LANGUAGES AND GROUPS

Murray Elder

Les Diablerets, March 7-8 2016

PLAN

Word problems:

- introduce various classes of formal languages: regular, counter, context-free, indexed, ET0L, EDT0L, context-sensitive
- time and space complexity for Turing machine algorithms

Other problems:

- solutions to equations over groups

Let G be a group with finite generating set $X = X^{-1}$.

The set $\{w \in X^* \mid w =_G 1\}$ is called the *word problem* for (G, X).

Let G be a group with finite generating set $X = X^{-1}$.

The set $\{w \in X^* \mid w =_G 1\}$ is called the *word problem* for (G, X).

More typically, the word problem is given as a *decision problem*:

input: $w \in X^*$ output: yes if $w =_G 1$, no otherwise

Let G be a group with finite generating set $X = X^{-1}$.

The set $\{w \in X^* \mid w =_G 1\}$ is called the *word problem* for (G, X).

More typically, the word problem is given as a *decision problem*:

input: $w \in X^*$ output: yes if $w =_G 1$, no otherwise

Eg: $\mathbb{Z}$, integers generated by $1, -1$, or multiplicatively by a, a^{-1}.

Let G be a group with finite generating set $X = X^{-1}$.

The set $\{w \in X^* \mid w =_G 1\}$ is called the *word problem* for (G, X).

More typically, the word problem is given as a *decision problem*:

input: $w \in X^*$ output: yes if $w =_G 1$, no otherwise

Eg: $\mathbb{Z}$, integers generated by $1, -1$, or multiplicatively by a, a^{-1}.

Eg: F_2, free group on a, b, the set of all reduced words in $a^{\pm 1}, b^{\pm 1}$ with operation of *concatenate then reduce*

How can we describe *how hard* the word problem is for some group?

How can we describe *how hard* the word problem is for some group?

- the complexity of the *Turing machine* (algorithm) that solves the decision problem

How can we describe *how hard* the word problem is for some group?

- the complexity of the *Turing machine* (algorithm) that solves the decision problem
- the complexity of the set $\{w \in X^* \mid w =_G 1\}$ in terms of *formal language theory*

DEFINITIONS

- *regular*: accepted by a finite state automaton (FSA)

DEFINITIONS

- *regular*: accepted by a finite state automaton (FSA)
- *1-counter*: FSA with additional counter;

- *regular*: accepted by a finite state automaton (FSA)
- *1-counter*: FSA with additional counter; *blind* if it cannot see the value of the counter while reading; accepts if the input word labels a path from start to accept which returns counter to 0;

DEFINITIONS

- *regular*: accepted by a finite state automaton (FSA)
- *1-counter*: FSA with additional counter; *blind* if it cannot see the value of the counter while reading; accepts if the input word labels a path from start to accept which returns counter to 0; *non-blind* if it can see if the counter if 0 while processing input; equivalent to having a stack with a token 1 plus bottom of stack token

- *regular*: accepted by a finite state automaton (FSA)
- *1-counter*: FSA with additional counter; *blind* if it cannot see the value of the counter while reading; accepts if the input word labels a path from start to accept which returns counter to 0; *non-blind* if it can see if the counter if 0 while processing input; equivalent to having a stack with a token 1 plus bottom of stack token Eg: $a^n b^n a^n$

- *regular*: accepted by a finite state automaton (FSA)
- *1-counter*: FSA with additional counter; *blind* if it cannot see the value of the counter while reading; accepts if the input word labels a path from start to accept which returns counter to 0; *non-blind* if it can see if the counter if 0 while processing input; equivalent to having a stack with a token 1 plus bottom of stack token Eg: $a^n b^n a^n$
- *context-free*: accepted by a *pushdown automaton* (FSA plus stack with finite set of tokens)

- *regular*: accepted by a finite state automaton (FSA)
- *1-counter*: FSA with additional counter; *blind* if it cannot see the value of the counter while reading; accepts if the input word labels a path from start to accept which returns counter to 0; *non-blind* if it can see if the counter if 0 while processing input; equivalent to having a stack with a token 1 plus bottom of stack token Eg: $a^n b^n a^n$
- *context-free*: accepted by a *pushdown automaton* (FSA plus stack with finite set of tokens) Eg: $a^n b^m a^m b^n$

- *regular*: accepted by a finite state automaton (FSA)
- *1-counter*: FSA with additional counter; *blind* if it cannot see the value of the counter while reading; accepts if the input word labels a path from start to accept which returns counter to 0; *non-blind* if it can see if the counter if 0 while processing input; equivalent to having a stack with a token 1 plus bottom of stack token Eg: $a^n b^n a^n$
- *context-free*: accepted by a *pushdown automaton* (FSA plus stack with finite set of tokens) Eg: $a^n b^m a^m b^n$
- *logspace*: Turing machine with input, work and output tapes; on input w length n uses at most $O(\log n)$ squares of the work tape.

Eg: $\mathbb{Z}$

- decision problem: linear time and logspace: scan word and update a binary counter ± 1.

Eg: $\mathbb{Z}$

- decision problem: linear time and logspace: scan word and update a binary counter ± 1.
- language: $\{w \in \{a^{\pm 1}\}^* \mid \#_a = \#_{a^{-1}}\}$ is blind 1-counter and not regular.

Eg: $\mathbb{Z}$

- decision problem: linear time and logspace: scan word and update a binary counter ± 1.
- language: $\{w \in \{a^{\pm 1}\}^* \mid \#_a = \#_{a^{-1}}\}$ is blind 1-counter and not regular.

Eg: F_2

- decision problem:
  - linear time: scan word and push/pop stack: each letter at worst is pushed then popped later, so each letter looked at at most twice.

Eg: $\mathbb{Z}$

- decision problem: linear time and logspace: scan word and update a binary counter ± 1.
- language: $\{w \in \{a^{\pm 1}\}^* \mid \#_a = \#_{a^{-1}}\}$ is blind 1-counter and not regular.

Eg: F_2

- decision problem:
  - linear time: scan word and push/pop stack: each letter at worst is pushed then popped later, so each letter looked at at most twice.
  - logspace [19]

Eg: $\mathbb{Z}$

- decision problem: linear time and logspace: scan word and update a binary counter ± 1.
- language: $\{w \in \{a^{\pm 1}\}^* \mid \#_a = \#_{a^{-1}}\}$ is blind 1-counter and not regular.

