups St Andrews 2009 in Bath

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language = those $w \in A^*$ which can be produced from S

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language $=$ those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members ... *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language = those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

    $S \to aaS$, $S \to abS$, $S \to baS$, $S \to bbS$, $S \to \varepsilon$

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members ... *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language = those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

    $S \to aaS$, $S \to abS$, $S \to baS$, $S \to bbS$, $S \to \varepsilon$

  - **context-free**: $X \to YZ$, $X \to a$, $X, Y, Z \in V$, $a \in A$

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language $=$ those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

    $S \to aaS$, $S \to abS$, $S \to baS$, $S \to bbS$, $S \to \varepsilon$

  - **context-free**: $X \to YZ$, $X \to a$, $X, Y, Z \in V$, $a \in A$

    $S \to AT$, $A \to a$, $T \to SA$, $S \to BU$, $B \to b$, $U \to SB$, $S \to \varepsilon$

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language $=$ those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

    $S \to aaS, \quad S \to abS, \quad S \to baS, \quad S \to bbS, \quad S \to \varepsilon$

  - **context-free**: $X \to YZ$, $X \to a$, $X, Y, Z \in V$, $a \in A$

    $S \to AT$, $A \to a$, $T \to SA$, $S \to BU$, $B \to b$, $U \to SB$, $S \to \varepsilon$

S

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language $=$ those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

    $S \to aaS, \quad S \to abS, \quad S \to baS, \quad S \to bbS, \quad S \to \varepsilon$

  - **context-free**: $X \to YZ$, $X \to a$, $X, Y, Z \in V$, $a \in A$

    $S \to AT$, $A \to a$, $T \to SA$, $S \to BU$, $B \to b$, $U \to SB$, $S \to \varepsilon$

S

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language $=$ those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

    $S \to aaS$, $S \to abS$, $S \to baS$, $S \to bbS$, $S \to \varepsilon$

  - **context-free**: $X \to YZ$, $X \to a$, $X, Y, Z \in V$, $a \in A$

    $S \to AT$, $A \to a$, $T \to SA$, $S \to BU$, $B \to b$, $U \to SB$, $S \to \varepsilon$

$S \to BU$

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language $=$ those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

    $S \to aaS$, $S \to abS$, $S \to baS$, $S \to bbS$, $S \to \varepsilon$

  - **context-free**: $X \to YZ$, $X \to a$, $X, Y, Z \in V$, $a \in A$

    $S \to AT$, $A \to a$, $T \to SA$, $S \to BU$, $B \to b$, $U \to SB$, $S \to \varepsilon$

$S \to BU \to bU$

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language $=$ those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

    $S \to aaS$, $S \to abS$, $S \to baS$, $S \to bbS$, $S \to \varepsilon$

  - **context-free**: $X \to YZ$, $X \to a$, $X, Y, Z \in V$, $a \in A$

    $S \to AT$, $A \to a$, $T \to SA$, $S \to BU$, $B \to b$, $U \to SB$, $S \to \varepsilon$

$S \to BU \to bU \to bSB$

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language $=$ those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

    $S \to aaS$, $S \to abS$, $S \to baS$, $S \to bbS$, $S \to \varepsilon$

  - **context-free**: $X \to YZ$, $X \to a$, $X, Y, Z \in V$, $a \in A$

    $S \to AT$, $A \to a$, $T \to SA$, $S \to BU$, $B \to b$, $U \to SB$, $S \to \varepsilon$

$S \to BU \to bU \to bSB \to bSb$

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language $=$ those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

    $S \to aaS$, $S \to abS$, $S \to baS$, $S \to bbS$, $S \to \varepsilon$

  - **context-free**: $X \to YZ$, $X \to a$, $X, Y, Z \in V$, $a \in A$

    $S \to AT$, $A \to a$, $T \to SA$, $S \to BU$, $B \to b$, $U \to SB$, $S \to \varepsilon$

$S \to BU \to bU \to bSB \to bSb \to bATb$

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language $=$ those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

    $S \to aaS$, $S \to abS$, $S \to baS$, $S \to bbS$, $S \to \varepsilon$

  - **context-free**: $X \to YZ$, $X \to a$, $X, Y, Z \in V$, $a \in A$

    $S \to AT$, $A \to a$, $T \to SA$, $S \to BU$, $B \to b$, $U \to SB$, $S \to \varepsilon$

$S \to BU \to bU \to bSB \to bSb \to bATb \to bASAb$

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language = those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

    $S \to aaS$, $S \to abS$, $S \to baS$, $S \to bbS$, $S \to \varepsilon$

  - **context-free**: $X \to YZ$, $X \to a$, $X, Y, Z \in V$, $a \in A$

    $S \to AT$, $A \to a$, $T \to SA$, $S \to BU$, $B \to b$, $U \to SB$, $S \to \varepsilon$

$S \to BU \to bU \to bSB \to bSb \to bATb \to bASAb \to baSab$

LANGUAGES

A language is a subset L of a free monoid A^*, $|A| < \infty$.

