Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Системное программирование

Шамрай Максим Борисович

Реализация алгоритма построения представления группы по машине Тьюринга

Бакалаврская работа

Научный руководитель:
доцент кафедры информатики, к. ф.-м. н., доцент С. В. Григорьев

Рецензент:
ведущий инженер ООО "Ланит-Терком" К. К. Смирнов

Санкт-Петербург
2020

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems

Software Engineering

Maksim Shamrai

Implementation of the algorithm for constructing a finitely presented group by a Turing machine

Bachelor's Thesis

Scientific supervisor:
Assistant Professor Semyon Grigorev

Reviewer:
Senior engineer at Lanit-Tercom LLC Kirill Smirnov

Saint-Petersburg
2020

Оглавление

Введение

Формальные языки являются теоретическим основанием системного программирования, а именно разработки трансляторов и статических анализаторов посредством лексического, синтаксического и семантического анализов. Теория формальных языков начала свое активное развитие в 50-х годах прошлого века [4, 5]. За это время она нашла свое применение не только в областях связанных непосредственно с системным программированием, но, кроме прочего, и в анализе графовых моделей данных [6, 2], которые применяются, например, в биоинформатике и социальных сетях.

Развитие влечет за собой новые теоретические проблемы и неразрешенные задачи. В качестве примера приведем проблему связанную с появлением таких классов языков, как конъюнктивные [10] и булевы [11]. Она касается ограничений соответствующих грамматик. Ограничения выразительной силы обычных контекстно-свободных грамматик известны довольно хорошо. Существуют прямые методы доказательства непредставимости определенных языков, таких как лемма накачки и ее варианты, которые показывают, что некоторые языки не могут быть заданы контекстно-свободной грамматикой. Напротив, нет методов доказывания непредставимости языков булевыми грамматиками, и это главный пробел в знаниях этих грамматик. Точно так же такие методы неизвестны и для конъюнктивных грамматик [12].

В последнее время наблюдается тенденция смещения направления методов изучения формальных языков. Наряду с классическими комбинаторными исследователи начинают все активней применять методы из других разделов математики. Одно из возможных направлений — исследование формальных языков с помощью обращения к теоретико-групповому аппарату.

Стоит отметить, что когда говорят о группах в языковом контексте, часто имеют в виду изоморфное представление группы, определение которого будет представлено далее. По описанию представления группы можно естественным образом задать формальный язык, однако в

обратную сторону — по грамматике построить группу — задача нетривиальная, то есть не существует простого способа построения представления группы по формальному языку. Для этого проводятся сложные математические преобразования распознающей язык машины Тьюринга, и производить их с помощью ручки и бумаги не является целесообразным, а изучать групповые свойства языков выглядит довольно перспективным направлением исследований [15, 13].

Таким образом, актуальной является задача алгоритмизации построения представления группы по машине Тьюринга, так как исследования языков в этом направлении представляют особый интерес. Также в данной работе будут затронуты только контекстно-свободные языки, так как они одни из самых хорошо изученных и востребованных.

Постановка задачи

Целью данной работы является предоставление исследователям возможности ставить вычислительные эксперименты по преобразованию формального языка в соответствующее представление группы.

Достижение поставленной цели обеспечивается решением следующих задач:

1. Реализовать алгоритм преобразования контекстно-свободной грамматики в машину Тьюринга.

2. Разработать алгоритм построения представления группы по машине Тьюринга.

3. Разработать интерпретаторы промежуточных машин для проверки эквивалентности преобразований в контексте сохранения распознаваемого языка.

4. Провести эксперименты.

1. Обзор

В данном разделе приведены определения и теоремы, используемые в данной работе, рассмотрена связь между формальными языками и группами, а также рассмотрен формальный алгоритм построения представления группы по машине Тьюринга, предложенный в статье [15].

1.1. Формальные языки

В этой работе используется стандартные определения теории формальных языков, которые можно увидеть, например, в работах Хомского [4, 5]. Далее будут рассмотрены основные определения, которые используется в данной работе.

Определение 1.1.1. *В классической формализации формальная грамматика G может быть представлена в виде четырехместного кортежа $G = (\Sigma, N, R, S)$, который состоит из следующих компонентов:*

- *Конечное множество Σ терминальных символов.*
- *Конечное множество N нетерминальных символов, которое не пересекается с Σ.*
- *Конечное множество R правил вывода, каждое правило представимо в виде: $(\Sigma \cup N)^* N (\Sigma \cup N)^* \to (\Sigma \cup N)^*$*
- *$S \in N$ — стартовый нетерминал.*

Определение 1.1.2. *Контекстно-свободная грамматика является формальной грамматикой (тип 2 в иерархии Хомского), в которой левые части всех правил вывода являются единичными нетерминалами.*

Примером правила контекстно-свободной грамматики может служить $A \to X_1 \dots X_l$. Оно означает, что каждое слово, представимое в виде конкатенации $X_1 \dots X_l$, обладает свойством A.

Определим автомат с магазинной памятью, который нам пригодится в алгоритме построения машины Тьюринга по контекстно-свободной

грамматике. Также ниже приведена теорема, благодаря которой можно утверждать, что этот алгоритм строит машины Тьюринга, которые распознают в точности контекстно-свободные языки.

Определение 1.1.3. *Автомат с магазинной памятью — это конечный автомат с дополнительным хранилищем стека, переходы которого основаны не только на входе и текущем состоянии, но также и на символах на стеке. Формальное определение выглядит как кортеж из семи состовляющих:* $M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F)$*, где*

- Q *— конечное множество состояний.*
- Σ *— конечный алфавит входных символов.*
- Γ *— конечный алфавит символов на стеке.*
- $\delta \subset (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (Q \times \Gamma^*)$ *— конечное множество переходов.*
- $q_0 \in Q$ *— начальное состояние.*
- $Z \in \Gamma$ *— начальный символ стека.*
- $F \subseteq Q$ *— множество конечных состояний.*

Теорема 1.1.1. *Множество языков, допускаемых автоматами с магазинной памятью, совпадает с множеством контекстно-свободных языков [7].*

1.2. Основные сведения из теории групп

В этом разделе даны определения свободной группе, представлению группы и нескольким базовым определениям и теоремам приводящих к ним. Все описанные в этом разделе определения и теоремы можно найти в книге А. Ю. Ольшанского [16].

Определение 1.2.1. *Множество* G *с заданной на нем операцией* $(\cdot)$ *называется группой, если верно следующее:*

- $\forall a, b, c \in G, (a \cdot b) \cdot c = a \cdot (b \cdot c)$

- $\exists e \in G : \forall a, a \cdot e = e \cdot a = a$
- $\forall a \in G \ \exists a^{-1} \in G : a \cdot a^{-1} = a^{-1} \cdot a = e$

В теории групп смежные классы разделяются в соответствии с положением элемента группы на правый и левый классы. Например, левый смежный класс может быть определен как $aH = \{ah : h \in H\}$, где $a \in G$, H — подгруппа в G. Подгруппа H группы G называется нормальной, если для всех элементов a из G соответствующие левые и правые смежные классы равны, то есть $aH = Ha$. Кроме того, смежные классы H в G образуют группу, которая называется фактор-группой.

