Finitely presented groups 4

Max Neunhöffer

LMS Short Course on Computational Group Theory
29 July – 2 August 2013

Let $G := \langle X \mid R \rangle$ with $\hat{X} := X \cup X^{-1}$. Assume that R is closed under rotation and inversion and all $r \in R$ are reduced.

Let $G := \langle X \mid R \rangle$ with $\hat{X} := X \cup X^{-1}$. Assume that R is closed under rotation and inversion and all $r \in R$ are reduced.

Definition (Piece)

A piece (w.r.t. R) is a nonempty word p that is a prefix of two different relators, i.e.: $pa, pb \in R$ for $a, b \in \hat{X}^*$ with $a \neq b$.

Let $G := \langle X \mid R \rangle$ with $\hat{X} := X \cup X^{-1}$. Assume that R is closed under rotation and inversion and all $r \in R$ are reduced.

Definition (Piece)

A piece (w.r.t. R) is a nonempty word p that is a prefix of two different relators, i.e.: $pa, pb \in R$ for $a, b \in \hat{X}^*$ with $a \neq b$.

Definition (Condition $C'(\lambda)$)

We say $\langle X \mid R \rangle$ is $C'(\lambda)$, if:

- for all $r = pa \in R$ where p is a piece, we have $|p| < \lambda \cdot |r|$.

($|r|$ is the length in letters).

Let $G := \langle X \mid R \rangle$ with $\hat{X} := X \cup X^{-1}$. Assume that R is closed under rotation and inversion and all $r \in R$ are reduced.

Definition (Piece)

A piece (w.r.t. R) is a nonempty word p that is a prefix of two different relators, i.e.: $pa, pb \in R$ for $a, b \in \hat{X}^*$ with $a \neq b$.

Definition (Condition $C'(\lambda)$)

We say $\langle X \mid R \rangle$ is $C'(\lambda)$, if:

- for all $r = pa \in R$ where p is a piece, we have $|p| < \lambda \cdot |r|$.

($|r|$ is the length in letters).

Definition (Condition $T(q)$)

We say $\langle X \mid R \rangle$ is $T(q)$, if the following holds:

- Let $3 \leq h < q$ and $(r_1, r_2, \ldots, r_h) \in R^h$ with no successive elements r_i, r_{i+1} or r_h, r_1 an inverse pair. Then at least one of the products $r_1 r_2, r_2 r_3, \ldots, r_h r_1$ is reduced without cancellation.

Theorem (Lyndon, Schupp)

Let $G = \langle X \mid R \rangle$ with R closed under rotation and inversion and all $r \in R$ are reduced. If $\langle X \mid R \rangle$ fulfills at least one of:

- *$C'(1/6)$ and $T(3)$, or*
- *$C'(1/4)$ and $T(4)$, or*
- *$C'(1/3)$ and $T(6)$,*

then Dehn's algorithm solves the word problem for G.

Theorem (Lyndon, Schupp)

Let $G = \langle X \mid R \rangle$ with R closed under rotation and inversion and all $r \in R$ are reduced. If $\langle X \mid R \rangle$ fulfills at least one of:

- *$C'(1/6)$ and $T(3)$, or*
- *$C'(1/4)$ and $T(4)$, or*
- *$C'(1/3)$ and $T(6)$,*

then Dehn's algorithm solves the word problem for G.

What is Dehn's algorithm?

Theorem (Lyndon, Schupp)

Let $G = \langle X \mid R \rangle$ with R closed under rotation and inversion and all $r \in R$ are reduced. If $\langle X \mid R \rangle$ fulfills at least one of:

- *$C'(1/6)$ and $T(3)$, or*
- *$C'(1/4)$ and $T(4)$, or*
- *$C'(1/3)$ and $T(6)$,*

then Dehn's algorithm solves the word problem for G.

What is Dehn's algorithm?
What does this mean for the structure of G?

Theorem (Lyndon, Schupp)

Let $G = \langle X \mid R \rangle$ with R closed under rotation and inversion and all $r \in R$ are reduced. If $\langle X \mid R \rangle$ fulfills at least one of:

- *$C'(1/6)$ and $T(3)$, or*
- *$C'(1/4)$ and $T(4)$, or*
- *$C'(1/3)$ and $T(6)$,*

then Dehn's algorithm solves the word problem for G.

What is Dehn's algorithm?
What does this mean for the structure of G?

Definition (Dehn RWS)

Write all $r \in R$ as $r = ab$ with $|a| > |b|$ and define a rule $a \to b^{-1}$.

Algorithm (Dehn's algorithm)

Let $G = \langle X \mid R \rangle$ and let $\mathcal{R}$ be a length-reducing RWS for $\hat{X} = X \cup X^{-1}$.

1. **Input:** a word $w \in \hat{X}^*$.
2. Freely reduce w.
3. If any rewrite rule matches, apply it and go back to 2.
4. **Output:** the new w.

Algorithm (Dehn's algorithm)

Let $G = \langle X \mid R \rangle$ and let $\mathcal{R}$ be a length-reducing RWS for $\hat{X} = X \cup X^{-1}$.

1. **Input:** a word $w \in \hat{X}^*$.
2. Freely reduce w.
3. If any rewrite rule matches, apply it and go back to 2.
4. **Output:** the new w.