Problem. How can one specify a language?

A procedure that lists all members . . . *rec. enum.*

- a dictionary *finite language*
- a **grammar** consists of a set of variables V, the set of terminals A, a start symbol $S \in V$ and production rules

  Language = those $w \in A^*$ which can be produced from S

  - **regular**: $X \to wY$, $X, Y \in V$, $w \in A^*$

    $S \to aaS$, $S \to abS$, $S \to baS$, $S \to bbS$, $S \to \varepsilon$

  - **context-free**: $X \to YZ$, $X \to a$, $X, Y, Z \in V$, $a \in A$

    $S \to AT$, $A \to a$, $T \to SA$, $S \to BU$, $B \to b$, $U \to SB$, $S \to \varepsilon$

$$S \to BU \to bU \to bSB \to bSb \to bATb \to bASAb \to baSab \to baab$$

LANGUAGES - DECIDING MEMBERSHIP

A procedure for deciding membership. *recursive*

LANGUAGES - DECIDING MEMBERSHIP

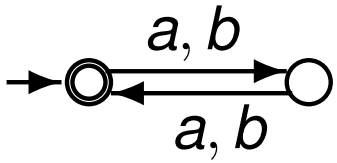
A procedure for deciding membership. *recursive*

- finite state automaton *regular*

LANGUAGES - DECIDING MEMBERSHIP

A procedure for deciding membership. *recursive*

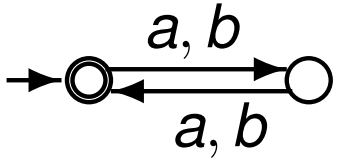
- finite state automaton *regular*



LANGUAGES - DECIDING MEMBERSHIP

A procedure for deciding membership. *recursive*

- finite state automaton *regular*

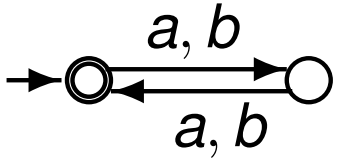


$$L = \{w \in A^* \mid |w| \text{ even}\}$$

LANGUAGES - DECIDING MEMBERSHIP

A procedure for deciding membership. *recursive*

- finite state automaton *regular*



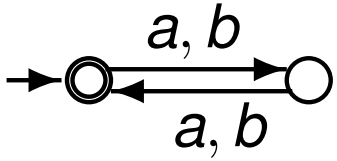
$$L = \{w \in A^* \mid |w| \text{ even}\}$$

- pushdown automaton *context-free*
  has in addition a LIFO store (Last-In-First-Out)

LANGUAGES - DECIDING MEMBERSHIP

A procedure for deciding membership. *recursive*

- finite state automaton *regular*



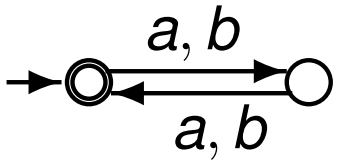
$$L = \{w \in A^* \mid |w| \text{ even}\}$$

- pushdown automaton *context-free*
  has in addition a LIFO store (Last-In-First-Out)
- nested stack automaton *indexed*

LANGUAGES - DECIDING MEMBERSHIP

A procedure for deciding membership. *recursive*

- finite state automaton *regular*



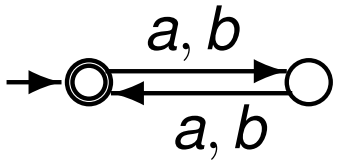
$$L = \{w \in A^* \mid |w| \text{ even}\}$$

- pushdown automaton *context-free*
  has in addition a LIFO store (Last-In-First-Out)
- nested stack automaton *indexed*
- halting Turing machine with running *time constraints*