Нормальное замыкание S^G подмножества S группы G является наименьшей нормальной подгруппой группы G, которая содержит подмножество S.

$$S^G = \{gsg^{-1} : g \in G,\ s \in S\} \tag{1}$$

Теорема 1.2.1 (Теорема о гомоморфизме)**.** *Следующий результат является одним из основополагающих результатов в теории групп. Если $\phi : G \to H$ является гомоморфизмом, то $Im(\phi) \cong G/Ker(\phi)$.*

Пусть A — произвольное множество символов, тогда $A^{-1} = \{a^{-1} : a \in A\}$ и $A \cup A^{-1}$ — групповой алфавит. ω — слово в алфавите $A \cup A^{-1}$, если $\omega = a_1 \dots a_n$, где $a_i \in A \cup A^{-1}$ $\forall i$ от 1 до n. Слово сократимо, если существует i, для которого буквы x_i и x_{i+1} взаимно обратны.

Определение 1.2.2. *Множество $G(A)$ всех несократимых слов в алфавите $A \cup A^{-1}$ является группой для операции конкатенации с последующим сокращением и называется свободной группой.*

Поскольку для любой группы с произвольным множеством образующих $\{g_i\}_{i \in I}$ существует сюръективный гомоморфизм $\phi : G(A) \to G$ такой, что $\phi(a_i) = g_i$, где $\{a_i\}_{i \in I} = A$ и $i \in I$. Итак, применяя теорему о гомоморфизме (см. теорему 1.2.1), получаем изоморфную фактор группу свободной группы по ядру гомоморфизма:

$$G \cong G(A)/Ker(\phi) \tag{2}$$

Так как по любой нормальной подгруппе возможно определить фактор группу, значит интересна возможность определения каждой группы как фактор группы свободной группы по некоторой нормальной ее подгруппе. В связи с этим интересны способы задания нормальной подгруппы в свободной группе, то есть способы задания ядра гомоморфизма из (2).

Чтобы определить отношение эквивалентности в словах свободной группы $G(A)$, необходимо описать элементарные преобразования слов, которые не меняются смежный класс слова в нормальном замыкании $R^{G(A)}$, где R является подмножеством в $G(A)$, следующим образом:

1. Сокращение.

2. Замена слова $\omega = \omega_1 r^{\pm} \omega_2$, где $r \in R$, словом $\omega_1\omega_2$ и наоборот.

Если возможно привести слова ω_1 и ω_2 друг к другу с помощью этих преобразований, то можно утверждать, что они эквивалентны.

Определение 1.2.3. *Множество отношений $\{r = 1 : r \in R\}$ называется определяющим для группы $G = \langle g_i \rangle$, если любое другое отношение между g_i получается из системы $\{r = 1 : r \in R\}$. Если множество отношений $\{r = 1 : r \in R\}$ является определяющим для группы G, тогда $G \cong G(A)/R^{G(A)}$ и тогда можно сказать, что G имеет групповое представление $\langle A \mid R \rangle$. Для обозначения этого пишут $G = \langle A \mid R \rangle$.*

Таким образом, было дано определение представлению группы, которое является ключевым в данной работе.

1.3. Связь формальных языков с группами

Рассмотрим связь между формальными языками и группами. Пусть Σ — конечный алфавит букв, $(\cdot)$ — конкатенация двух слов, и ε — пустое слово. Тогда

- $(\Sigma^+, \cdot)$ — свободная полугруппа.
- $(\Sigma^*, \cdot, \varepsilon)$ — свободный моноид.

- $((\Sigma \cup \Sigma^{-1})^*, \cdot, \varepsilon)$ — свободная группа.

Действительно, так как для полугруппы достаточно, чтобы групповая операция была ассоциативная

$$\forall a, b, c \in \Sigma^+, a(bc) = (ab)c$$

, моноиду еще необходим нейтральный элемент

$$\forall a \in \Sigma^*, a\varepsilon = \varepsilon a = a$$

, и группе не хватает лишь обратных элементов

$$\forall a \in (\Sigma \cup \Sigma^{-1})^* \, \exists a^{-1} \in (\Sigma \cup \Sigma^{-1})^* : a \cdot a^{-1} = a^{-1} \cdot a = \varepsilon$$

Кроме этой очевидной связи, в теории групп существует понятие проблемы слов группы, которое может быть сформулировано следующим образом.

Определение 1.3.1. *Проблема слов для конечно порожденной группы G — это алгоритмическая задача определения, представляют ли два слова, состоящие из порождающих, один и тот же элемент. Часто ее сводят к равенству слов единице, что эквивалентно.*

Итак, если A — конечное множество образующих для G, а A^{-1} — множество его инверсий, то проблема слов — это проблема принадлежности для множества слов над алфавитом $\Sigma = A \cup A^{-1}$ групповой единице при отображении естественным гомоморфизмом $\phi : \Sigma^* \to G$.

Множество таких слов можно записать следующим образом (см. рис. 1):

$$W(G) = \phi^{-1}(1) = \{\omega \in \Sigma^* : \phi(\omega) = 1\}$$

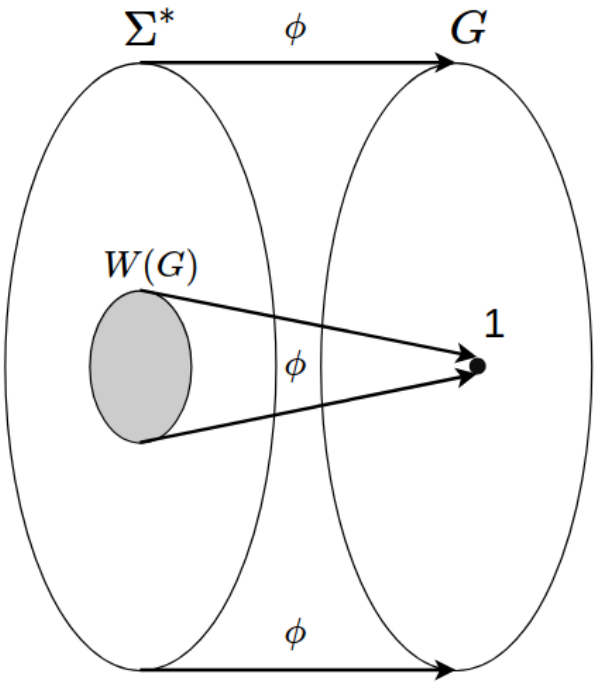

Рис. 1: Проблема слов группы

Для проблемы слов группы уже доказан ряд утверждений, которые показывают, что алгебраические свойства групп соответствуют свойствам формальных языков. Например, для конечно порожденной группы G верно следующее:

- Проблема слов для G разрешима $\iff$ $W(G)$ — рекурсивный язык.
- $W(G)$ – регулярный язык $\iff$ G — конечна [1].
- $W(G)$ – контекстно-свободный язык $\iff$ $\exists H < G$ — свободная подгруппа конечного индекса [9].