Note that 3. is not deterministic.

Algorithm (Dehn's algorithm)

Let $G = \langle X \mid R \rangle$ and let $\mathcal{R}$ be a length-reducing RWS for $\hat{X} = X \cup X^{-1}$.

1. **Input:** a word $w \in \hat{X}^*$.
2. Freely reduce w.
3. If any rewrite rule matches, apply it and go back to 2.
4. **Output:** the new w.

Note that 3. is not deterministic.

Saying that "Dehn's algorithm solves the word problem" means:

- The output is the empty word ε if and only if $w =_G 1$,
- not depending on which rewrite is applied in 3.

Algorithm (Dehn's algorithm)

Let $G = \langle X \mid R \rangle$ and let $\mathcal{R}$ be a length-reducing RWS for $\hat{X} = X \cup X^{-1}$.

1. **Input:** a word $w \in \hat{X}^*$.
2. Freely reduce w.
3. If any rewrite rule matches, apply it and go back to 2.
4. **Output:** the new w.

Note that 3. is not deterministic.

Saying that "Dehn's algorithm solves the word problem" means:

- The output is the empty word ε if and only if $w =_G 1$,
- not depending on which rewrite is applied in 3.

Note:

- For a general RWS, this does not make sense at all.

Algorithm (Dehn's algorithm)

Let $G = \langle X \mid R \rangle$ and let $\mathcal{R}$ be a length-reducing RWS for $\hat{X} = X \cup X^{-1}$.

1. **Input:** a word $w \in \hat{X}^*$.
2. Freely reduce w.
3. If any rewrite rule matches, apply it and go back to 2.
4. **Output:** the new w.

Note that 3. is not deterministic.

Saying that "Dehn's algorithm solves the word problem" means:

- The output is the empty word ε if and only if $w =_G 1$,
- not depending on which rewrite is applied in 3.

Note:

- For a general RWS, this does not make sense at all.
- If $w \neq_G 1$, then the output can be different, depending on the choice in 3.

Algorithm (Dehn's algorithm)

Let $G = \langle X \mid R \rangle$ and let $\mathcal{R}$ be a length-reducing RWS for $\hat{X} = X \cup X^{-1}$.

1. **Input:** a word $w \in \hat{X}^*$.
2. Freely reduce w.
3. If any rewrite rule matches, apply it and go back to 2.
4. **Output:** the new w.

Note that 3. is not deterministic.

Saying that "Dehn's algorithm solves the word problem" means:

- The output is the empty word ε if and only if $w =_G 1$,
- not depending on which rewrite is applied in 3.

Note:

- For a general RWS, this does not make sense at all.
- If $w \neq_G 1$, then the output can be different, depending on the choice in 3.
- For a word of length n, this terminates after at most n steps.

If $ab \in R$ with $|a| > |b|$ and $w = xay$, then Dehn rewrites this to $xb^{-1}y$.

If $ab \in R$ with $|a| > |b|$ and $w = xay$, then Dehn rewrites this to $xb^{-1}y$.
Thus: $w = x(ab)x^{-1}\ xb^{-1}y$

If $ab \in R$ with $|a| > |b|$ and $w = xay$, then Dehn rewrites this to $xb^{-1}y$.
Thus: $w = x(ab)x^{-1} \, xb^{-1}y$
So w is written as a conjugate of a relator times a shorter word.

Lemma

If $G = \langle X \mid R \rangle$ is small cancellation, then Dehn works and every word $w \in \hat{X}^*$ of length n that is equal to 1 in G is the product of at most n conjugates of a relator. Thus, the Dehn function $\delta(n) \leq n$ for all n.

If $ab \in R$ with $|a| > |b|$ and $w = xay$, then Dehn rewrites this to $xb^{-1}y$.
Thus: $w = x(ab)x^{-1}\, xb^{-1}y$
So w is written as a conjugate of a relator times a shorter word.

Lemma

If $G = \langle X \mid R \rangle$ is small cancellation, then Dehn works and every word $w \in \hat{X}^*$ of length n that is equal to 1 in G is the product of at most n conjugates of a relator. Thus, the Dehn function $\delta(n) \leq n$ for all n.

Definition (Hyperbolic group)

A group is called hyperbolic, if it has a finite presentation with a Dehn function that is bounded by a linear function.

If $ab \in R$ with $|a| > |b|$ and $w = xay$, then Dehn rewrites this to $xb^{-1}y$.
Thus: $w = x(ab)x^{-1}\ xb^{-1}y$
So w is written as a conjugate of a relator times a shorter word.

Lemma

If $G = \langle X \mid R \rangle$ is small cancellation, then Dehn works and every word $w \in \hat{X}^*$ of length n that is equal to 1 in G is the product of at most n conjugates of a relator. Thus, the Dehn function $\delta(n) \leq n$ for all n.

Definition (Hyperbolic group)

A group is called hyperbolic, if it has a finite presentation with a Dehn function that is bounded by a linear function.

We have for a group $G = \langle X \mid R \rangle$:

$$\text{small cancellation} \implies \text{Dehn's algorithm works} \implies \text{hyperbolic}$$

If $ab \in R$ with $|a| > |b|$ and $w = xay$, then Dehn rewrites this to $xb^{-1}y$.
Thus: $w = x(ab)x^{-1} \, xb^{-1}y$
So w is written as a conjugate of a relator times a shorter word.