LANGUAGES - DECIDING MEMBERSHIP

A procedure for deciding membership. *recursive*

- finite state automaton *regular*



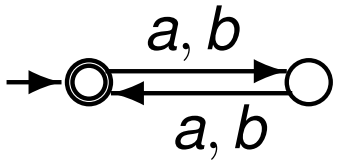
$$L = \{w \in A^* \mid |w| \text{ even}\}$$

- pushdown automaton *context-free*
  has in addition a LIFO store (Last-In-First-Out)
- nested stack automaton *indexed*
- halting Turing machine with running *time constraints*
- halting Turing machine with storage *space constraints*

LANGUAGES - DECIDING MEMBERSHIP

A procedure for deciding membership. *recursive*

- finite state automaton *regular*



$$L = \{w \in A^* \mid |w| \text{ even}\}$$

- pushdown automaton *context-free*
  has in addition a LIFO store (Last-In-First-Out)
- nested stack automaton *indexed*
- halting Turing machine with running *time constraints*
- halting Turing machine with storage *space constraints*
- halting Turing machine *recursive*

Languages - in Groups

Combinatorial Group Theory is naturally concerned with words.

$$G = \langle X \mid \mathcal{R} \rangle \qquad A = X \cup X^{-1}, \mathcal{R} \subset A^*$$

Languages - in Groups

Combinatorial Group Theory is naturally concerned with words.

$$G = \langle X \mid \mathcal{R} \rangle \qquad A = X \cup X^{-1}, \mathcal{R} \subset A^*$$

- Normal Form Theorems for free products (with amalgamation) and HNN-extensions

Languages - in Groups

Combinatorial Group Theory is naturally concerned with words.

$$G = \langle X \mid \mathcal{R} \rangle \qquad A = X \cup X^{-1}, \mathcal{R} \subset A^*$$

- Normal Form Theorems for free products (with amalgamation) and HNN-extensions
- Word Problem: $WP_X(G) = \{ w \in A^* \mid w =_G 1 \}$

Languages - in Groups

Combinatorial Group Theory is naturally concerned with words.

$$G = \langle X \mid \mathcal{R} \rangle \qquad A = X \cup X^{-1}, \mathcal{R} \subset A^*$$

- Normal Form Theorems for free products (with amalgamation) and HNN-extensions
- Word Problem: $WP_X(G) = \{w \in A^* \mid w =_G 1\}$
- Coword Problem: $coWP_X(G) = \{w \in A^* \mid w \neq_G 1\}$

Languages - in Groups

Combinatorial Group Theory is naturally concerned with words.

$$G = \langle X \mid \mathcal{R} \rangle \qquad A = X \cup X^{-1}, \mathcal{R} \subset A^*$$

- Normal Form Theorems for free products (with amalgamation) and HNN-extensions
- Word Problem: $WP_X(G) = \{ w \in A^* \mid w =_G 1 \}$
- Coword Problem: $coWP_X(G) = \{ w \in A^* \mid w \neq_G 1 \}$
- Geodesics: $GL_X(G) = \{ w \in A^* \mid v =_G w \Rightarrow |w| \leq |v| \}$

Languages - in Groups

Combinatorial Group Theory is naturally concerned with words.

$$G = \langle X \mid \mathcal{R} \rangle \qquad A = X \cup X^{-1}, \mathcal{R} \subset A^*$$

- Normal Form Theorems for free products (with amalgamation) and HNN-extensions
- Word Problem: $WP_X(G) = \{ w \in A^* \mid w =_G 1 \}$
- Coword Problem: $coWP_X(G) = \{ w \in A^* \mid w \neq_G 1 \}$
- Geodesics: $GL_X(G) = \{ w \in A^* \mid v =_G w \Rightarrow |w| \leq |v| \}$
- Multiplication: $MT_X(G) = \{ (u, v, w) \in A^{*3} \mid uv =_G w \}$

Languages - in Groups

Combinatorial Group Theory is naturally concerned with words.

$$G = \langle X \mid \mathcal{R} \rangle \qquad A = X \cup X^{-1}, \mathcal{R} \subset A^*$$

- Normal Form Theorems for free products (with amalgamation) and HNN-extensions
- Word Problem: $WP_X(G) = \{ w \in A^* \mid w =_G 1 \}$
- Coword Problem: $coWP_X(G) = \{ w \in A^* \mid w \neq_G 1 \}$
- Geodesics: $GL_X(G) = \{ w \in A^* \mid v =_G w \Rightarrow |w| \leq |v| \}$
- Multiplication: $MT_X(G) = \{ (u, v, w) \in A^{*3} \mid uv =_G w \}$
- Conjugacy Problem:
  $CP_X(G) = \{ (u, v) \in A^{*2} \mid \exists g \in G : \ u^g = v \}$