1.4. Алгоритм построения представления группы

Описанная в этом разделе статья посвящена исследованию связей между асимптотическими функциями групп и вычислительной сложностью [15]. В частности, авторы показывают, как построить конечно-представленную группу с NP-полной проблемой слова. Для этого они

доказывают несколько теорем, которые и будут представлены в этом разделе.

1.4.1. Машина Тьюринга

Сапир, Биргет и Рипс в своей работе [15] используют следующую нотацию машины Тьюринга.

Определение 1.4.1. *Машина Тьюринга имеет k лент и k голов и может быть описана как шестиместный кортеж:* $M = \langle X, \Gamma, Q, \Theta, \overline{s_1}, \overline{s_0} \rangle$, *где*

- X — *входной алфавит.*
- Γ — *алфавит лент.*
- $Q = \cup_{i=1}^{k} Q_i$ — *множество состояний голов на лентах машины.*
- Θ — *множество команд машины.*
- $\overline{s_1}$ — *k-вектор начальных состояний машины.*
- $\overline{s_0}$ — *k-вектор конечных состояний машины.*

Кроме этого, авторы вводят специальные обозначения α и ω для крайних левых и правых ячеек соответственно и предполагают, что самая левая (самая правая) ячейка на каждой ленте — это всегда α (ω). Тем не менее, когда они говорят о слове, записанном на ленте, они на самом деле не включают эти ячейки в состояние головы, и в каждый момент времени голова наблюдает только две ячейки на каждой ленте.

Конфигурация ленты — это слово $\alpha uqv\omega$, где q — текущее состояние головы на ленте, u (v) — слово слева (справа) от головы. Входная конфигурация — это конфигурация, в которой слово, записанное на первой ленте, из X^+, все остальные ленты пусты, голова видит правый маркер ω, и состояния формируются начальный вектор s_1. Принимающая конфигурация — это любая конфигурация, в которой вектор состояния s_0.

Команда машины Тьюринга определяется состояниями голов и некоторыми из $2k$ букв, наблюдаемых головами. В результате команды возможно заменить какие-то из этих $2k$ букв другими буквами, добавить новые ячейки на ленты, удалить ячейки на лентах и переместить голову на одну ячейку влево (вправо) на лентах. Единственным ограничением является то, что машина может вставлять или удалять ячейки на ленте, но непосредственно перед знаком ω, и ячейка не может быть вставлена слева (справа) от α (ω), и эти крайние маркеры нельзя удалять.

Далее в качестве примера приведены команды одноленточной машины Тьюринга, которые в результате заменяют (3), вставляют (4), перемещают влево (5) и удаляют ячейки (6).

$$uqv \to u'q'v' \tag{3}$$

$$q\omega \to uq'\omega \tag{4}$$

$$uq \to q'u \tag{5}$$

$$uq\omega \to q'\omega \tag{6}$$

где u, v, $u^{'}$, $v^{'}$ — символы алфавита ленты.

1.4.2. Симметричные машины Тьюринга

В дальнейших теоремах будет использоваться класс машин Тьюринга, называемый симметричными машинами Тьюринга. Дадим определения, которые можно увидеть, например, в [3].

Определение 1.4.2. *Машина Тьюринга M симметрична, если для каждой команды в наборе команд существует обратная ей команда. То есть для любой команды $uqv \to u'q'v'$ из множества команд M в нем также есть команда $u'q'v' \to uqv$.*

Определение 1.4.3. *Симметризация машины Тьюринга M — это машина M^{sym}, полученная из M путем добавления обратной команды τ^{-1}*

в множество команд для каждой команды τ из M.

Определение 1.4.4. *Машина Тьюринга M подтвергается симметризации тогда и только тогда, когда M и M^{sym} эквивалентны (т. е. они принимают один и тот же язык).*

Следующая теорема является фундаментальной в изучении проблемы слов и была получена Э. Постом и А. А. Марковым независимо друг от друга в 1947 году.

Теорема 1.4.1. *Каждая детерминированная машина Тьюринга подтвергается симметризации. С другой стороны, не все недетерминированные машины Тьюринга могут быть симметризованы.*

1.4.3. S-машина

Далее перейдем к последнему определению, введенному авторами, системы переписывания под названием S-машина, которая играет ключевую роль в построении представления группы.

Определение 1.4.5. *Пусть n — натуральное число. Аппаратное обеспечение (hardware) S-машины — это пара (Y, Q), где*

- *Y — это n-вектор, где Y_i — алфавит на ленте i.*
- *Q является $(n+1)$-вектором непересекающихся множеств Q_i, элементы которых являются состояниями соответствующей ленты i.*

Множества элементов Q и Y также не пересекаются.

С каждым аппаратным обеспечением $S = (Y, Q)$ можно связать язык допустимых слов $L(S) = Q_1F(Y_1)Q_2 \dots F(Y_n)Q_{n+1}$, где $F(Y_j)$ — язык всех сокращенных групповых слов в алфавите $Y_j \cup Y_j^{-1}$.

Правила перезаписи или S-правила имеют следующую форму: $[U_1 \to V_1, \dots, U_m \to V_m]$ где выполняются следующие условия:

1. Каждый U_i является подсловом допустимого слова, начинающимся с символа Q_l и заканчивающейся буквой Q_r.

2. Если $i < j$, то $r(i) < l(j)$.

3. Каждый V_i также является подсловом допустимого слова, чьи состояния Q принадлежат $Q_{l(i)} \cup \cdots \cup Q_{r(i)}$ и которое содержит по состоянию из Q_l и из Q_r.

4. Буквы ленты не вставляются слева от состояний Q_1 и справа от состояний Q_{n+1}.

Применить S-правило к слову W означает заменить одновременно подслова U_i на подслова V_i, $i = 1, \ldots, m$. После каждого применения правила переписывания, слово автоматически сокращается.

Например, если к слову $q_1aaq_2bq_3$ применить S-правило
$[q_1 \to p_1a^{-1}, q_2bq_3 \to a^{-1}p_2b'q_3]$, где $q_i \in Q_i$, $p_i \in Q_i$, $a \in Y_1$, $b, b' \in Y_2$, то результат применения этого правила — $p_1p_2b'q_3$.

Кроме того, с каждым S-правилом τ нам нужно связать обратное S-правило τ^{-1} следующим образом: если
$\tau = [U_1 \to x_1V_1y_1, \ldots, U_m \to x_mV_my_m]$, тогда
$\tau^{-1} = [V_1 \to x_1^{-1}U_1y_1^{-1}, \ldots, V_m \to x_m^{-1}U_my_m^{-1}]$

Стоит отметить, что на протяжении всей своей работы авторы предполагают, что S-машина симметрична; то есть, если S-машина содержит правило переписывания τ, оно также содержит правило τ^{-1}.

1.4.4. Используемые в построении теоремы

Итак, ранее были даны все необходимые определения, и далее будут представлены теормемы их связывающие, на доказательствах которых и основан наш алгоритм построения представления группы по машине Тьюринга.