Lemma

If $G = \langle X \mid R \rangle$ is small cancellation, then Dehn works and every word $w \in \hat{X}^*$ of length n that is equal to 1 in G is the product of at most n conjugates of a relator. Thus, the Dehn function $\delta(n) \leq n$ for all n.

Definition (Hyperbolic group)

A group is called hyperbolic, if it has a finite presentation with a Dehn function that is bounded by a linear function.

We have for a group $G = \langle X \mid R \rangle$:

$$\begin{aligned} \text{small cancellation} &\implies \text{Dehn's algorithm works} \implies \text{hyperbolic} \\ &\implies \text{has presentation with a working Dehn} \end{aligned}$$

If $ab \in R$ with $|a| > |b|$ and $w = xay$, then Dehn rewrites this to $xb^{-1}y$.
Thus: $w = x(ab)x^{-1}\ xb^{-1}y$
So w is written as a conjugate of a relator times a shorter word.

Lemma

If $G = \langle X \mid R \rangle$ is small cancellation, then Dehn works and every word $w \in \hat{X}^*$ of length n that is equal to 1 in G is the product of at most n conjugates of a relator. Thus, the Dehn function $\delta(n) \leq n$ for all n.

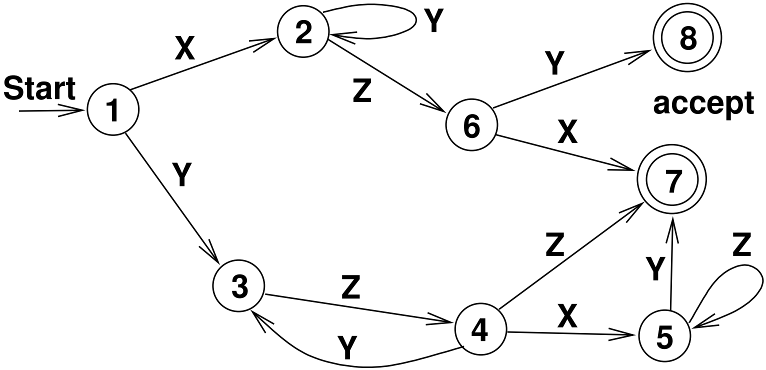
Definition (Hyperbolic group)

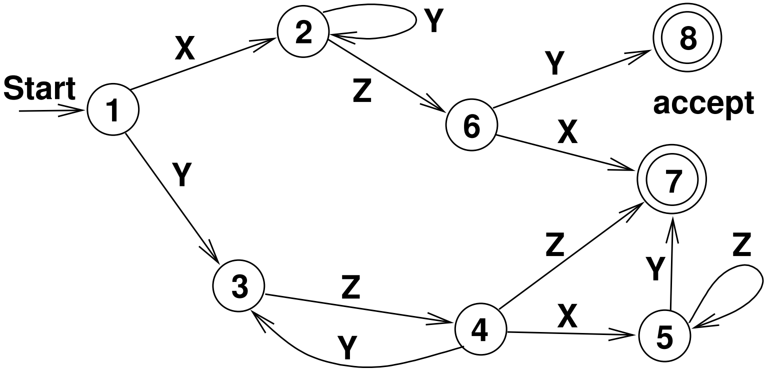
A group is called hyperbolic, if it has a finite presentation with a Dehn function that is bounded by a linear function.

We have for a group $G = \langle X \mid R \rangle$:

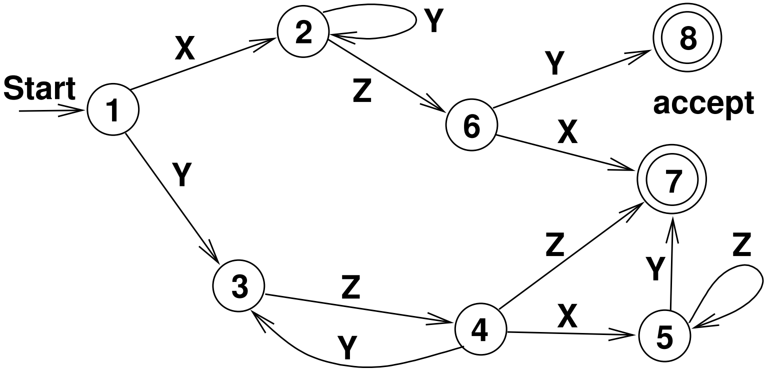
small cancellation $\Longrightarrow$ Dehn's algorithm works $\Longrightarrow$ hyperbolic
$\Longrightarrow$ has presentation with a working Dehn

Question: How do we execute Dehn's algorithm efficiently?

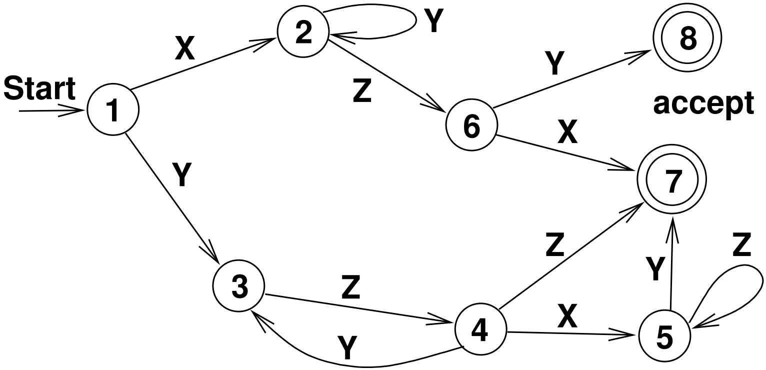

Start
1
2
3
4
5
6
7
8
accept
X
Y
Z
Y
X
Y
Y
Z
Z
Y
X
Y
Z
Z



Every path in this digraph has a label.