Pushdown Automata

A pushdown automaton is given by

- a finite set A, the input alphabet

Pushdown Automata

A pushdown automaton is given by

- a finite set A, the input alphabet
- a finite set Ω, the work symbols

Pushdown Automata

A pushdown automaton is given by

- a finite set A, the input alphabet
- a finite set Ω, the work symbols
- a finite set Σ of states, with start state $\sigma_0 \in \Sigma$

Pushdown Automata

A pushdown automaton is given by

- a finite set A, the input alphabet
- a finite set Ω, the work symbols
- a finite set Σ of states, with start state $\sigma_0 \in \Sigma$
- a finite collection of transitions: $(\sigma, \omega, a) \overset{\tau}{\mapsto} (\sigma', \mu)$, where $\sigma, \sigma' \in \Sigma$, $\omega \in \Omega$, $a \in A \cup \{\varepsilon\}$, $\mu \in \{\varepsilon, \omega\} \cup \omega\Omega$

Pushdown Automata

A pushdown automaton is given by

- a finite set A, the input alphabet
- a finite set Ω, the work symbols
- a finite set Σ of states, with start state $\sigma_0 \in \Sigma$
- a finite collection of transitions: $(\sigma, \omega, a) \overset{\tau}{\mapsto} (\sigma', \mu)$, where $\sigma, \sigma' \in \Sigma$, $\omega \in \Omega$, $a \in A \cup \{\varepsilon\}$, $\mu \in \{\varepsilon, \omega\} \cup \omega\Omega$
- an initialy empty LIFO store, called stack

Pushdown Automata

A pushdown automaton is given by

- a finite set A, the input alphabet
- a finite set Ω, the work symbols
- a finite set Σ of states, with start state $\sigma_0 \in \Sigma$
- a finite collection of transitions: $(\sigma, \omega, a) \overset{\tau}{\mapsto} (\sigma', \mu)$, where $\sigma, \sigma' \in \Sigma$, $\omega \in \Omega$, $a \in A \cup \{\varepsilon\}$, $\mu \in \{\varepsilon, \omega\} \cup \omega\Omega$
- an initialy empty LIFO store, called stack

Being in state σ with ω at the top of the stack,

Pushdown Automata

A pushdown automaton is given by

- a finite set A, the input alphabet
- a finite set Ω, the work symbols
- a finite set Σ of states, with start state $\sigma_0 \in \Sigma$
- a finite collection of transitions: $(\sigma, \omega, a) \overset{\tau}{\mapsto} (\sigma', \mu)$, where $\sigma, \sigma' \in \Sigma$, $\omega \in \Omega$, $a \in A \cup \{\varepsilon\}$, $\mu \in \{\varepsilon, \omega\} \cup \omega\Omega$
- an initialy empty LIFO store, called stack

Being in state σ with ω at the top of the stack, when a is read from the input tape,

Pushdown Automata

A pushdown automaton is given by

- a finite set A, the input alphabet
- a finite set Ω, the work symbols
- a finite set Σ of states, with start state $\sigma_0 \in \Sigma$
- a finite collection of transitions: $(\sigma, \omega, a) \overset{\tau}{\mapsto} (\sigma', \mu)$, where $\sigma, \sigma' \in \Sigma$, $\omega \in \Omega$, $a \in A \cup \{\varepsilon\}$, $\mu \in \{\varepsilon, \omega\} \cup \omega\Omega$
- an initialy empty LIFO store, called stack

Being in state σ with ω at the top of the stack, when a is read from the input tape, the machine chooses (randomly) a transition with left hand side (σ, ω, a), say τ,

Pushdown Automata

A pushdown automaton is given by

- a finite set A, the input alphabet
- a finite set Ω, the work symbols
- a finite set Σ of states, with start state $\sigma_0 \in \Sigma$
- a finite collection of transitions: $(\sigma, \omega, a) \overset{\tau}{\mapsto} (\sigma', \mu)$, where $\sigma, \sigma' \in \Sigma$, $\omega \in \Omega$, $a \in A \cup \{\varepsilon\}$, $\mu \in \{\varepsilon, \omega\} \cup \omega\Omega$
- an initialy empty LIFO store, called stack

Being in state σ with ω at the top of the stack, when a is read from the input tape, the machine chooses (randomly) a transition with left hand side (σ, ω, a), say τ, and then replaces ω by μ and moves into state σ'.