Eg: F_2

- decision problem:
  - linear time: scan word and push/pop stack: each letter at worst is pushed then popped later, so each letter looked at at most twice.
  - logspace [19]
- language: context-free and not regular.

F_2 is a subgroup of $SL_2(\mathbb{Z})$: $a \longrightarrow \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}, b \longrightarrow \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$

F_2 is a subgroup of $SL_2(\mathbb{Z})$: $a \longrightarrow \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}, b \longrightarrow \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$

Lipton and Zalcstein [19] gave the following logspace algorithm to solve the word problem for linear groups:

- on input $w \in X^*$, multiply I by the matrix for each letter and store the product with each entry $\mod p$ for some small number.
- if at the end the matrix is not I, then $w \neq_G 1$.
- if at the end the matrix is I for all small numbers, then $w =_G 1$.

Theorem (Anisimov [4])

(G, X) *has regular word problem iff G is finite*

Theorem (Anisimov [4])

(G, X) has regular word problem iff G is finite

Proof: If G is finite, draw the *Cayley graph* for (G, X): node for each $g \in G$, directed edge (g, h) labeled $x \in X^{\pm 1}$ if $h =_G gx$.

The graph is a FSA with start and accept state 1 accepting the word problem for G.

Theorem (Anisimov [4])

(G, X) has regular word problem iff G is finite

Proof: If G is finite, draw the *Cayley graph* for (G, X): node for each $g \in G$, directed edge (g, h) labeled $x \in X^{\pm 1}$ if $h =_G gx$.

The graph is a FSA with start and accept state 1 accepting the word problem for G.

Now suppose G is infinite. If M is a FSA with n states accepting the word problem for G, let $w \in (X^{\pm 1})^*$ be a *geodesic* of length $> n$. Then w has prefix $w_1 w_2$ where both $w_1, w_1 w_2$ end at the same state of M.

Theorem (Anisimov [4])

(G, X) *has regular word problem iff* G *is finite*

Proof: If G is finite, draw the *Cayley graph* for (G, X): node for each $g \in G$, directed edge (g, h) labeled $x \in X^{\pm 1}$ if $h =_G gx$.

The graph is a FSA with start and accept state 1 accepting the word problem for G.

Now suppose G is infinite. If M is a FSA with n states accepting the word problem for G, let $w \in (X^{\pm 1})^*$ be a *geodesic* of length $> n$. Then w has prefix $w_1 w_2$ where both $w_1, w_1 w_2$ end at the same state of M.

Then $w_1 w_1^{-1}$ and $w_1 w_2 w_1^{-1}$ both end at the same state, and since $w_1 w_1^{-1} =_G 1$ this is an accept state, which means $w_1 w_2 w_1^{-1} =_G 1$. Thus $w_2 =_G 1$ so w was not geodesic

Context-free languages can also be described in terms of a *grammar*:

- two finite alphabets T, N with $S \in N$
- finite set of rules $A \longrightarrow w \in (T \cup N)^*$ with $A \in N$
- language is the set of all words in T^* that can be produced by applying rules to S

Context-free languages can also be described in terms of a *grammar*:

- two finite alphabets T, N with $S \in N$
- finite set of rules $A \longrightarrow w \in (T \cup N)^*$ with $A \in N$
- language is the set of all words in T^* that can be produced by applying rules to S

Eg: $T = \{a^{\pm 1}, b^{\pm 1}\}, N = \{S\}$, rules

$$S \longrightarrow aSa^{-1}S \mid bSb^{-1}S \mid a^{-1}SaS \mid b^{-1}SbS \mid \epsilon$$

If a language L is context-free, there is a constant k (depending on the lengths of things in the grammar) so that for words w in L longer than k,

any derivation (drawn as a *parse tree*) must include the same nonterminal twice. This means $w = uvxyz$ and $uv^ixy^iz \in L$ for all $i \in \mathbb{N}$.

If a language L is context-free, there is a constant k (depending on the lengths of things in the grammar) so that for words w in L longer than k,

any derivation (drawn as a *parse tree*) must include the same nonterminal twice. This means $w = uvxyz$ and $uv^ixy^iz \in L$ for all $i \in \mathbb{N}$.

Eg: $L = \{a^nb^na^n \mid n \in \mathbb{N}\}, L = \{ww \mid w \in \{a,b\}^*\}$ are not context-free.

(G, X) has a *Dehn's algorithm* if there exists a finite list

$$\{(u_i, v_i) \mid u_i, v_i \in X^*, u_i =_G v_i, |u_i| > |v_i|\}$$

so that $1 \neq w =_G 1$ implies u_i is a factor of w.

(G, X) has a *Dehn's algorithm* if there exists a finite list

$$\{(u_i, v_i) \mid u_i, v_i \in X^*, u_i =_G v_i, |u_i| > |v_i|\}$$

so that $1 \neq w =_G 1$ implies u_i is a factor of w.

Eg: $F_2 : (aa^{-1}, 1), (a^{-1}a, 1), (bb^{-1}, 1), (b^{-1}b, 1)$.

(G, X) has a *Dehn's algorithm* if there exists a finite list

$$\{(u_i, v_i) \mid u_i, v_i \in X^*, u_i =_G v_i, |u_i| > |v_i|\}$$

so that $1 \neq w =_G 1$ implies u_i is a factor of w.

Eg: $F_2 : (aa^{-1}, 1), (a^{-1}a, 1), (bb^{-1}, 1), (b^{-1}b, 1)$.

G is *hyperbolic* if and only if it has a Dehn's algorithm (Cannon [7]).

(G, X) has a *Dehn's algorithm* if there exists a finite list

$$\{(u_i, v_i) \mid u_i, v_i \in X^*, u_i =_G v_i, |u_i| > |v_i|\}$$

so that $1 \neq w =_G 1$ implies u_i is a factor of w.

Eg: $F_2 : (aa^{-1}, 1), (a^{-1}a, 1), (bb^{-1}, 1), (b^{-1}b, 1)$.

G is *hyperbolic* if and only if it has a Dehn's algorithm (Cannon [7]).