Так, например, следующая теорема утверждает, что для любой машины Тьюринга M (в том числе недетерминированной) существует эквивалентная ей симметричная машина Тьюринга, но которая не является ее симметризацией M^{sym}. То есть для построения эквиалентной симметричной машины Тьюринга для любой машины Тьюринга производятся более сложные построения, чем обычная симметризация в

теореме 1.4.1. Действительно, первыми аналогичную теорему доказали Г. Р. Леви и Х. Пападимитриу [8]. Авторы рассматриваемой нами статьи модифицируют построение такой машины из [3].

Теорема 1.4.2. *Для любой машины Тьюринга M, распознающей язык L, существует Машина Тьюринга M' со следующими свойствами:*

1. *Язык, который распознает M' — это L.*
2. *M' симметрична.*
3. *Машина принимает слово только тогда, когда все ленты пусты.*
4. *Любая команда M' или ее обратная имеет одну из следующих форм для некоторого i:*

$$\{q_1\omega \to q_1'\omega, ..., q_{i-1}\omega \to q_{i-1}\omega, aq_i\omega \to q_i\omega, q_{i+1}\omega \to q_{i+1}\omega, ...\} \quad (7)$$

$$\{q_1\omega \to q_1'\omega, ..., q_{i-1}\omega \to q_{i-1}\omega, \alpha q_i\omega \to \alpha q_i\omega, q_{i+1}\omega \to q_{i+1}\omega, ...\} \quad (8)$$

   где a принадлежит ленточному алфавиту ленты i, а q_j, q_j являются состояниями ленты j.

5. *Буквы, используемые на разных лентах, включая состояния, принадлежат не пересекающимся алфавитам.*

Отметим, что симметричные машины Тьюринга более похожи на группы, чем обычные, из-за возможности обратных вычислений, которые возможны в группах. Доказательство этой теоремы позволяет построить симметричную машину Тьюринга, сохраняя распознаваемый язык оригинальной машины.

На следующем этапе авторы предлагают построить S-машину из симметричной Машина Тьюринга для достижения еще большего сходства с группами. Существует «естественный» способ преобразования машины Тьюринга в S-машину. Для этого возьмем машину Тьюринга M, удовлетворяющую всем условиям теоремы 1.4.2, объединим все ленты машины M и заменим каждую команду

$$aq\omega \to q'\omega$$

на

$$q\omega \to a^{-1}\omega$$

, таким образом команды M становятся S-правилами. К сожалению, S-машина, построенная таким образом, не наследует большинство свойств оригинальной машины M. Это происходит потому, что алфавит S-машины содержит символы обратного алфавита. Следовательно, получается, что с одной стороны, S-машины гораздо лучше подходят для имитации своей работы групповыми отношениями, чем обычные машины Тьюринга, так как S-машина может работать с символами обратного алфавита (и, более того, S-машины сами рассматриваются в [14] как HNN-расширения свободной группы с базисом $Y \cup Q$), а с другой стороны, когда симметричная машина Тьюринга начинает работать как S-машина, число принятых слов может бесконтрольно увеличиваться, так как существует возможность распознавать отрицательные слова. Из этого становится очевидным, что для сохранения распознаваемого языка необходимо проводить дополнительные построения.

Прежде чем приступить к построению S-машины, авторы делают следующие переименования. Для каждого $q \in Q$ слово $q\omega$ обозначается через F_q, а левый маркер на ленте $\#i$ заменяют на E_i. После переименования правила симметричной машины Тьюринга или их обратные имеют одну из следующих форм для некоторого i:

$$\{F_{q_1} \to F_{q'_1}, \ldots, aF_{q_i} \to F_{q'_i}, \ldots, F_{q_k} \to F_{q'_k}\} \tag{9}$$

, где $a \in Y$, или

$$\{F_{q_1} \to F_{q'_1}, \ldots, E_iF_{q_i} \to E_iF_{q'_i}, \ldots, F_{q_k} \to F_{q'_k}\} \tag{10}$$

Кроме того, для любой конфигурации $C = (E_1u_1F_{q_1}, ..., E_ku_kF_{q_k})$ машины M, они вводят значение $\sigma(C)$, которое является допустимым словом для $S(M)$:

$E(0)\alpha^n x(0)F(0)$
$E(1)u_1 x(1)F_{q_1}(1)E'(1)p(1)\delta^{||u_1||}q(1)r(1)s(1)t(1)$
$\overline{p}(1)\overline{q}(1)\overline{r}(1)\overline{s}(1)\overline{t}(1)F'_{q_1}(1)$
...
$E(k)u_k x(k)F_{q_k}(k)E'(k)p(k)\delta^{||u_k||}q(k)r(k)s(k)t(k)$
$\overline{p}(k)\overline{q}(k)\overline{r}(k)\overline{s}(k)\overline{t}(k)F'_{q_k}(k)$
$E'(k+1)x(k+1)\omega^n F'(k+1)$
, где $n = |u_1| + ... + |u_k|$.

Для построения S-машины, эквивалентной симметричной машине Тьюринга по распознаваемому языку, авторы дополнительно вводят одинадцать небольших S-машин (отсюда новые симолы на ленте в конфигурации выше), которые выполняют функцию контроля отрицательных символов. После чего для симуляции каждого правила симметричной машины Тьюринга вида (9) запускают эти небольшие S-машины, связывая их между собой отдельными S-правилами. Таким образом, число правил в S-машине значительно больше, чем в эквивалентной ей симметричной машине Тьюринга. Следующая теорема утверждает, что для любой машины Тьюринга, удовлетворяющей теореме 1.4.2, существует S-машина, симулирующая ее.

Теорема 1.4.3. *Пусть $M = \langle X, Y, Q, \Theta, s_1, s_0\rangle$ — машина Тьюринга, которая удовлетворяет условия теоремы 1.4.2. Пусть $W_0 = \sigma(C_0)$, где C_0 — это принимающая конфигурация машины Тьюринга M. Тогда конфигурация C машины M допустима для тогда и только тогда, когда $S(M)$ может правилами переписывания перевести $\sigma(C)$ к W_0.*

На последнем этапе трансформации авторы строят представление группы. Обратите внимание, что они называют одно правило из каждой пары взаимно обратных правил из множества правил Θ положительным, а другое отрицательным. Множество всех положительных правил обозначается как Θ^+, а множество все негативные правила обозначаются через Θ^-.

Теорема 1.4.4. *Пусть $S(M)$ — S-машина, описанная в теореме 1.4.3, тогда для любого натурального числа N, существует представление*

группы $G_N(S)$ с множеством образующих A и множеством отношений $P_N(S)$, где

$$A = \bigcup_{i=1}^{17k+6} Q_i \cup \{\alpha, \omega, \delta\} \cup \bigcup_{i=1}^{k} Y_i \cup \{k_j | j = 1, \ldots, 2N\} \cup \Theta^+$$

и множество отношений состоит из следующих:

1. *Переходные отношения.*

   Эти отношения соответствуют элементам Θ^+.