Pushdown Automata

A pushdown automaton is given by

- a finite set A, the input alphabet
- a finite set Ω, the work symbols
- a finite set Σ of states, with start state $\sigma_0 \in \Sigma$
- a finite collection of transitions: $(\sigma, \omega, a) \overset{\tau}{\mapsto} (\sigma', \mu)$, where $\sigma, \sigma' \in \Sigma$, $\omega \in \Omega$, $a \in A \cup \{\varepsilon\}$, $\mu \in \{\varepsilon, \omega\} \cup \omega\Omega$
- an initialy empty LIFO store, called stack

Being in state σ with ω at the top of the stack, when a is read from the input tape, the machine chooses (randomly) a transition with left hand side (σ, ω, a), say τ, and then replaces ω by μ and moves into state σ'.

The automaton accepts the input word, if the stack is empty at the end of some computation.

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x),\ x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon),\ x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy),\ x, y \in A,\ y \neq X$

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x),\ x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon),\ x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy),\ x, y \in A,\ y \neq X$

Consider the input word *aBaAbAAbBa*

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x),\ x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon),\ x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy),\ x, y \in A,\ y \neq X$

Consider the input word *aBaAbAAbBa*

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x)$, $x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon)$, $x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy)$, $x, y \in A$, $y \neq X$

Consider the input word $aBaAbAAbBa$

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x),\ x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon),\ x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy),\ x, y \in A,\ y \neq X$

Consider the input word *aBaAbAAbBa*

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x),\ x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon),\ x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy),\ x, y \in A,\ y \neq X$

Consider the input word *aBaAbAAbBa*

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x),\ x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon),\ x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy),\ x, y \in A,\ y \neq X$

Consider the input word *aBaAbAAbBa*

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x),\ x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon),\ x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy),\ x, y \in A,\ y \neq X$

Consider the input word *aBaAbAAbBa*

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x),\ x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon),\ x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy),\ x, y \in A,\ y \neq X$

Consider the input word *aBaAbAAbBa*

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x)$, $x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon)$, $x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy)$, $x, y \in A$, $y \neq X$

Consider the input word *aBaAbAAbBa*

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x),\ x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon),\ x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy),\ x, y \in A,\ y \neq X$

Consider the input word *aBaAbAAbBa*

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x),\ x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon),\ x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy),\ x, y \in A,\ y \neq X$

Consider the input word *aBaAbAAbBa*

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x),\ x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon),\ x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy),\ x, y \in A,\ y \neq X$

Consider the input word *aBaAbAAbBa*

Prototypical Pushdown Automaton

Example

$A = \Omega = \{a, b, A, B\}, \quad \Sigma = \{\sigma_0\}$

$(\sigma_0, \star, x) \mapsto (\sigma_0, \star x),\ x \in A$

$(\sigma_0, x, X) \mapsto (\sigma_0, \varepsilon),\ x \in A$

$(\sigma_0, x, y) \mapsto (\sigma_0, xy),\ x, y \in A,\ y \neq X$

Consider the input word *aBaAbAAbBa*

$\boxed{\star}$

ACCEPT :-)

Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode

Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

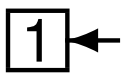
- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