Can we solve the word problem using a stack? Let's try:

(G, X) has a *Dehn's algorithm* if there exists a finite list

$$\{(u_i, v_i) \mid u_i, v_i \in X^*, u_i =_G v_i, |u_i| > |v_i|\}$$

so that $1 \neq w =_G 1$ implies u_i is a factor of w.

Eg: $F_2 : (aa^{-1}, 1), (a^{-1}a, 1), (bb^{-1}, 1), (b^{-1}b, 1)$.

G is *hyperbolic* if and only if it has a Dehn's algorithm (Cannon [7]).

Can we solve the word problem using a stack? Let's try:

- scan the tape and push letters onto a stack;

(G, X) has a *Dehn's algorithm* if there exists a finite list

$$\{(u_i, v_i) \mid u_i, v_i \in X^*, u_i =_G v_i, |u_i| > |v_i|\}$$

so that $1 \neq w =_G 1$ implies u_i is a factor of w.

Eg: $F_2 : (aa^{-1}, 1), (a^{-1}a, 1), (bb^{-1}, 1), (b^{-1}b, 1)$.

G is *hyperbolic* if and only if it has a Dehn's algorithm (Cannon [7]).

Can we solve the word problem using a stack? Let's try:

- scan the tape and push letters onto a stack;
- at each step check the top of the stack for a word u_i (to do this, pop $m = \max|u_i|$ letters off and record in a finite memory)

(G, X) has a *Dehn's algorithm* if there exists a finite list

$$\{(u_i, v_i) \mid u_i, v_i \in X^*, u_i =_G v_i, |u_i| > |v_i|\}$$

so that $1 \neq w =_G 1$ implies u_i is a factor of w.

Eg: $F_2 : (aa^{-1}, 1), (a^{-1}a, 1), (bb^{-1}, 1), (b^{-1}b, 1)$.

G is *hyperbolic* if and only if it has a Dehn's algorithm (Cannon [7]).

Can we solve the word problem using a stack? Let's try:

- scan the tape and push letters onto a stack;
- at each step check the top of the stack for a word u_i (to do this, pop $m = \max |u_i|$ letters off and record in a finite memory)
- if u_i is on top of the stack, replace it by v_i

(G, X) has a *Dehn's algorithm* if there exists a finite list

$$\{(u_i, v_i) \mid u_i, v_i \in X^*, u_i =_G v_i, |u_i| > |v_i|\}$$

so that $1 \neq w =_G 1$ implies u_i is a factor of w.

Eg: $F_2 : (aa^{-1}, 1), (a^{-1}a, 1), (bb^{-1}, 1), (b^{-1}b, 1)$.

G is *hyperbolic* if and only if it has a Dehn's algorithm (Cannon [7]).

Can we solve the word problem using a stack? Let's try:

- scan the tape and push letters onto a stack;
- at each step check the top of the stack for a word u_i (to do this, pop $m = \max |u_i|$ letters off and record in a finite memory)
- if u_i is on top of the stack, replace it by v_i
- problem: this might produce another u_j inside the stack

Do this on a tape instead of a stack: linear time

- let $m = \max |u_i|$, $n =$ length of input

Do this on a tape instead of a stack: linear time

- let $m = \max|u_i|$, $n =$ length of input
- write each letter on tape, checking suffix for u_i

Do this on a tape instead of a stack: linear time

- let $m = \max |u_i|$, $n =$ length of input
- write each letter on tape, checking suffix for u_i
- if found, replace by v_i, and check for u_j by scanning m steps back from v_i

Do this on a tape instead of a stack: linear time

- let $m = \max|u_i|$, $n =$ length of input
- write each letter on tape, checking suffix for u_i
- if found, replace by v_i, and check for u_j by scanning m steps back from v_i
- if found, replace by v_j, repeat (moving backwards)

Do this on a tape instead of a stack: linear time

- let $m = \max |u_i|$, $n =$ length of input
- write each letter on tape, checking suffix for u_i
- if found, replace by v_i, and check for u_j by scanning m steps back from v_i
- if found, replace by v_j, repeat (moving backwards)
- overall, the pointer moves backwards at most n times (since word length is reduced), by at most m steps each time

Do this on a tape instead of a stack: linear time

- let $m = \max |u_i|$, $n =$ length of input
- write each letter on tape, checking suffix for u_i
- if found, replace by v_i, and check for u_j by scanning m steps back from v_i
- if found, replace by v_j, repeat (moving backwards)
- overall, the pointer moves backwards at most n times (since word length is reduced), by at most m steps each time
- it moves forwards n times plus at most nm (the steps it made backwards)

Do this on a tape instead of a stack: linear time

- let $m = \max |u_i|$, $n =$ length of input
- write each letter on tape, checking suffix for u_i
- if found, replace by v_i, and check for u_j by scanning m steps back from v_i
- if found, replace by v_j, repeat (moving backwards)
- overall, the pointer moves backwards at most n times (since word length is reduced), by at most m steps each time
- it moves forwards n times plus at most nm (the steps it made backwards)

Open problem: logspace? Not all hyperbolic groups are linear (see [1])

Theorem (Muller-Schupp [20])

(G, X) *has context-free word problem iff* G *is virtually free*

virtually P = has a finite index subgroup with property P

Theorem (Muller-Schupp [20])

(G, X) *has context-free word problem iff G is virtually free*

virtually P = has a finite index subgroup with property P

See [9] for a nice discussion of this result.

An indexed grammar is

- nonterminals N, including a start symbol $S \in N$
- terminals A
- *flags* or *indices* F
- productions of the following types:
  - $A \longrightarrow v \in (A \cup N)^*$
  - $A_f \longrightarrow v \in (A \cup N)^*$
  - $A \longrightarrow A_f$

An indexed grammar is

- nonterminals N, including a start symbol $S \in N$
- terminals A
- *flags* or *indices* F
- productions of the following types:
  - $A \longrightarrow v \in (A \cup N)^*$
  - $A_f \longrightarrow v \in (A \cup N)^*$
  - $A \longrightarrow A_f$

Eg:

- $S \longrightarrow T_\$$
- $T \longrightarrow T_f \mid T_g \mid UU$
- $U_f \longrightarrow aU$
- $U_g \longrightarrow bU$
- $U_\$ \longrightarrow 1$

$$S \longrightarrow T_\$ \longrightarrow T_{\$f} \longrightarrow T_{\$fg} \longrightarrow T_{\$fgf} \longrightarrow U_{\$fgf}U_{\$fgf} \longrightarrow aU_{\$fg}aU_{\$fg}$$

Equivalent to indexed [2, 3].