   Пусть $\tau \in \Theta^+, \tau = [U_1 \to V_1, ..., U_p \to V_p]$.

   Тогда включаем отношения

   $$U_1^\tau = V_1, ..., U_p^\tau = V_p$$

   в $P_N(S)$.

   Если для какого-то j от 1 до $17k + 6$ символы от Q_j не будут появляться ни в одной из U_i, то также включают отношения

   $$q_j^\tau = q_j$$

   для каждого $q_j \in Q_j$.

2. *Вспомогательные отношения.*

   Это всевозможные отношения формы

   $$\tau x = x\tau$$

   , где $x \in \{\alpha, \omega, \delta\} \cup_{i=1}^{k} Y_i, \tau \in \Theta^+$ и все отношения формы

   $$\tau k_i = k_i \tau$$

   , $i = 1, ..., 2N, \tau \in \Theta^+$.

3. *Отношения хаба.*

   Пусть для каждого слова u, $K(u)$ будет обозначать следующее

слово:

$$K(u) = (u^{-1}k_1uk_2u^{-1}k_3uk_4...u^{-1}k_{2N-1}uk_{2N})(k_{2N}u^{-1}k_{2N-1}u...k_2u^{-1}k_1u)^{-1}$$

Тогда отношение хаба будет следующее:

$$K(W_0) = 1$$

Итак, используя эту информацию, следующая теорема является основной в рассматриваемой статье.

Теорема 1.4.5. *Пусть $L \subseteq \Sigma^+$ язык, принимаемый машиной Тьюринга M, тогда существует конечно представленная группа $G(M) = \langle A \mid R \rangle$ и инъективное отображение $K : \Sigma^+ \to (A \cup A^{-1})^+$ такое что:*
$u \in L \iff K(u) \in W(G)$

Авторы описали и доказали все конструкции в приведенных выше теоремах, но никто не автоматизировал эти конструкции на данный момент. В данной же работе предпринята попытка автоматизации этих построений.

2. Описание реализации

В этом разделе приведено краткое описание алгоритма построения представления группы по контекстно-свободной грамматике.

В качестве средства реализации в данной работе используется язык функционального программирования Haskell[1]. Его выбор был обусловлен наличием в нем богатой и удобной системы типов, которая применяется в приложении для представления алгебраических типов, таких как формальная грамматика, машина Тьюринга, S-машина и представление группы. Алгоритм был разделен на модули, каждый из которых содержит функциональность определенного шага преобразования. На рис. 2 представлена архитектура алгоритма, где изображен порядок преобразования контекстно-свободной грамматики в представление группы. Далее будут подробно рассмотрены типы и модули алгоритма.

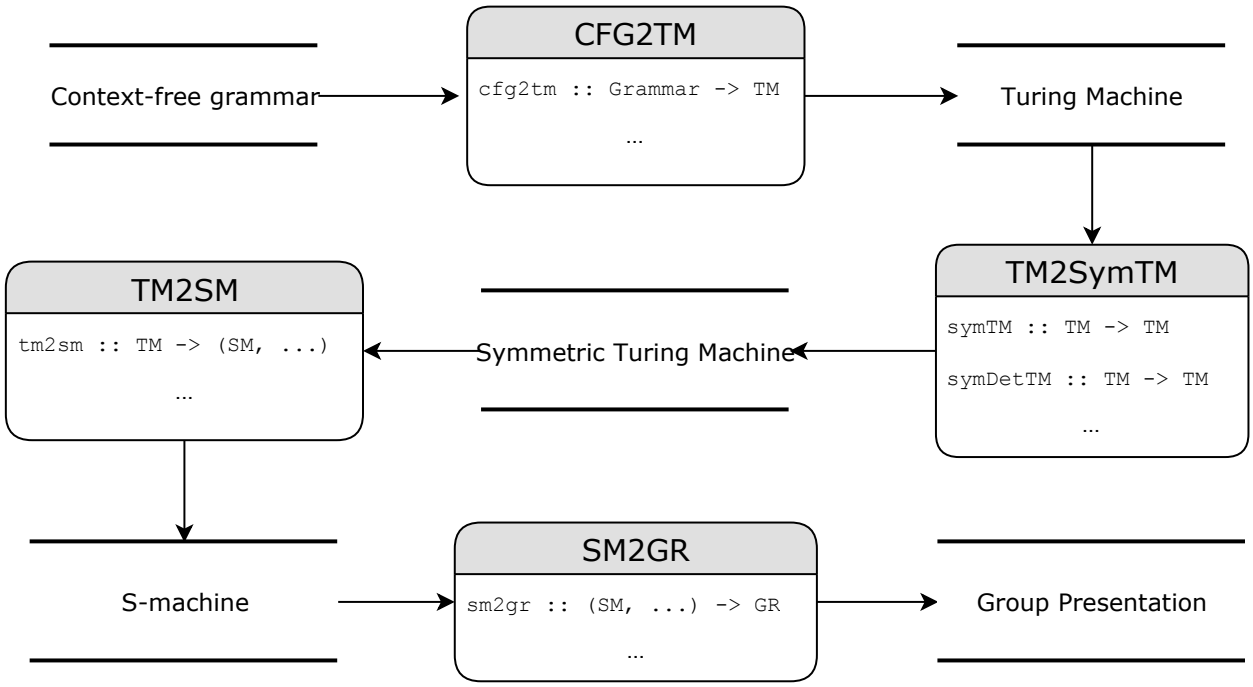

Рис. 2: Архитектура реализации

[1] https://github.com/YaccConstructor/LangToGroup

2.1. Типы данных, реализованные в работе

В данной работе были использованы типы данных языка Haskell. На рис. 3 изображен листинг кода с типами данных, которые представляют грамматики, машины Тьюринга, S-машины и представления групп соответственно. Стоит отметить, что типы данных были намеренно описаны так же, как и в определениях соответствующих понятий. Это было сделано для поддержания максимальной схожести со статьей.

```
- грамматика
newtype Grammar = Grammar
                        (Set Nonterminal, - множество нетерминалов
                        Set Terminal, - множество терминалов
                        Set Relation, - множество правил
                        StartSymbol) - стартовый нетерминал
- машина Тьюринга
newtype TM = TM
                (InputAlphabet, - входной алфавит
                [TapeAlphabet], - алфавиты лент
                MultiTapeStates, - множества состояний лент
                Commands, - множество команд
                StartStates, - вектор стартовых состояний
                AccessStates) - вектор конечных состояний
- S-машина
data SM =  SM
            {yn :: [[Y]], - алфавиты лент
            qn :: [Set State], - множество состояний лент
            srs :: [SRule]} - правила
- представление группы
newtype GR =  GR
                (Set A, - множество порождающих
                Set GrRelation) - множество отношений
```

Рис. 3: Осноные типы данных, используемые в работе

2.2. Построение машины Тьюринга по контекстно-свободной грамматике

Первая часть алгоритма не связана со статьей, рассмотренной в предыдущем разделе, и направлена на то, чтобы построить машину Тьюринга по контекстно-свободной грамматике с эквивалентным распознающимся языком. Чтобы достичь этого, был реализован автомат с магазинной памятью в терминах машины Тьюринга, которая была представлена в определении 1.4.1, с некоторыми расширениями для обеспечения детерминированности машины там, где это возможно.