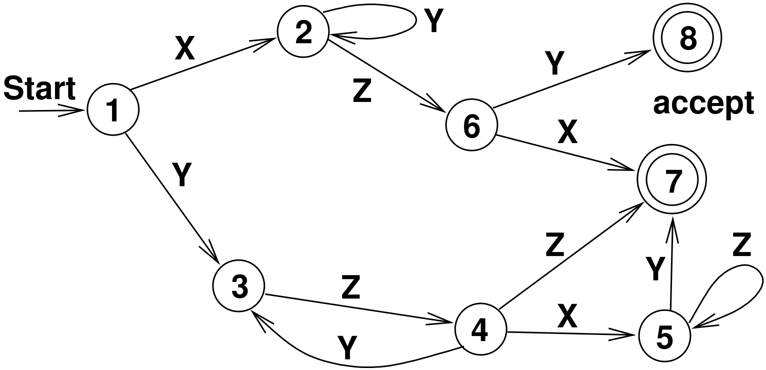


Every path in this digraph has a label.
There is one start state and some accept states.



Every path in this digraph has a label.
There is one start state and some accept states.

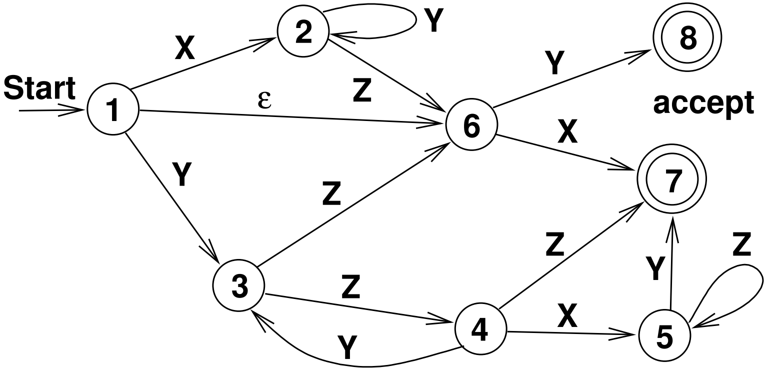
$$\mathcal{L} := \{\text{labels of paths from start to an accept state}\} \subseteq \{X, Y, Z\}^*$$



Every path in this digraph has a label.
There is one start state and some accept states.

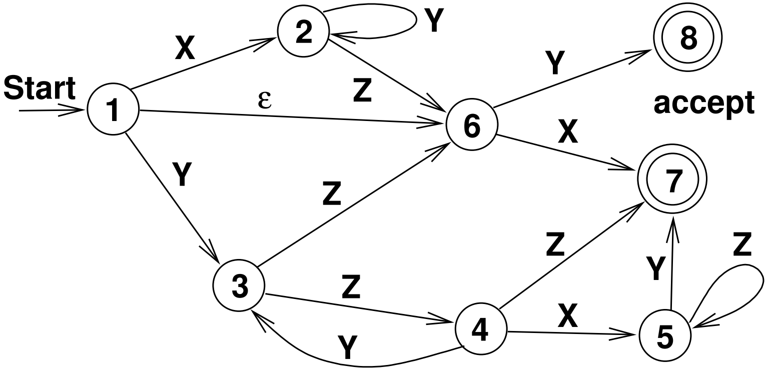
$$\mathcal{L} := \{\text{labels of paths from start to an accept state}\} \subseteq \{X, Y, Z\}^*$$

This is a regular language: $XY^*Z(Y + X) + YZ(YZ)^*(Z + XZ^*Y)$.



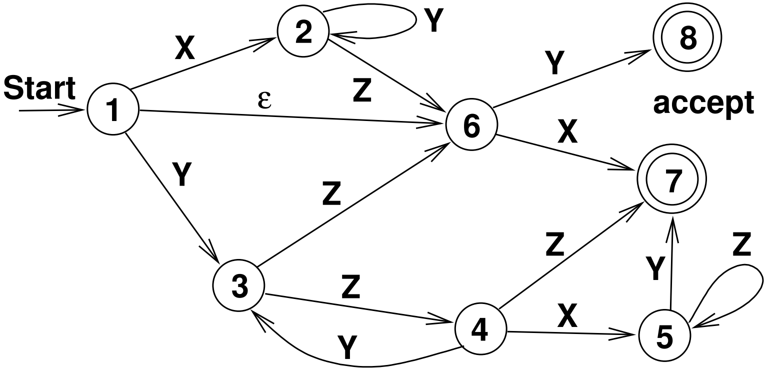
Non-deterministic variants:

- Allow empty (or ε) transitions.



Non-deterministic variants:

- Allow empty (or ε) transitions.
- Allow more than one transition with the same label leaving a state.