Nested stack has a root node and a pointer which can read the current node

Moves:

- push (branch sideways if pointer not at a leaf)
- move up-down most recently formed branch of the stack (between the root and most recently added node)
- pop (only if pointer at a leaf)

Equivalent to indexed [2, 3].

Nested stack has a root node and a pointer which can read the current node

Moves:

- push (branch sideways if pointer not at a leaf)
- move up-down most recently formed branch of the stack (between the root and most recently added node)
- pop (only if pointer at a leaf)

Eg: $L = \{ww \mid w \in \{a, b\}^*\}$, $L = \{a^n b^n a^n \mid n \in \mathbb{N}\}$

Equivalent to indexed [2, 3].

Nested stack has a root node and a pointer which can read the current node

Moves:

- push (branch sideways if pointer not at a leaf)
- move up-down most recently formed branch of the stack (between the root and most recently added node)
- pop (only if pointer at a leaf)

Eg: $L = \{ww \mid w \in \{a,b\}^*\}$, $L = \{a^n b^n a^n \mid n \in \mathbb{N}\}$

Eg: $L = \{ab^{i_1}ab^{i_2}\dots ab^{i_n} \mid i_1 < i_2 < \cdots < i_n\}$ (intermediate growth [16])

Recall our failed attempt to accept the word problem for a hyperbolic group using a stack. With a nested stack we can read and append inside the stack

Recall our failed attempt to accept the word problem for a hyperbolic group using a stack. With a nested stack we can read and append inside the stack, but still we cannot replace u_i by v_i correctly.

Recall our failed attempt to accept the word problem for a hyperbolic group using a stack. With a nested stack we can read and append inside the stack, but still we cannot replace u_i by v_i correctly.

Conjecture: word problem indexed if and only if group is virtually free (so no advantage to using a nested stack)

Recall our failed attempt to accept the word problem for a hyperbolic group using a stack. With a nested stack we can read and append inside the stack, but still we cannot replace u_i by v_i correctly.

Conjecture: word problem indexed if and only if group is virtually free (so no advantage to using a nested stack)

Subconjecture: word problem for $\mathbb{Z}^2$ is not indexed.

Recall our failed attempt to accept the word problem for a hyperbolic group using a stack. With a nested stack we can read and append inside the stack, but still we cannot replace u_i by v_i correctly.

Conjecture: word problem indexed if and only if group is virtually free (so no advantage to using a nested stack)

Subconjecture: word problem for $\mathbb{Z}^2$ is not indexed.

Gilman and Shapiro: word problem accepted by deterministic limited erasing nested stack automaton if and only if virtually free

Gilman [15]

If $L \subseteq X^*$ is indexed and $m \in \mathbb{N}$, there is a constant $k > 0$ so that $|w| \geq k$ can be written as $w = w_1 \ldots w_r$

- $m < r \leq k$
- $|w_i| > 0$
- each choice of m factors is included in a proper subproduct which lies in L

Gilman [15]

If $L \subseteq X^*$ is indexed and $m \in \mathbb{N}$, there is a constant $k > 0$ so that $|w| \geq k$ can be written as $w = w_1 \ldots w_r$

- $m < r \leq k$
- $|w_i| > 0$
- each choice of m factors is included in a proper subproduct which lies in L

Eg: $L = \{(ab^n)^n \mid n \in \mathbb{N}\}$ not indexed:

Gilman [15]

If $L \subseteq X^*$ is indexed and $m \in \mathbb{N}$, there is a constant $k > 0$ so that $|w| \geq k$ can be written as $w = w_1 \dots w_r$

- $m < r \leq k$
- $|w_i| > 0$
- each choice of m factors is included in a proper subproduct which lies in L

Eg: $L = \{(ab^n)^n \mid n \in \mathbb{N}\}$ not indexed: $m = 1$, pick $(ab^n)^n$ with $n > k$, then some w_i must contain two or more a's, so proper subproduct has factor $ab^n a$.

Gilman [15]

If $L \subseteq X^*$ is indexed and $m \in \mathbb{N}$, there is a constant $k > 0$ so that $|w| \geq k$ can be written as $w = w_1 \ldots w_r$

- $m < r \leq k$
- $|w_i| > 0$
- each choice of m factors is included in a proper subproduct which lies in L

Eg: $L = \{(ab^n)^n \mid n \in \mathbb{N}\}$ not indexed: $m = 1$, pick $(ab^n)^n$ with $n > k$, then some w_i must contain two or more a's, so proper subproduct has factor ab^na.

Doesn't help with word problem of $\mathbb{Z}^2$ though

An *ET0L-system* is a tuple $(C, T, \Delta, \#)$ where

- C is a finite alphabet
- $T \subseteq C$
- $\# \in C$
- $\Delta = \{f_1, \ldots, f_n\}$ with each $f_i : C \longrightarrow \mathcal{P}(C^*) \setminus \emptyset$.

A table f_i acts on a word $w \in C^*$ as follows: if $w = x_1 \ldots x_k$ with $x_i \in C$, for each x_j we can choose any $u_j \in f_i(x_j)$ and replace x_j by u_j.

An *ET0L-system* is a tuple $(C, T, \Delta, \#)$ where

- C is a finite alphabet
- $T \subseteq C$
- $\# \in C$
- $\Delta = \{f_1, \ldots, f_n\}$ with each $f_i : C \longrightarrow \mathcal{P}(C^*) \setminus \emptyset$.

A table f_i acts on a word $w \in C^*$ as follows: if $w = x_1 \ldots x_k$ with $x_i \in C$, for each x_j we can choose any $u_j \in f_i(x_j)$ and replace x_j by u_j.

The *language* generated by the system is the set $L \subseteq T^*$ obtained by applying some word in Δ^* to $\#$ (and being left only with letters in T).