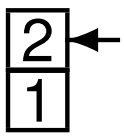


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

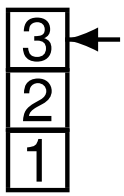


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

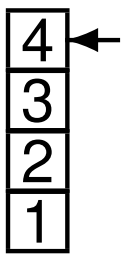


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

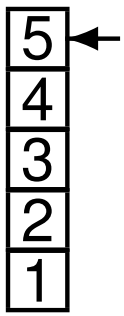


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

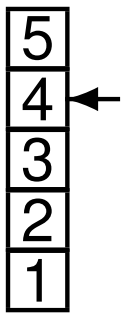


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

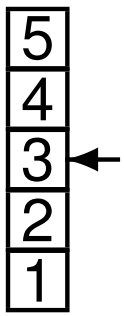


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

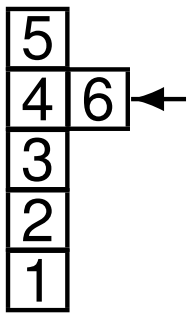


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

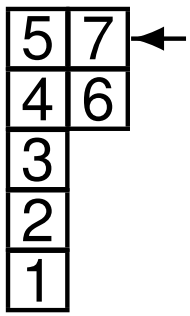


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

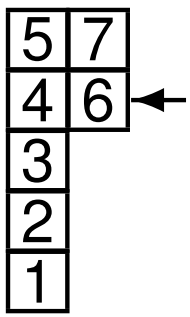


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

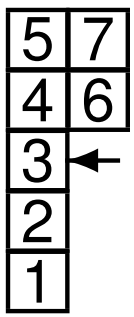


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

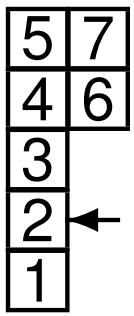


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

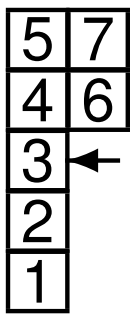


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

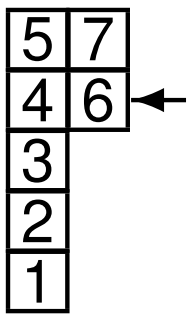


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

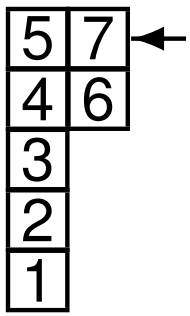


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

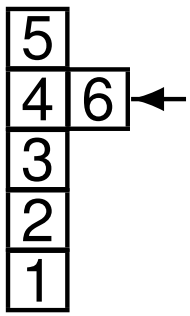


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

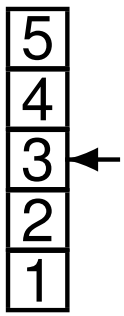


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

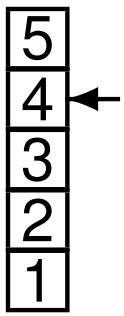


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

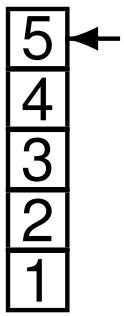


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it

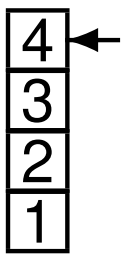


Nested Stack Automata

An indexed language is one accepted by a nested stack automaton.

A nested stack automaton is more powerful than a pushdown automaton in that

- the read head can move up and down in the active stack in read-only mode
- a new stack can be opened, which then becomes the *active stack* together with whatever is below it



WORD PROBLEMS - CLASSICAL RESULTS

LEMMA

Let $\mathcal{C}$ be a class of languages which is closed under inverse homomorphism and let X and Y be finite generating sets for a group G. Then $WP_X(G) \in \mathcal{C} \Leftrightarrow WP_Y(G) \in \mathcal{C}$.

WORD PROBLEMS - CLASSICAL RESULTS

LEMMA

Let $\mathcal{C}$ be a class of languages which is closed under inverse homomorphism and let X and Y be finite generating sets for a group G. Then $WP_X(G) \in \mathcal{C} \Leftrightarrow WP_Y(G) \in \mathcal{C}$.

So we simply write $WP(G)$.

WORD PROBLEMS - CLASSICAL RESULTS

LEMMA

Let $\mathcal{C}$ be a class of languages which is closed under inverse homomorphism and let X and Y be finite generating sets for a group G. Then $WP_X(G) \in \mathcal{C} \Leftrightarrow WP_Y(G) \in \mathcal{C}$.

So we simply write $WP(G)$.

THEOREM (NOVIKOV-BOONE 1950S)

There exists a f.p. group with non-recursive word problem.

Word Problems - classical results

Lemma

Let $\mathcal{C}$ be a class of languages which is closed under inverse homomorphism and let X and Y be finite generating sets for a group G. Then $WP_X(G) \in \mathcal{C} \Leftrightarrow WP_Y(G) \in \mathcal{C}$.

So we simply write $WP(G)$.

Theorem (Novikov-Boone 1950s)

There exists a f.p. group with non-recursive word problem.

Theorem (Annisimov 1971)

$|G| < \infty \Leftrightarrow WP(G)$ *is regular.*

WORD PROBLEMS - CLASSICAL RESULTS

LEMMA

Let $\mathcal{C}$ be a class of languages which is closed under inverse homomorphism and let X and Y be finite generating sets for a group G. Then $WP_X(G) \in \mathcal{C} \Leftrightarrow WP_Y(G) \in \mathcal{C}$.