Алгоритм принимает формальную грамматику в нормальной форме Хомского в качестве входных данных. Это необходимо для облегчения реализации и делается заранее.

Как было упомянуто ранее в теореме 1.1.1, автомат с магазинной памятью позволяет нам распознавать контекстно-свободные языки и, в конце концов, с машиной Тьюринга, распознающей контекстно-свободные языки, возможно построить представление групп для контекстно-свободных языков (см. теорему 1.4.5). Таким образом, в рамках этой работы не разработаны распознаватели для более широких классов языков, которые в общем случае машина Тьюринга может распознавать (например, рекурсивно перечислимые,булевы, конъюнктивные), поэтому для них в настоящее время нельзя построить представление группы. Однако для классов языков, являющихся подклассом контекстно-свободных, это возможно.

Преобразование контекстной-свободной грамматики в машину Тьюринга реализовано в модуле **CFG2TM** алгоритма (см. рис. 4) и может быть запущено с помощью функции *cfg2tm*.

CFG2TM

```
cfg2tm :: Grammar -> TM

genPreviewCommand :: [Relation] -> ... -> (..., [[TapeCommand]], ...)

genRelationCommand :: Relation -> ... -> (..., [[TapeCommand]])

genEraseCommand :: Terminal -> [TapeCommand]
```

Рис. 4: Схема модуля преобразования контекстно-свободной грамматики в машину Тьюринга

Итак, машина Тьюринга, распознающая контекстно-свободные языки, состоит из двух лент: первая лента для входного алфавита, вторая для имитации стека магазинного автомата. $\overline{s_1} = (q_0^1, q_0^2)$, $\overline{s_0} = (q_1^1, q_2^2)$ — вектора начальных и конечных состояний соответственно.

Первая команда машины Тьюринга помещает на вторую ленту стартовый нетерминал:

$$q_0^1\omega \to q_0^1\omega$$
$$q_0^2\omega \to Sq_1^2\omega$$

Также вводится принимающая команда, которая переводит текущий вектор состояний машины Тьюринга в вектор конечных состояний, если на лентах пусто, то есть если слово было успешно принято:

$$\alpha q_0^1\omega \to \alpha q_1^1\omega$$
$$\alpha q_1^2\omega \to \alpha q_2^2\omega$$

Следующие команды зависят от грамматики и разделяются на команды, производящие предпросмотр, команды, сгенерированные из правил грамматики, и команды, полученные из терминалов грамматики. Рассмотрим их по порядку.

Далее будет показано, что нам выгодно распознавать язык при воз-

можности детерминированной машиной Тьюринга. Существуют грамматики, язык которых можно распознать детерминированной машиной Тьюринга, но по которым невозможно однозначно сразу определить какое правило нужно использовать. В качестве примера рассмотрим грамматику

$$S \to aSb$$
$$S \to ab$$

Видя на ленте только литеру a, нельзя одним правилом определить, какое из этих двух правил нужно применить, так как первая литера правой части правил одинакова. Поэтому в функции *genPreviewCommand* генерируются правила, которые выполняют функцию предпросмотра на одну литеру вперед для определения, какое правило нужно применять следующим. Функция *genPreviewCommand* на нашем примере генерирует следующие команды машины Тьюринга:

$aq_0^1 \to q_2^1 a$
$Sq_1^2\omega \to Sq_1^2\omega$ ’

$aq_2^1 a \to aq_5^1 a$ $\quad$ $bq_2^1 a \to bq_3^1 a$
$Sq_1^2\omega \to Sq_1^2\omega$ ’ $\quad$ $Sq_1^2\omega \to Sq_1^2\omega$ ’

$q_5^1 a \to aq_5^1$ $\quad$ $q_3^1 a \to aq_3^1$
$Sq_1^2\omega \to Sq_1^2\omega$ ’ $\quad$ $Sq_1^2\omega \to Sq_1^2\omega$ ’

$aq_5^1\omega \to aq_6^1\omega$ $\quad$ $aq_3^1\omega \to aq_4^1\omega$
$Sq_1^2\omega \to Sq_1^2\omega$ ’ $\quad$ $Sq_1^2\omega \to Sq_1^2\omega$.

Таким образом, если после литеры a стоит снова литера a, то первая лента машины Тьюринга перейдет в состояние q_6^1, а если стоит литера b, то в состояние q_4^1, и так обеспечивается детерминированность машины Тьюринга, кодируя информацию о следующей литере состоянием.

Далее рассмотрим какие команды машины Тьюринга генерирует функция *genRelationCommand*. Пусть в грамматике есть правило $A \to aA'$, тогда, чтобы симулировать это правило, машине Тьюринга необходимо на второй ленте заменить A на символы из правой части правила. Таким образом, из правила $A \to aA'$ получаются следующие команды машины Тьюринга:

$$aq_0^1\omega \to aq_0^1\omega \qquad q_0^1\omega \to q_0^1\omega \qquad q_0^1\omega \to q_0^1\omega$$
$$Aq_1^2\omega \to A'q_3^2\omega \;, \quad q_3^2\omega \to aq_4^2\omega \;, \quad q_4^2\omega \to q_1^2\omega \;.$$

Если же правая сторона правила состоит из одного символа, например, $A \to a$, то нет необходимости в генерации нового состояния для записи на стек, и можно обойтись следующей командой машины Тьюринга:

$$aq_0^1\omega \to aq_0^1\omega$$
$$Aq_1^2\omega \to aq_1^2\omega \;.$$

В случае, когда в грамматике есть правило, в котором правая часть является пустым словом, необходимо иначе генерировать команды машины Тьюринга. Например, рассмотрим грамматику

$$S \to \varepsilon$$
$$S \to aSb$$

, тогда для правила $S \to \varepsilon$ функция *genRelationCommand* сгенерирует следующие команды машины Тьюринга:

$$\alpha q_0^1\omega \to \alpha q_0^1\omega \qquad \alpha q_0^1\omega \to \alpha q_0^1\omega \qquad bq_0^1\omega \to bq_0^1\omega$$
$$q_0^2\omega \to Sq_1^2\omega \;, \quad Sq_1^2\omega \to q_1^2\omega \;, \quad Sq_1^2\omega \to q_1^2\omega \;.$$

Первая из них кладет на стек S при стартовых состояниях и пустой входной ленте, вторая применяет правило, если входная лента пуста, а третья, если следующий символ на ленте b.

Теперь рассмотрим команды машины Тьюринга, которые соответствуют терминалам грамматики и выполняют функцию распознавания входного слова. Чтобы некоторый символ входного алфавита был распознан машиной Тьюринга, необходимо, чтобы он был наблюдаем на входной и стековой ленте. Считается, что машина Тьюринга принимает некоторое слово, если после работы машины ее ленты пусты и она находится в конечном состоянии. Таким образом, процесс распознавания символа заключается в стирании этого символа с двух лент одновременно. Итак, любому терминалу a соответствует следующая команда:

$$aq_0^1\omega \to q_0^1\omega$$
$$aq_1^2\omega \to q_1^2\omega \;.$$

Функция сопоставляющая каждому терминалу правило такого вида называется *genEraseCommand*.