ET0L

An *ET0L-system* is a tuple $(C, T, \Delta, \#)$ where

- C is a finite alphabet
- $T \subseteq C$
- $\# \in C$
- $\Delta = \{f_1, \ldots, f_n\}$ with each $f_i : C \longrightarrow \mathcal{P}(C^*) \setminus \emptyset$.

A table f_i acts on a word $w \in C^*$ as follows: if $w = x_1 \ldots x_k$ with $x_i \in C$, for each x_j we can choose any $u_j \in f_i(x_j)$ and replace x_j by u_j.

The *language* generated by the system is the set $L \subseteq T^*$ obtained by applying some word in Δ^* to $\#$ (and being left only with letters in T).

Eg: $C = \{a, b, A, B, \#\}, T = \{a, b\}$, tables
$f_1 = \{(a, \{a\}), (b, \{b, AA\}), (A, \{aA, \epsilon\}), (B, \{AA, B\}), (\#, \{B, \#\})\}$

Eg: $C = \{a, b, \#\}, T = \{a, b\}$, tables

$$
\begin{aligned}
f_1 &= \{\{(a, \{a\}), (b, \{b\}), (\#, \{a\#a\#, \#\})\}, \\
f_2 &= \{\{(a, \{a\}), (b, \{b\}), (\#, \{b\#b\#, \#\})\}, \\
f_3 &= \{\{(a, \{a\}), (b, \{b\}), (\#, \{\epsilon, \#\})\},
\end{aligned}
$$

context-free $\subset$ ET0L $\subset$ indexed:

context-free: make every f_i include $\{(c, c) \mid c \in C\}$

context-free $\subset$ ET0L $\subset$ indexed:

context-free: make every f_i include $\{(c, c) \mid c \in C\}$

indexed:

- $S \notin C$, make production $S \longrightarrow \#_{\$}$
- make productions $S \longrightarrow S_{f_i}$ for each table
- then each derivation starts with $S \Rightarrow \#_{\$ f_{i_1} \dots f_{i_k}}$
- for each $c \in C, f_i \in \Delta$ make productions $c_{f_i} \longrightarrow w$ where $w \in f_i(c)$
- for each $t \in T$ make production $t_{\$} \longrightarrow t'$.

An ET0L system is *deterministic* (called EDT0L) if $|f_i(c)| = 1$ for each table and each $c \in C$, that is, there is no choice about how to rewrite different letters when you apply a table.

So for example, we cannot use the same trick to simulate context-free grammars

An ET0L system is *deterministic* (called EDT0L) if $|f_i(c)| = 1$ for each table and each $c \in C$, that is, there is no choice about how to rewrite different letters when you apply a table.

So for example, we cannot use the same trick to simulate context-free grammars

In fact, the word problem for free groups (of sufficiently high rank) is not EDT0L [12, 21]

An ET0L system is *deterministic* (called EDT0L) if $|f_i(c)| = 1$ for each table and each $c \in C$, that is, there is no choice about how to rewrite different letters when you apply a table.

So for example, we cannot use the same trick to simulate context-free grammars

In fact, the word problem for free groups (of sufficiently high rank) is not EDT0L [12, 21]

Eg: $L = \{ww \mid w \in \{a, b\}^*\}$ and $L = \{a^n b^n a^n \mid n \in \mathbb{N}\}$ are EDT0L.

An ET0L system is *deterministic* (called EDT0L) if $|f_i(c)| = 1$ for each table and each $c \in C$, that is, there is no choice about how to rewrite different letters when you apply a table.

So for example, we cannot use the same trick to simulate context-free grammars

In fact, the word problem for free groups (of sufficiently high rank) is not EDT0L [12, 21]

Eg: $L = \{ww \mid w \in \{a,b\}^*\}$ and $L = \{a^n b^n a^n \mid n \in \mathbb{N}\}$ are EDT0L.

Eg: $L = \{a^{2^n} \mid n \in \mathbb{N}\}$ is EDT0L

A language $L \subseteq X^*$ is *context-sensitive* if there is a linear space algorithm that decides which words are in L.

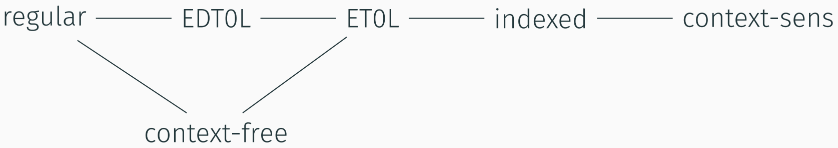
Equivalently, there is a grammar description.

A language $L \subseteq X^*$ is *context-sensitive* if there is a linear space algorithm that decides which words are in L.

Equivalently, there is a grammar description.

Shapiro [22] showed that every finitely generated subgroup of an automatic group is context-sensitive

— ie its a big class (includes group with undecidable conjugacy problem, non finitely presented etc.)


regular
EDT0L
ET0L
indexed
context-sens
context-free

A formal language class $\mathcal{C}$ is *closed under inverse homomorphism* if $L \subseteq X^*$ in $\mathcal{C}$ and $\varphi : Y^* \longrightarrow X^*$ a letter homomorphism ($\varphi(y) = u \in X^*$ for each $y \in Y$) implies $\varphi^{-1}(L)$ is in $\mathcal{C}$.

A formal language class $\mathcal{C}$ is *closed under inverse homomorphism* if $L \subseteq X^*$ in $\mathcal{C}$ and $\varphi : Y^* \longrightarrow X^*$ a letter homomorphism ($\varphi(y) = u \in X^*$ for each $y \in Y$) implies $\varphi^{-1}(L)$ is in $\mathcal{C}$.

Examples

- regular
- context-free
- indexed
- ET0L
- context-sensitive

A formal language class $\mathcal{C}$ is *closed under inverse homomorphism* if $L \subseteq X^*$ in $\mathcal{C}$ and $\varphi : Y^* \longrightarrow X^*$ a letter homomorphism ($\varphi(y) = u \in X^*$ for each $y \in Y$) implies $\varphi^{-1}(L)$ is in $\mathcal{C}$.