So we simply write $WP(G)$.

THEOREM (NOVIKOV-BOONE 1950S)

There exists a f.p. group with non-recursive word problem.

THEOREM (ANNISIMOV 1971)

$|G| < \infty \Leftrightarrow WP(G)$ *is regular.*

THEOREM (MULLER-SCHUPP 1983/85, DUNWOODY 1980)

G is virtually free $\Leftrightarrow$ $WP(G)$ is (deterministic) context-free.

Word Problems - classical results

Lemma

Let $\mathcal{C}$ be a class of languages which is closed under inverse homomorphism and let X and Y be finite generating sets for a group G. Then $WP_X(G) \in \mathcal{C} \Leftrightarrow WP_Y(G) \in \mathcal{C}$.

So we simply write $WP(G)$.

Theorem (Novikov-Boone 1950s)

There exists a f.p. group with non-recursive word problem.

Theorem (Annisimov 1971)

$|G| < \infty \Leftrightarrow WP(G)$ *is regular.*

Theorem (Muller-Schupp 1983/85, Dunwoody 1980)

G is virtually free $\Leftrightarrow WP(G)$ is (deterministic) context-free.

Problem

Is a group with indexed word problem virtually free?

Real-Time Word Problems

An *n*-tape real-time language is one accepted by a deterministic Turing machine with n work tapes that can execute at most $K > 0$ instructions per input symbol read.

Real-Time Word Problems

An *n*-tape real-time language is one accepted by a deterministic Turing machine with n work tapes that can execute at most $K > 0$ instructions per input symbol read. This is more restrictive than linear time!

Real-Time Word Problems

An *n*-tape real-time language is one accepted by a deterministic Turing machine with n work tapes that can execute at most $K > 0$ instructions per input symbol read. This is more restrictive than linear time!

Theorem (Holt-Rees 2001)

Word hyperbolic groups, f.g. nilpotent groups and geometrically finite hyperbolic groups have tidy 4-tape real-time word problem.

Real-Time Word Problems

An n-tape real-time language is one accepted by a deterministic Turing machine with n work tapes that can execute at most $K > 0$ instructions per input symbol read. This is more restrictive than linear time!

Theorem (Holt-Rees 2001)

Word hyperbolic groups, f.g. nilpotent groups and geometrically finite hyperbolic groups have tidy 4-tape real-time word problem.

Theorem (Holt-R 2003)

1. $WP(\langle a, b \mid ab = ba^p \rangle)$ *is* 5*-tape tidy real-time.*

Real-Time Word Problems

An *n*-tape real-time language is one accepted by a deterministic Turing machine with n work tapes that can execute at most $K > 0$ instructions per input symbol read. This is more restrictive than linear time!

Theorem (Holt-Rees 2001)

Word hyperbolic groups, f.g. nilpotent groups and geometrically finite hyperbolic groups have tidy 4-tape real-time word problem.

Theorem (Holt-R 2003)

1. $WP(\langle a, b \mid ab = ba^p \rangle)$ *is* 5*-tape tidy real-time.*
2. $WP(F_2 \times F_2)$ *is* 2*-tape but not* 1*-tape real-time.*

Real-Time Word Problems

An *n*-tape real-time language is one accepted by a deterministic Turing machine with n work tapes that can execute at most $K > 0$ instructions per input symbol read. This is more restrictive than linear time!

Theorem (Holt-Rees 2001)

Word hyperbolic groups, f.g. nilpotent groups and geometrically finite hyperbolic groups have tidy 4-tape real-time word problem.

Theorem (Holt-R 2003)

1. $WP(\langle a, b \mid ab = ba^p \rangle)$ *is* 5*-tape tidy real-time.*
2. $WP(F_2 \times F_2)$ *is* 2*-tape but not* 1*-tape real-time.*

Problem

Show that $WP({F_2}^n)$ *is* n*-tape but not* $(n-1)$*-tape real-time.*

Context-Free Coword Problems

Remark. Many language classes are not closed under taking complements, so coword problems become interesting.

Context-Free Coword Problems

Remark. Many language classes are not closed under taking complements, so coword problems become interesting.

Theorem (Holt-Rees-R-Thomas 2005)

Groups with context-free coword problem are closed under

1. *finite direct products*
2. *finitely generated subgroups*
3. *finite extensions*

Context-Free Coword Problems

Remark. Many language classes are not closed under taking complements, so coword problems become interesting.