Таким образом, с помощью рассмотренных здесь функций производится преобразование контекстно-свободной грамматики в машину Тьюринга, распознающую язык этой грамматики. Кроме этого, в приложении А приведен пример построения машины Тьюринга, используя данный алгоритм.

2.3. Симметризация машины Тьюринга

Следующий шаг после построения машины Тьюринга — это ее симметризация. Построение эквивалентной симметричной машины Тьюринга производится в модуле **TM2SymTM** (см. рис. 5).

TM2SymTM

```
symTM :: TM -> TM

symDetTM :: TM -> TM

threePhaseProcessing :: TM -> TM

firstPhase :: [State] -> ... -> [[TapeCommand]]

secondPhase :: State -> ... -> [[TapeCommand]]

thirdPhase :: State -> ... -> [[TapeCommand]]

symCommands :: [[TapeCommand]] -> [[TapeCommand]]

doubleCommands :: ... -> [[TapeCommand]] -> (..., [[TapeCommand]])

one2KCmds :: (..., [[TapeCommand]]) -> (..., [[TapeCommand]])

cmd2SIDCmd :: (..., [[TapeCommand]]) -> (..., [[TapeCommand]])
```

Рис. 5: Схема модуля симметризации машины Тьюринга

В ходе данной работы было реализовано два алгоритма симметризации машины Тьюринга. Один из них основан на теореме 1.4.2, которая доказана в статье [15], а другой на теореме 1.4.1. Первый алгоритм может стоить эквивалентную симметричную машину Тьюринга как для

детерминированной, так и для недетерминированной машины Тьюринга. Тут стоит отметить, что этот алгоритм в данной работе называется алгоритмом симметризации недетерминированной машины Тьюринга, так как детерминированные машины Тьюринга являются подклассом недетерминированных. Второй же алгоритм может давать на выходе эквивалентную симметричную машину Тьюринга, только если ему на вход была подана детерминированная машина Тьюринга, но при этом размер и сложность получившейся машины получается гораздо меньше, что будет показано далее.

Перейдем к рассмотрению этих двух алгоритмов.

2.3.1. Алгоритм симметризации недетерминированной машины Тьюринга

Весь алгоритм может быть запущен вместе с функцией $symTM$, которая интегрирует функции, которые будут рассмотрены далее.

Итак, первая фаза алгоритма, как и в статье [15], подразумевает построение новой машины Тьюринга M', которая равна исходной за исключением добавленной $k+1$ ленты, где k — число лент у входной машины, такой, что ее алфавит состоит из команд входной машины Тьюринга. После чего, генерируются команды машины Тьюринга, которые недетерминированно заполняют $k+1$ ленту, единственное тут ограничение — это то, что первая добавленная команда должна завершать вычисление исходной машины. Таким образом, после первой фазы на $k+1$ ленте машины Тьюринга M' должна лежать последовательность команд исходной машины Тьюринга, где первая команда завершает ее вычисление (переводит вектор состояний в вектор конечных состояний). После этого происходит переход ко второй фазе.

Вторая фаза заключается в том, что M' пытается исполнить команды в том порядке, в котором они находятся на $k+1$ ленте. Для этого из команды исходной машины Тьюринга:

$$\tau = \{U_1 \to V_1, ..., U_k \to V_k\}$$

строится команда M':

$$\tau' = \{U_1 \to V_1, ..., U_k \to V_k, \tau q \to q\tau\}$$

где q — состояние M' на ленте $k+1$.

Авторы утверждают, что после таких вычислений первая лента M' опустеет, а $k{+}1$ справа будет наблюдать α только, если в течение первой фазы на $k{+}1$ ленту была записана последовательность команд исходной машины Тьюринга, которая приводит ее к конечным состояниям.

После второй фазы M' возвращает голову к ω на $k+1$ ленте и начинается третья фаза, которая заключается в очищении всех лент и переходе в вектор конечных состояний.

Эти три фазы реализованы в функциях с соответствующими названиями *firstPhase*, *secondPhase*, *thirdPhase*, а функция *threePhaseProcessing* интегрирует их и строит M' после этих трех фаз, в конце данной работы в приложении В можно увидеть результат работы этой функции на машине Тьюринга из одного правила.

Как утверждается в [3], теперь можно из M' построить эквивалентную симметричную машину Тьюринга добавлением в множество команд каждой команде ее обратную. В алгоритме это делает функция *symCommands*.

Таким образом, после этих шагов M' удовлетворяет первым трем свойствам из теоремы 1.4.2, и остается преобразовать ее так, чтобы она удовлетворяла последним двум свойствам.

Для достижения четвертого свойства происходит удваивание лент. Так для любой ленты i появляется лента $i + 1/2$, такая что если изначально конфигурация ленты i: $\alpha uqv\omega$, то после удвоения конфигурации лент i и $i{+}1/2$: $\alpha uq\omega$ и $\alpha\bar{v}q\omega$ соответственно, где u, v — слова, $\bar{v}$ — слово v, записанное с права налево, q состояние, которое далее переименовывается с целью достижения 5 свойства. В связи с этим необходимо заменить команды:

$$\{u_1q_1v_1 \to u'_1q'_1v'_1, ..., u_kq_kv_k \to u'_kq'_kv'_k\}$$

на команды:

$$\{u_1q_1\omega \to u'_1q'_1\omega, \bar{v_1}q_1\omega \to \bar{v'_1}q'_1\omega, ..., u_kq_k\omega \to u'_kq'_k\omega, \bar{v_k}q_k\omega \to \bar{v'_k}q'_k\omega\}$$

Если v в какой-то из команд равно ω, то $\bar{v} = \alpha$. Очевидно, что машина Тьюринга, которую преобразовали таким образом, будет распознавать такой же язык как и исходная машина Тьюринга. Эти преобразования реализованы в функции *doubleCommands*.

Следующее действие — это разделение каждой команды на $2k$ команд, чтобы каждая новая команда затрагивала только одну ленту. Данная операция производится в функции *one2KCmds*. Для этого генерируется для каждой команды множество уникальных состояний, чтобы поддерживать порядок вычисления. Таким образом получаются команды следующего вида:

$$\{q_1\omega \to q_1'\omega, ..., u_iq_i\omega \to u_i'q_i'\omega, ...\}$$

Заметим, что если $u_i = \alpha$, то команда уже имеет вид (8) так как u_i' тоже равно α, иначе требуется еще раз удвоить команду, чтобы они имели вид (7) и это делает функция *cmd2SIDCmd*:

$$\{q_1\omega \to q_1''\omega, ..., u_iq_i\omega \to q_i''\omega, ...\}$$

$$\{q_1''\omega \to q_1'\omega, ..., q_i''\omega \to u_i'q_i'\omega, ...\}$$

Таким образом, производится построение машины Тьюринга, которая удовлетворяет 4 свойству теоремы 1.4.2. Чтобы удовлетворить 5 свойству, достаточно сгенерировать уникальные имена для состояний и литер, что в алгоритме происходит в процессе построения.