Examples

- regular
- context-free
- indexed
- ET0L
- context-sensitive

Non-examples: EDT0L: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is EDT0L and the homomorphic image of $\{w \in \{a, b\}^* \mid \exists n \in \mathbb{N}$ such that $|w|_a = 2^n\}$ which is not EDT0L [10, 11].

A formal language class $\mathcal{C}$ is *closed under inverse homomorphism* if $L \subseteq X^*$ in $\mathcal{C}$ and $\varphi : Y^* \longrightarrow X^*$ a letter homomorphism ($\varphi(y) = u \in X^*$ for each $y \in Y$) implies $\varphi^{-1}(L)$ is in $\mathcal{C}$.

Examples

- regular
- context-free
- indexed
- ET0L
- context-sensitive

Non-examples: EDT0L: $\{a^{2^n} \mid n \in \mathbb{N}\}$ is EDT0L and the homomorphic image of $\{w \in \{a,b\}^* \mid \exists n \in \mathbb{N} \text{ such that } |w|_a = 2^n\}$ which is not EDT0L [10, 11].

Fact: if a language class C is closed under inverse homomorphism and intersection with regular languages then (G, X) has word problem in C if and only if (G, Y) does

- regular word problem if and only if finite (Anisimov [4, 5])

WORD PROBLEM: SUMMARY

- regular word problem if and only if finite (Anisimov [4, 5])
- context-free if and only if virtually free (Muller-Schupp [20])

WORD PROBLEM: SUMMARY

- regular word problem if and only if finite (Anisimov [4, 5])
- context-free if and only if virtually free (Muller-Schupp [20])
- 1-counter if and only if virtually cyclic (Herbst [17])

WORD PROBLEM: SUMMARY

- regular word problem if and only if finite (Anisimov [4, 5])
- context-free if and only if virtually free (Muller-Schupp [20])
- 1-counter if and only if virtually cyclic (Herbst [17])
- blind k-counter if and only if virtually abelian (E-Kambites-Ostheimer [13])

WORD PROBLEM: SUMMARY

- regular word problem if and only if finite (Anisimov [4, 5])
- context-free if and only if virtually free (Muller-Schupp [20])
- 1-counter if and only if virtually cyclic (Herbst [17])
- blind k-counter if and only if virtually abelian (E-Kambites-Ostheimer [13])
- deterministic limited erasing nested stack automaton if and only if virtually free (Gilman-Shapiro [14])

WORD PROBLEM: SUMMARY

- regular word problem if and only if finite (Anisimov [4, 5])
- context-free if and only if virtually free (Muller-Schupp [20])
- 1-counter if and only if virtually cyclic (Herbst [17])
- blind k-counter if and only if virtually abelian (E-Kambites-Ostheimer [13])
- deterministic limited erasing nested stack automaton if and only if virtually free (Gilman-Shapiro [14])

Open problems:

- indexed conjecture: virtually free
- ET0L conjecture: virtually free
- EDT0L conjecture: finite

Recall: $L \subseteq T^*$ is EDT0L if there is $T \subseteq C \ni \#$ and a finite set of tables f_i that rewrite each $c \in C$ by $u_c \in C^*$ in parallel.

Recall: $L \subseteq T^*$ is EDT0L if there is $T \subseteq C \ni \#$ and a finite set of tables f_i that rewrite each $c \in C$ by $u_c \in C^*$ in parallel.

An alternative definition [6] restricts how the tables are applied.

Recall: $L \subseteq T^*$ is EDT0L if there is $T \subseteq C \ni \#$ and a finite set of tables f_i that rewrite each $c \in C$ by $u_c \in C^*$ in parallel.

An alternative definition [6] restricts how the tables are applied.

A language $L \subseteq A^*$ is EDT0L if $\exists$

- $C \supseteq A$ an extended alphabet (finite)
- $\# \in C$
- R regular set of endomorphisms $h : C^* \longrightarrow C^*$

so that $L = \{h(\#) \mid h \in R\}$.

Problem: on input

$$aXXb = YYbX$$

find substitutions for X, Y by words in a, b, a^{-1}, b^{-1} so that both sides are equal in the free group on $A_+ = \{a, b\}$.

Problem: on input

$$aXXb = YYbX$$

find substitutions for X, Y by words in a, b, a^{-1}, b^{-1} so that both sides are equal in the free group on $A_+ = \{a, b\}$.

How hard is the problem of finding all solutions to some equation:

Problem: on input

$$aXXb = YYbX$$

find substitutions for X, Y by words in a, b, a^{-1}, b^{-1} so that both sides are equal in the free group on $A_+ = \{a, b\}$.

How hard is the problem of finding all solutions to some equation:

- the complexity of the *Turing machine* (algorithm)

Problem: on input

$$aXXb = YYbX$$

find substitutions for X, Y by words in a, b, a^{-1}, b^{-1} so that both sides are equal in the free group on $A_+ = \{a, b\}$.

How hard is the problem of finding all solutions to some equation:

- the complexity of the *Turing machine* (algorithm)
- the complexity of the solution set in terms of *formal language theory*

Context-sensitive: input $(u_1, \ldots, u_k)$ of length $n = \sum |u_i|$, can decide

- u_i reduced words in $A_\pm$
- $X_i \longrightarrow u_i$ is a solution

in linear space (write the equation on the tape replacing X_i by u_i)

Context-sensitive: input $(u_1, \ldots, u_k)$ of length $n = \sum |u_i|$, can decide

- u_i reduced words in $A_\pm$
- $X_i \longrightarrow u_i$ is a solution

in linear space (write the equation on the tape replacing X_i by u_i)

But this does not tell us much — the *emptiness problem* for context-sensitive languages in undecidable [18]

Theorem (Ciobanu-Diekert-E [8])

For any equation over a free group, the set

$$\{(u_1, \ldots, u_k) \mid u_i \text{ reduced}, X_i \longrightarrow u_i \text{ is a solution}\}$$

is EDT0L.

Theorem (Ciobanu-Diekert-E [8])

For any equation over a free group, the set

$$\{(u_1, \ldots, u_k) \mid u_i \text{ reduced}, X_i \longrightarrow u_i \text{ is a solution}\}$$

is EDT0L.

More explicitly, let $n = |A_+| + |UV|$ where $U = V$ is an equation over a free group F_{A_+}.