Non-deterministic variants:

- Allow empty (or ε) transitions.
- Allow more than one transition with the same label leaving a state.

However: The classes of languages of deterministic and non-deterministic finite state automata are the same.

Assume $\mathcal{R}$ is a RWS and assume for simplicity that no left hand side (LHS) of a rewrite is properly contained in another one.

Assume $\mathcal{R}$ is a RWS and assume for simplicity that no left hand side (LHS) of a rewrite is properly contained in another one.

Definition (FSA for a RWS)

States:
Define a state for every prefix of a LHS of a rewrite.

Assume $\mathcal{R}$ is a RWS and assume for simplicity that no left hand side (LHS) of a rewrite is properly contained in another one.

Definition (FSA for a RWS)

States:

Define a state for every prefix of a LHS of a rewrite.
The empty prefix is the start state.

Assume $\mathcal{R}$ is a RWS and assume for simplicity that no left hand side (LHS) of a rewrite is properly contained in another one.

Definition (FSA for a RWS)

States:
Define a state for every prefix of a LHS of a rewrite.
The empty prefix is the start state.
The complete LHSs are the accept states.

Assume $\mathcal{R}$ is a RWS and assume for simplicity that no left hand side (LHS) of a rewrite is properly contained in another one.

Definition (FSA for a RWS)

States:

Define a state for every prefix of a LHS of a rewrite.
The empty prefix is the start state.
The complete LHSs are the accept states.

Transitions:

If XY is a non-accepting state, then there is a transition labelled with "Z" to XYZ if this is still a prefix of a LHS.

Assume $\mathcal{R}$ is a RWS and assume for simplicity that no left hand side (LHS) of a rewrite is properly contained in another one.

Definition (FSA for a RWS)

States:
Define a state for every prefix of a LHS of a rewrite.
The empty prefix is the start state.
The complete LHSs are the accept states.

Transitions:
If XY is a non-accepting state, then there is a transition labelled with "Z" to XYZ if this is still a prefix of a LHS.
If XYZ is not a prefix, then there is a transition labelled with "Z" to the longest suffix of XYZ that is a prefix of a LHS.

Assume $\mathcal{R}$ is a RWS and assume for simplicity that no left hand side (LHS) of a rewrite is properly contained in another one.

Definition (FSA for a RWS)

States:

Define a state for every prefix of a LHS of a rewrite.
The empty prefix is the start state.
The complete LHSs are the accept states.

Transitions:

If XY is a non-accepting state, then there is a transition labelled with "Z" to XYZ if this is still a prefix of a LHS.
If XYZ is not a prefix, then there is a transition labelled with "Z" to the longest suffix of XYZ that is a prefix of a LHS.

This defines a deterministic FSA which recognises LHSs.

Assume $\mathcal{R}$ is a RWS and assume for simplicity that no left hand side (LHS) of a rewrite is properly contained in another one.

Definition (FSA for a RWS)

States:

Define a state for every prefix of a LHS of a rewrite.
The empty prefix is the start state.
The complete LHSs are the accept states.

Transitions:

If XY is a non-accepting state, then there is a transition labelled with "Z" to XYZ if this is still a prefix of a LHS.
If XYZ is not a prefix, then there is a transition labelled with "Z" to the longest suffix of XYZ that is a prefix of a LHS.

This defines a deterministic FSA which recognises LHSs.

$\Longrightarrow$ Very fast algorithm to recognise rewrite rules that apply.

Assume $\mathcal{R}$ is a RWS and assume for simplicity that no left hand side (LHS) of a rewrite is properly contained in another one.

Definition (FSA for a RWS)

States:

Define a state for every prefix of a LHS of a rewrite.
The empty prefix is the start state.
The complete LHSs are the accept states.

Transitions:

If XY is a non-accepting state, then there is a transition labelled with "Z" to XYZ if this is still a prefix of a LHS.
If XYZ is not a prefix, then there is a transition labelled with "Z" to the longest suffix of XYZ that is a prefix of a LHS.

This defines a deterministic FSA which recognises LHSs.

$\Longrightarrow$ Very fast algorithm to recognise rewrite rules that apply.
$\Longrightarrow$ Crucial step for Dehn's algorithm.

Sometimes, we want to describe relations on $\hat{X}^*$ by a FSA:

Sometimes, we want to describe relations on $\hat{X}^*$ by a FSA:

Definition (2-variable FSA by padding)

Define $p : \hat{X}^* \times \hat{X}^* \to ((\hat{X} \cup \{\$\}) \times (\hat{X} \cup \{\$\}))^*$ by padding the shorter word at the end with \$ symbols:

$$\begin{aligned} p(ABC, DEFGH) &= (A, D)(B, E)(C, F)(\$, G)(\$, H) \\ p(ABC, D) &= (A, D)(B, \$)(C, \$) \end{aligned}$$

Sometimes, we want to describe relations on $\hat{X}^*$ by a FSA:

Definition (2-variable FSA by padding)

Define $p : \hat{X}^* \times \hat{X}^* \to ((\hat{X} \cup \{\$\}) \times (\hat{X} \cup \{\$\}))^*$ by padding the shorter word at the end with \$ symbols:

$$\begin{aligned} p(ABC, DEFGH) &= (A, D)(B, E)(C, F)(\$, G)(\$, H) \\ p(ABC, D) &= (A, D)(B, \$)(C, \$) \end{aligned}$$

A FSA with alphabet $\hat{X} \cup \{\$\}$ accepts a pair $(v, w) \in \hat{X}^* \times \hat{X}^*$ iff there is a path from the start state to an accept state with label $p(v, w)$.