Theorem (Holt-Rees-R-Thomas 2005)

Groups with context-free coword problem are closed under

1. *finite direct products*
2. *finitely generated subgroups*
3. *finite extensions*
4. *restricted wreath product with virtually free top group*

Context-Free Coword Problems

Remark. Many language classes are not closed under taking complements, so coword problems become interesting.

Theorem (Holt-Rees-R-Thomas 2005)

Groups with context-free coword problem are closed under

1. *finite direct products*
2. *finitely generated subgroups*
3. *finite extensions*
4. *restricted wreath product with virtually free top group*

Theorem (Holt-Rees-R-Thomas 2005)

1. *G polycyclic: $\mathrm{coWP}(G) \in \mathcal{CF} \Leftrightarrow G$ is virtually abelian.*
2. *$\mathrm{coWP}(\langle a, b \mid a^p b = b a^q \rangle) \in \mathcal{CF} \Leftrightarrow p = \pm q$*

Context-Free Coword Problems

Remark. Many language classes are not closed under taking complements, so coword problems become interesting.

Theorem (Holt-Rees-R-Thomas 2005)

Groups with context-free coword problem are closed under

1. *finite direct products*
2. *finitely generated subgroups*
3. *finite extensions*
4. *restricted wreath product with virtually free top group*

Theorem (Holt-Rees-R-Thomas 2005)

1. *G polycyclic: $\mathrm{coWP}(G) \in \mathcal{CF} \Leftrightarrow G$ is virtually abelian.*
2. *$\mathrm{coWP}(\langle a, b \mid a^p b = b a^q \rangle) \in \mathcal{CF} \Leftrightarrow p = \pm q$*

Theorem (Lehnert-Schweitzer 2006)

All Higman-Thompson groups $G_{n,r}$ have context-free coword problem.

Indexed Coword Problems

Theorem (Holt-R 2006)

1. *Under some mild conditions, groups with indexed coword problem are closed under taking free products.*
2. *Bounded automata groups have indexed coword problem.*

Indexed Coword Problems

Theorem (Holt-R 2006)

1. *Under some mild conditions, groups with indexed coword problem are closed under taking free products.*
2. *Bounded automata groups have indexed coword problem.*

In particular

- $coWP((\mathbb{Z} \times \mathbb{Z}) \star C_2)$ is indexed
- 'The' Grigorchuk group has indexed coword problem

Indexed Coword Problems

Theorem (Holt-R 2006)

1. *Under some mild conditions, groups with indexed coword problem are closed under taking free products.*
2. *Bounded automata groups have indexed coword problem.*

In particular

- $coWP((\mathbb{Z} \times \mathbb{Z}) \star C_2)$ is indexed
- 'The' Grigorchuk group has indexed coword problem

Problem

- *Show that* $coWP((\mathbb{Z} \times \mathbb{Z}) * C_2)$ *is not context-free.*
- *Show that the coword problem of 'the' Grigorchuk group is not context-free.*

Context-Free Conjugacy Problem

Recall $CP_X(G) = \{(u, v) \in A^{*2} \mid \exists\, g \in G :\ u^g = v\}$.

Context-Free Conjugacy Problem

Recall $CP_X(G) = \{(u, v) \in A^{*2} \mid \exists\, g \in G :\ u^g = v\}$.

With multiple input words, a machine may read them synchronously or asynchronously.

Context-Free Conjugacy Problem

Recall $CP_X(G) = \{(u, v) \in A^{*2} \mid \exists\, g \in G :\ u^g = v\}$.

With multiple input words, a machine may read them synchronously or asynchronously.

$ICP_X(G) = \{(u, v^{-1}) \in A^{*2} \mid \exists\, g \in G :\ u^g = v\}$.

Context-Free Conjugacy Problem

Recall $CP_X(G) = \{(u, v) \in A^{*2} \mid \exists\, g \in G :\ u^g = v\}$.

With multiple input words, a machine may read them synchronously or asynchronously.

$ICP_X(G) = \{(u, v^{-1}) \in A^{*2} \mid \exists\, g \in G :\ u^g = v\}$.

Theorem (Holt-Rees-R 2009)

	sync. context-free	*async. context-free*
$CP(G)$	*virtually cyclic*	*virtually cyclic*
$ICP(G)$	*virtually cyclic*	*virtually free*

The End