Theorem (Ciobanu-Diekert-E [8])

For any equation over a free group, the set

$$\{(u_1, \ldots, u_k) \mid u_i \text{ reduced}, X_i \longrightarrow u_i \text{ is a solution}\}$$

is EDT0L.

More explicitly, let $n = |A_+| + |UV|$ where $U = V$ is an equation over a free group F_{A_+}.

We construct in NSPACE($n \log n$) a finite direct labeled graph where

- nodes are modified versions of the equation
- edges are labeled by letter homomorphisms
- every solution in reduced words is encoded by some path from an initial to final node in the graph.

Here is a naïve first attempt.

- Input: $XaYbaXa = bYb^3ZP$ equation in a free monoid.
- Guess the first letter of some variable, and replace. Eg: $Y \longrightarrow \overline{a}Y$.
- Guess $Y \longrightarrow 1$.
- Repeat. If there is a solution, this method will find it!

Here is a naïve first attempt.

- Input: $XaYbaXa = bYb^3ZP$ equation in a free monoid.
- Guess the first letter of some variable, and replace. Eg: $Y \longrightarrow \overline{a}Y$.
- Guess $Y \longrightarrow 1$.
- Repeat. If there is a solution, this method will find it!

Issues:

- if there is no solution, we will never stop.

Here is a naïve first attempt.

- Input: $XaYbaXa = bYb^3ZP$ equation in a free monoid.
- Guess the first letter of some variable, and replace. Eg: $Y \longrightarrow \overline{a}Y$.
- Guess $Y \longrightarrow 1$.
- Repeat. If there is a solution, this method will find it!

Issues:

- if there is no solution, we will never stop.
- $Y \longrightarrow \overline{a}Y$ increases the length of the equation (there is no cancellation) — can get arbitrarily long.

Here is a naïve first attempt.

- Input: $XaYbaXa = bYb^3ZP$ equation in a free monoid.
- Guess the first letter of some variable, and replace. Eg: $Y \longrightarrow \overline{a}Y$.
- Guess $Y \longrightarrow 1$.
- Repeat. If there is a solution, this method will find it!

Issues:

- if there is no solution, we will never stop.
- $Y \longrightarrow \overline{a}Y$ increases the length of the equation (there is no cancellation) — can get arbitrarily long.
- to ensure solutions are reduced words, need to keep track of letters popped out of variables

To keep the equation length bounded, we can try to *compress* constants using new constants. Eg: $ab \longrightarrow c$, $aa \longrightarrow d$, $aa \longrightarrow a$.

To keep the equation length bounded, we can try to *compress* constants using new constants. Eg: $ab \longrightarrow c$, $aa \longrightarrow d$, $aa \longrightarrow a$.

Issues:

- might need many new constants.

To keep the equation length bounded, we can try to *compress* constants using new constants. Eg: $ab \longrightarrow c$, $aa \longrightarrow d$, $aa \longrightarrow a$.

Issues:

- might need many new constants.
- $aa \longrightarrow d$ means that $ad = da$, so we are no longer in a free monoid.

To keep the equation length bounded, we can try to *compress* constants using new constants. Eg: $ab \longrightarrow c$, $aa \longrightarrow d$, $aa \longrightarrow a$.

Issues:

- might need many new constants.
- $aa \longrightarrow d$ means that $ad = da$, so we are no longer in a free monoid.
- $aa \longrightarrow a$ only works if all blocks of a have *even* length

Edges correspond to making one of the following moves on an equation

pop

- $X \longrightarrow aX,\ \overline{X} \longrightarrow \overline{X}\,\overline{a}$
- $X \longrightarrow 1,\ \overline{X} \longrightarrow 1$

split

- $X \longrightarrow X'X,\ \overline{X} \longrightarrow \overline{X}\,\overline{X'}$

compress

- $aa \longrightarrow a,\ \overline{a}\,\overline{a} \longrightarrow \overline{a}$
- $aa \longrightarrow c,\ \overline{a}\,\overline{a} \longrightarrow \overline{c}$
- $ab \longrightarrow c,\ \overline{b}\,\overline{a} \longrightarrow \overline{c}$

CONSTRAINTS

We need to introduce a symbol $\#$ that serves various purposes, but require it not appear in any solution $X_i \longrightarrow u$

We need to introduce a symbol $\#$ that serves various purposes, but require it not appear in any solution $X_i \longrightarrow u$

We ensure that solutions are in reduced words, and do not use $\#$, by using morphisms μ to a finite monoid.

We need to introduce a symbol $\#$ that serves various purposes, but require it not appear in any solution $X_i \longrightarrow u$

We ensure that solutions are in reduced words, and do not use $\#$, by using morphisms μ to a finite monoid.

Let $N = A_\pm \times A_\pm \cup \{0,1\}$ with multiplication

$$\begin{array}{l} 0 \cdot x = 0 = x \cdot 0 \\ 1 \cdot x = x = x \cdot 1 \end{array} \quad \text{and} \quad (a,b)\cdot(c,d) = \begin{cases} (a,d) & b \neq \overline{c} \\ 0 & b = \overline{c} \end{cases}$$

We need to introduce a symbol $\#$ that serves various purposes, but require it not appear in any solution $X_i \longrightarrow u$

We ensure that solutions are in reduced words, and do not use $\#$, by using morphisms μ to a finite monoid.

Let $N = A_\pm \times A_\pm \cup \{0, 1\}$ with multiplication

$$\begin{array}{l} 0 \cdot x = 0 = x \cdot 0 \\ 1 \cdot x = x = x \cdot 1 \end{array} \quad \text{and} \quad (a, b) \cdot (c, d) = \begin{cases} (a, d) & b \neq \overline{c} \\ 0 & b = \overline{c} \end{cases}$$

Then define the morphism $\mu : (A_\pm \cup \{\#\})^* \longrightarrow N$ by $\mu(a) = (a, a), \mu(1) = 1$ and $\mu(\#) = 0$.

If $u \in (A_{\pm} \cup \{\#\})^*$ then $\mu(u) = 0$ if u contains $\#$ or is not reduced, is 1 if and only if $u = 1$, and otherwise $\mu(u) = (a, b)$ where a, b are the first and last letters of u.