Sometimes, we want to describe relations on $\hat{X}^*$ by a FSA:

Definition (2-variable FSA by padding)

Define $p : \hat{X}^* \times \hat{X}^* \to ((\hat{X} \cup \{\$\}) \times (\hat{X} \cup \{\$\}))^*$ by padding the shorter word at the end with \$ symbols:

$$\begin{aligned} p(ABC, DEFGH) &= (A, D)(B, E)(C, F)(\$, G)(\$, H) \\ p(ABC, D) &= (A, D)(B, \$)(C, \$) \end{aligned}$$

A FSA with alphabet $\hat{X} \cup \{\$\}$ accepts a pair $(v, w) \in \hat{X}^* \times \hat{X}^*$ iff there is a path from the start state to an accept state with label $p(v, w)$.

We prepare ourselves for the definition of automatic groups:

Sometimes, we want to describe relations on $\hat{X}^*$ by a FSA:

Definition (2-variable FSA by padding)

Define $p : \hat{X}^* \times \hat{X}^* \to ((\hat{X} \cup \{\$\}) \times (\hat{X} \cup \{\$\}))^*$ by padding the shorter word at the end with \$ symbols:

$$\begin{aligned} p(ABC, DEFGH) &= (A, D)(B, E)(C, F)(\$, G)(\$, H) \\ p(ABC, D) &= (A, D)(B, \$)(C, \$) \end{aligned}$$

A FSA with alphabet $\hat{X} \cup \{\$\}$ accepts a pair $(v, w) \in \hat{X}^* \times \hat{X}^*$ iff there is a path from the start state to an accept state with label $p(v, w)$.

We prepare ourselves for the definition of automatic groups:

Definition (Word acceptor)

Let $G = \langle X \mid R \rangle$ and $\hat{X} := X \cup X^{-1}$. A FSA on $\hat{X}$ is called a word acceptor for G, if it accepts at least one word for each element of G.

Sometimes, we want to describe relations on $\hat{X}^*$ by a FSA:

Definition (2-variable FSA by padding)

Define $p : \hat{X}^* \times \hat{X}^* \to ((\hat{X} \cup \{\$\}) \times (\hat{X} \cup \{\$\}))^*$ by padding the shorter word at the end with \$ symbols:

$$\begin{aligned} p(ABC, DEFGH) &= (A, D)(B, E)(C, F)(\$, G)(\$, H) \\ p(ABC, D) &= (A, D)(B, \$)(C, \$) \end{aligned}$$

A FSA with alphabet $\hat{X} \cup \{\$\}$ accepts a pair $(v, w) \in \hat{X}^* \times \hat{X}^*$ iff there is a path from the start state to an accept state with label $p(v, w)$.

We prepare ourselves for the definition of automatic groups:

Definition (Word acceptor)

Let $G = \langle X \mid R \rangle$ and $\hat{X} := X \cup X^{-1}$. A FSA on $\hat{X}$ is called a word acceptor for G, if it accepts at least one word for each element of G. It is called a unique word acceptor, if it accepts exactly one word for each element of G.

Definition (Automatic group)

Let G be a group that is generated as a monoid by the set $\hat{X}$. Then G is automatic w.r.t. $\hat{X}$, if there exist FSA W and M_x for $x \in \hat{X} \cup \{\varepsilon\}$, s.th.:

- W has alphabet $\hat{X}$ and is a word acceptor for G, and

Definition (Automatic group)

Let G be a group that is generated as a monoid by the set $\hat{X}$. Then G is automatic w.r.t. $\hat{X}$, if there exist FSA W and M_x for $x \in \hat{X} \cup \{\varepsilon\}$, s.th.:

- W has alphabet $\hat{X}$ and is a word acceptor for G, and
- M_x has alphabet $\hat{X} \cup \{\$\}$ and $(v, w) \in \hat{X}^* \times \hat{X}^*$ (where v and w are accepted by W) is accepted by M_x iff $vx =_G w$.

Definition (Automatic group)

Let G be a group that is generated as a monoid by the set $\hat{X}$. Then G is automatic w.r.t. $\hat{X}$, if there exist FSA W and M_x for $x \in \hat{X} \cup \{\varepsilon\}$, s.th.:

- W has alphabet $\hat{X}$ and is a word acceptor for G, and
- M_x has alphabet $\hat{X} \cup \{\$\}$ and $(v, w) \in \hat{X}^* \times \hat{X}^*$ (where v and w are accepted by W) is accepted by M_x iff $vx =_G w$.

The automata W and M_x are called an automatic structure for G, the M_x are the multiplier automata.

Definition (Automatic group)

Let G be a group that is generated as a monoid by the set $\hat{X}$. Then G is automatic w.r.t. $\hat{X}$, if there exist FSA W and M_x for $x \in \hat{X} \cup \{\varepsilon\}$, s.th.:

- W has alphabet $\hat{X}$ and is a word acceptor for G, and
- M_x has alphabet $\hat{X} \cup \{\$\}$ and $(v, w) \in \hat{X}^* \times \hat{X}^*$ (where v and w are accepted by W) is accepted by M_x iff $vx =_G w$.

The automata W and M_x are called an automatic structure for G, the M_x are the multiplier automata.

Theorem (Epstein et al. 1992)

Being automatic is a property of G and not of $\hat{X}$.

Definition (Automatic group)

Let G be a group that is generated as a monoid by the set $\hat{X}$. Then G is automatic w.r.t. $\hat{X}$, if there exist FSA W and M_x for $x \in \hat{X} \cup \{\varepsilon\}$, s.th.:

- W has alphabet $\hat{X}$ and is a word acceptor for G, and
- M_x has alphabet $\hat{X} \cup \{\$\}$ and $(v, w) \in \hat{X}^* \times \hat{X}^*$ (where v and w are accepted by W) is accepted by M_x iff $vx =_G w$.

The automata W and M_x are called an automatic structure for G, the M_x are the multiplier automata.

Theorem (Epstein et al. 1992)

Being automatic is a property of G and not of $\hat{X}$.

Definition (Shortlex automatic structure)

If W accepts precisely the shortlex minimal words of $\hat{X}^*$ for the elements of G, then $(W, \{M_x\})$ is a shortlex automatic structure.

Definition (Word differences)

Let $v, w \in \hat{X}^*$ and let v_i be the prefix of v of length i. The word differences of v and w are $D(v, w) := \{v_i^{-1} w_i \mid i \in \mathbb{N}\} \subseteq G$.

Definition (Word differences)

Let $v, w \in \hat{X}^*$ and let v_i be the prefix of v of length i. The word differences of v and w are $D(v, w) := \{v_i^{-1} w_i \mid i \in \mathbb{N}\} \subseteq G$.
Note that all $D(v, w)$ are finite sets.

Definition (Word differences)

Let $v, w \in \hat{X}^*$ and let v_i be the prefix of v of length i. The word differences of v and w are $D(v, w) := \{v_i^{-1} w_i \mid i \in \mathbb{N}\} \subseteq G$.
Note that all $D(v, w)$ are finite sets.

Theorem

Let $(W, \{M_x\})$ be an automatic structure. The set

$$D := \bigcup_{(v,w) \text{ accepted by some } M_x} D(v, w)$$

is finite.

Idea of shortlex automatic structure computation

Let $G = \langle X \mid R \rangle$ and set $\hat{X} := X \cup X^{-1}$.

1. Run a shortlex Knuth-Bendix on a RWS coming from the monoid presentation.

Idea of shortlex automatic structure computation

Let $G = \langle X \mid R \rangle$ and set $\hat{X} := X \cup X^{-1}$.

1. Run a shortlex Knuth-Bendix on a RWS coming from the monoid presentation.
2. Stop after some time, even if it has not completed.

Idea of shortlex automatic structure computation

Let $G = \langle X \mid R \rangle$ and set $\hat{X} := X \cup X^{-1}$.

1. Run a shortlex Knuth-Bendix on a RWS coming from the monoid presentation.
2. Stop after some time, even if it has not completed.
3. Compute word differences as above, and approximate FSA to recognise them.

Idea of shortlex automatic structure computation

Let $G = \langle X \mid R \rangle$ and set $\hat{X} := X \cup X^{-1}$.

1. Run a shortlex Knuth-Bendix on a RWS coming from the monoid presentation.
2. Stop after some time, even if it has not completed.
3. Compute word differences as above, and approximate FSA to recognise them.
4. Compute a candidate for the word acceptor W.

Idea of shortlex automatic structure computation

Let $G = \langle X \mid R \rangle$ and set $\hat{X} := X \cup X^{-1}$.

1. Run a shortlex Knuth-Bendix on a RWS coming from the monoid presentation.
2. Stop after some time, even if it has not completed.
3. Compute word differences as above, and approximate FSA to recognise them.
4. Compute a candidate for the word acceptor W.
5. Compute candidates for the multiplier FSA M_x.

Idea of shortlex automatic structure computation

Let $G = \langle X \mid R \rangle$ and set $\hat{X} := X \cup X^{-1}$.

1. Run a shortlex Knuth-Bendix on a RWS coming from the monoid presentation.
2. Stop after some time, even if it has not completed.
3. Compute word differences as above, and approximate FSA to recognise them.
4. Compute a candidate for the word acceptor W.
5. Compute candidates for the multiplier FSA M_x.
6. Carry out correctness tests, terminate if OK, otherwise go back.

Idea of shortlex automatic structure computation

Let $G = \langle X \mid R \rangle$ and set $\hat{X} := X \cup X^{-1}$.

1. Run a shortlex Knuth-Bendix on a RWS coming from the monoid presentation.
2. Stop after some time, even if it has not completed.
3. Compute word differences as above, and approximate FSA to recognise them.
4. Compute a candidate for the word acceptor W.
5. Compute candidates for the multiplier FSA M_x.
6. Carry out correctness tests, terminate if OK, otherwise go back.

`http://tinyurl.com/MNGAPsess/GAP_FP_9.g`

The class of automatic groups is

- closed under taking direct products,

The class of automatic groups is

- closed under taking direct products,
- closed under taking free products with finite amalgamated subgroup,

The class of automatic groups is

- closed under taking direct products,
- closed under taking free products with finite amalgamated subgroup,
- closed under taking HNN-extensions with finite conjugated subgroup.