If $u \in (A_\pm \cup \{\#\})^*$ then $\mu(u) = 0$ if u contains $\#$ or is not reduced, is 1 if and only if $u = 1$, and otherwise $\mu(u) = (a, b)$ where a, b are the first and last letters of u.

Given an equation in $A_\pm, \#$ and variables X_i we *guess* the first and last letters of each X_i (or that $X_i \longrightarrow 1$) by guessing a value for $\mu(X_i)$. As we modify our equation (pop and compress moves), we make sure that the μ values are consistent, and updated as we pop and compress letters

If $u \in (A_\pm \cup \{\#\})^*$ then $\mu(u) = 0$ if u contains $\#$ or is not reduced, is 1 if and only if $u = 1$, and otherwise $\mu(u) = (a, b)$ where a, b are the first and last letters of u.

Given an equation in $A_\pm, \#$ and variables X_i we *guess* the first and last letters of each X_i (or that $X_i \longrightarrow 1$) by guessing a value for $\mu(X_i)$. As we modify our equation (pop and compress moves), we make sure that the μ values are consistent, and updated as we pop and compress letters

In this way, if there is a way to consistently find a solution following our procedure, then it is guaranteed to be in reduced words not using $\#$.

A key difference between our result and Makanin-Razborov diagrams is that we are able to ensure that solutions are in reduced words, ie or maps are simply letter homomorphisms (so EDT0L) not group homomorphisms, where controlling the form of solutions seems difficult,

A key difference between our result and Makanin-Razborov diagrams is that we are able to ensure that solutions are in reduced words, ie or maps are simply letter homomorphisms (so EDT0L) not group homomorphisms, where controlling the form of solutions seems difficult,

Note that we can obtain the set of all solutions as words over X^* (not necessarily reduced)

A key difference between our result and Makanin-Razborov diagrams is that we are able to ensure that solutions are in reduced words, ie or maps are simply letter homomorphisms (so EDT0L) not group homomorphisms, where controlling the form of solutions seems difficult,

Note that we can obtain the set of all solutions as words over X^* (not necessarily reduced)

by taking the EDT0L grammar giving reduced solutions, and combining it with the context-free grammar for the word problem of the free group over X.

A key difference between our result and Makanin-Razborov diagrams is that we are able to ensure that solutions are in reduced words, ie or maps are simply letter homomorphisms (so EDT0L) not group homomorphisms, where controlling the form of solutions seems difficult,

Note that we can obtain the set of all solutions as words over X^* (not necessarily reduced)

by taking the EDT0L grammar giving reduced solutions, and combining it with the context-free grammar for the word problem of the free group over X.

The result is ET0L (and not EDT0L for free groups of rank two or more).

Thank you

REFERENCES I

- http://www.math.utah.edu/~bestvina/eprints/questions.pdf, http://mathoverflow.net/questions/110208/understanding-groups-that-are-not-linear.

- A. V. Aho.
  Indexed grammars—an extension of context-free grammars.
  J. ACM, 15:647–671, 1968.

- A. V. Aho.
  Nested stack automata.
  J. ACM, 16:383–406, 1969.

A. V. Anisimov.
Group languages.
Kibernetika, 4/1971:18–24, 1971.
English translation in Cybernetics and Systems Analysis 4 (1973), 594-601.

A. V. Anisimov and F. D. Seifert.
Zur algebraischen Charakteristik der durch kontext-freie Sprachen definierten Gruppen.
Elektron. Informationsverarbeit. Kybernetik, 11(10–12):695–702, 1975.

P. R. Asveld.
Controlled iteration grammars and full hyper-AFL's.
Information and control, 34, 1977.

J. W. Cannon.
The combinatorial structure of cocompact discrete hyperbolic groups.
Geom. Dedicata, 16(2):123–148, 1984.

L. Ciobanu, V. Diekert, and M. Elder.
Solution sets for equations over free groups are EDT0L languages.
In *Automata, languages, and programming. Part II*, volume 9135 of *Lecture Notes in Comput. Sci.*, pages 134–145. Springer, Heidelberg, 2015.

V. Diekert and A. Weiß.
Context-free groups and their structure trees.
Internat. J. Algebra Comput., 23(3):611–642, 2013.

A. Ehrenfeucht and G. Rozenberg.
The number of occurrences of letters versus their distribution in some EOL languages.
Information and Control, 26:256–271, 1974.

A. Ehrenfeucht and G. Rozenberg.
On inverse homomorphic images of deterministic ETOL languages.
In *Automata, languages, development*, pages 179–189. North-Holland, Amsterdam, 1976.

A. Ehrenfeucht and G. Rozenberg.
On some context free languages that are not deterministic ETOL languages.
RAIRO Informat. Théor., 11(4):273–291, i, 1977.

M. Elder, M. Kambites, and G. Ostheimer.
On groups and counter automata.
Internat. J. Algebra Comput., 18(8):1345–1364, 2008.

R. Gilman and M. Shapiro.
On groups whose word problem is solved by a nested stack automaton.
1998; http://arxiv.org/abs/math/9812028v1.

R. H. Gilman.
A shrinking lemma for indexed languages.
Theoret. Comput. Sci., 163(1-2):277–281, 1996.

R. I. Grigorchuk and A. Machì.
An example of an indexed language of intermediate growth.
Theoret. Comput. Sci., 215(1-2):325–327, 1999.

T. Herbst.
On a subclass of context-free groups.
Informatique Theorique et Applications, 25:255–272, 1991.

J. E. Hopcroft and J. D. Ullman.
Introduction to automata theory, languages, and computation.
Addison-Wesley Publishing Co., Reading, Mass., 1979.
Addison-Wesley Series in Computer Science.

R. J. Lipton and Y. Zalcstein.
Word problems solvable in logspace.
J. Assoc. Comput. Mach., 24(3):522–526, 1977.

D. E. Muller and P. E. Schupp.
Groups, the theory of ends, and context-free languages.
J. Comput. System Sci., 26(3):295–310, 1983.

B. Rozoy.
The Dyck language D_i^* is not generated by any matrix grammar of finite index.
Information and Computation, 74:64–89, 1987.

M. Shapiro.
A note on context-sensitive languages and word problems.
Internat. J. Algebra Comput., 4(4):493–497, 1994.