The class of automatic groups is

- closed under taking direct products,
- closed under taking free products with finite amalgamated subgroup,
- closed under taking HNN-extensions with finite conjugated subgroup.

Furthermore:

- Hyperbolic groups are automatic.

The class of automatic groups is

- closed under taking direct products,
- closed under taking free products with finite amalgamated subgroup,
- closed under taking HNN-extensions with finite conjugated subgroup.

Furthermore:

- Hyperbolic groups are automatic.
- Free factors of automatic groups are automatic.

The class of automatic groups is

- closed under taking direct products,
- closed under taking free products with finite amalgamated subgroup,
- closed under taking HNN-extensions with finite conjugated subgroup.

Furthermore:

- Hyperbolic groups are automatic.
- Free factors of automatic groups are automatic.
- It has not been proved that direct factors of automatic groups are automatic.

The class of automatic groups is

- closed under taking direct products,
- closed under taking free products with finite amalgamated subgroup,
- closed under taking HNN-extensions with finite conjugated subgroup.

Furthermore:

- Hyperbolic groups are automatic.
- Free factors of automatic groups are automatic.
- It has not been proved that direct factors of automatic groups are automatic.
- If $[G : H] < \infty$, then G is automatic iff H is.

The class of automatic groups is

- closed under taking direct products,
- closed under taking free products with finite amalgamated subgroup,
- closed under taking HNN-extensions with finite conjugated subgroup.

Furthermore:

- Hyperbolic groups are automatic.
- Free factors of automatic groups are automatic.
- It has not been proved that direct factors of automatic groups are automatic.
- If $[G : H] < \infty$, then G is automatic iff H is.

Thus:

Automatic groups are a large class of groups with solvable word problem.

In this lecture series we have not mentioned a lot of topics:

In this lecture series we have not mentioned a lot of topics:

- Polycyclic groups (see Bettina's series)

In this lecture series we have not mentioned a lot of topics:

- Polycyclic groups (see Bettina's series)
- Parallelisation of algorithms

In this lecture series we have not mentioned a lot of topics:

- Polycyclic groups (see Bettina's series)
- Parallelisation of algorithms
- Quotient algorithms: Nilpotent, Soluble, p-Quotient

In this lecture series we have not mentioned a lot of topics:

- Polycyclic groups (see Bettina's series)
- Parallelisation of algorithms
- Quotient algorithms: Nilpotent, Soluble, *p*-Quotient
- Finding matrix representations (see W. Plesken et al.)

In this lecture series we have not mentioned a lot of topics:

- Polycyclic groups (see Bettina's series)
- Parallelisation of algorithms
- Quotient algorithms: Nilpotent, Soluble, *p*-Quotient
- Finding matrix representations (see W. Plesken et al.)
- Finding presentations if group is given in another representation

In this lecture series we have not mentioned a lot of topics:

- Polycyclic groups (see Bettina's series)
- Parallelisation of algorithms
- Quotient algorithms: Nilpotent, Soluble, *p*-Quotient
- Finding matrix representations (see W. Plesken et al.)
- Finding presentations if group is given in another representation
- Symmetric presentations (see R. Curtis et al.)

In this lecture series we have not mentioned a lot of topics:

- Polycyclic groups (see Bettina's series)
- Parallelisation of algorithms
- Quotient algorithms: Nilpotent, Soluble, *p*-Quotient
- Finding matrix representations (see W. Plesken et al.)
- Finding presentations if group is given in another representation
- Symmetric presentations (see R. Curtis et al.)
- Infinite presentations

In this lecture series we have not mentioned a lot of topics:

- Polycyclic groups (see Bettina's series)
- Parallelisation of algorithms
- Quotient algorithms: Nilpotent, Soluble, *p*-Quotient
- Finding matrix representations (see W. Plesken et al.)
- Finding presentations if group is given in another representation
- Symmetric presentations (see R. Curtis et al.)
- Infinite presentations
- Laws (e.g. Burnside groups and algorithms for such problems)

In this lecture series we have not mentioned a lot of topics:

- Polycyclic groups (see Bettina's series)
- Parallelisation of algorithms
- Quotient algorithms: Nilpotent, Soluble, p-Quotient
- Finding matrix representations (see W. Plesken et al.)
- Finding presentations if group is given in another representation
- Symmetric presentations (see R. Curtis et al.)
- Infinite presentations
- Laws (e.g. Burnside groups and algorithms for such problems)
- New developments in algorithmic small cancellation theory (see Richard's talk yesterday)

In this lecture series we have not mentioned a lot of topics:

- Polycyclic groups (see Bettina's series)
- Parallelisation of algorithms
- Quotient algorithms: Nilpotent, Soluble, p-Quotient
- Finding matrix representations (see W. Plesken et al.)
- Finding presentations if group is given in another representation
- Symmetric presentations (see R. Curtis et al.)
- Infinite presentations
- Laws (e.g. Burnside groups and algorithms for such problems)
- New developments in algorithmic small cancellation theory (see Richard's talk yesterday)

Derek F. Holt, Bettina Eick, Eamonn A. O'Brien:
"Handbook of Computational Group Theory"