2.3.2. Алгоритм симметризации детерминированной машины Тьюринга

Заметим, что первые три фазы алгоритма симметризации недетерминированной машины Тьюринга направленны на построение эквивалентной симметричной машины Тьюринга и, если у нас есть альтернативный вариант ее построения, эти действия можно не производить.

Как было написано выше, согласно теореме 1.4.1 можно симметризовать детерминированную машину Тьюринга, просто добавив в множество команд каждой команде ее обратную, и этого достаточно для

построения эквивалентной симметричной машины Тьюринга. Но свойства 4 и 5 теоремы 1.4.2 все равно нужно обеспечить в симметричной машине, чтобы перейти на следующий шаг построения представления группы. Поэтому для успешного построения эквивалентной симметричной машины Тьюринга по детерминированной машине Тьюринга достаточно выполнить функцию *symDetTM*, которая интегрирует в себе функции *doubleCommands*, *one2KCmds*, *cmd2SIDCmd* и *symCommands*, рассмотренные выше и гарантирующие выполнение свойств 4 и 5. Кроме того, в приложении В можно увидеть пример работы алгоритма.

2.4. Построение S-машины по симметричной машине Тьюринга

В этом разделе будет рассмотрен алгоритм построения S-машины, симулирующей симметричную машину Тьюринга и в некотором смысле сохраняющей язык, распознаваемый этой машиной Тьюринга. Модуль, в котором реализованы функции, необходимые для построения, называется **TM2SM**, и его схема представлена на рис. 6. Функция, производящая построение S-машины, называется *tm2sm*, далее рассмотрим функции, суперпозицией которых она является.

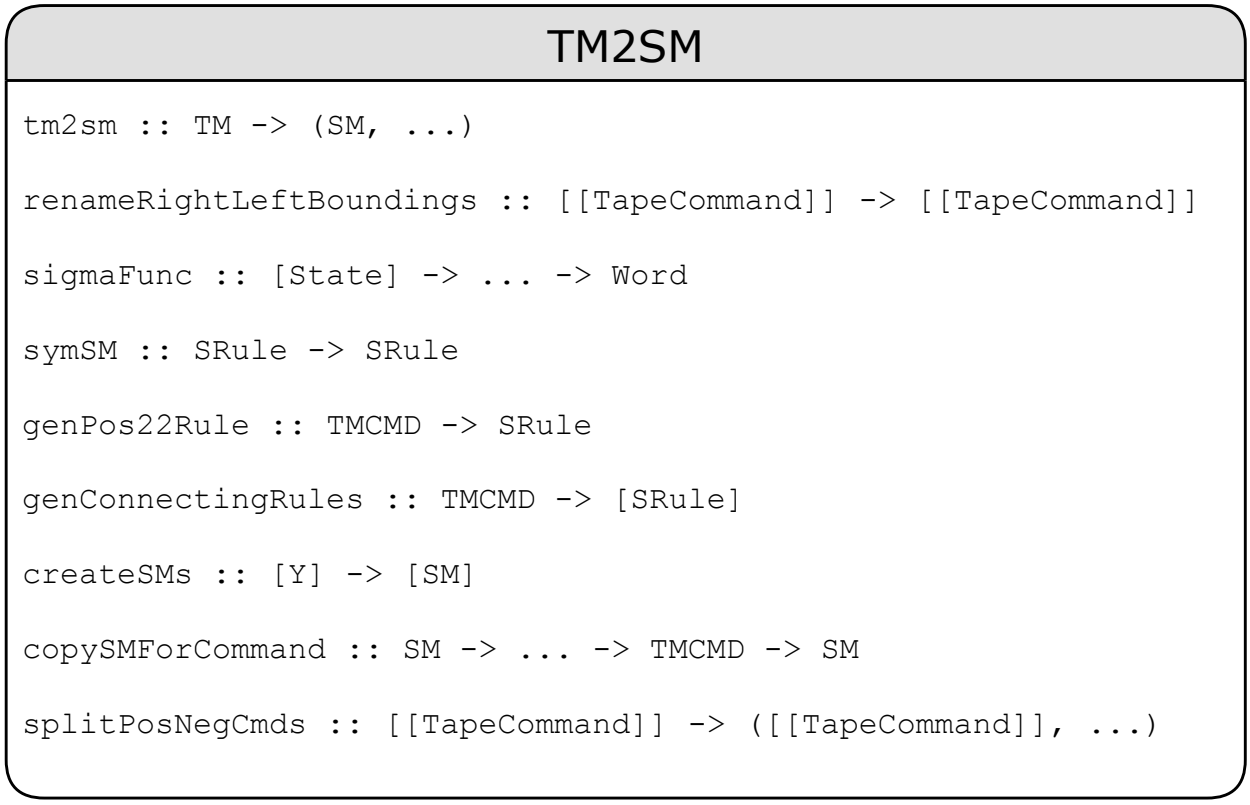

Рис. 6: Схема модуля построения S-машины, симулирующей машину Тьюринга

Как уже было написано в обзоре, перед непосредственно построением S-машины происходит переименования команд машины Тьюринга так, чтобы они имели одну из форм (9, 10). Это действие в данной работе производит функция *renameRightLeftBoundings*. Также в функции *createSMs* строятся 11 вспомогательных S-машин: S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_α, S_ω, и в функции *genConnectingRules* генерируются S-правила, которые являются переходными между вспомогательными машинами.

После этого каждой команде машины Тьюринга вида (9) сопоставляется множество S-правил вспомогательных S-машин и переходных S-правил, а всем командам вида (10) сопоставляется одно S-правило, которое не производит вычислений, но меняет состояние. Сопоставление происходит путем добавления к S-правилам метки принадлежности к определенной команде машины Тьюринга.

Поиск команд вида (9, 10) осуществляет функция *splitPosNegCmds*, сопоставление множеств S-правил команде вида (9) производит функция *copySMForCommand*, а генерацию S-правила для команды вида (10) функция *genPos22Rule*. Кроме этого реализована функция симметризации S-машины *symSM* и функция $\sigma(C)$ (*sigmaFunc*), значение которой было описано в обзоре.

Состояния получившейся S-машины — это все состояния промаркированных вспомогательных S-машин, объединенные со всеми также промаркированными E_i и F_q, которые были получены после переименования машин Тьюринга, где $i = 1..k$ — номер ленты, $q \in Q$ — состояние машины Тьюринга. Алфавит S-машины — объединенные алфавиты вспомогательных S-машин с алфавитом машины Тьюринга.

2.5. Построение представления группы по S-машине

Процесс построения представления группы по S-машине был подробно описан в теореме 1.4.4. В этом разделе покажем, какие функции производят это построение. **SM2GR** — модуль, в котором содержаться эти функции (см. рис. 7).

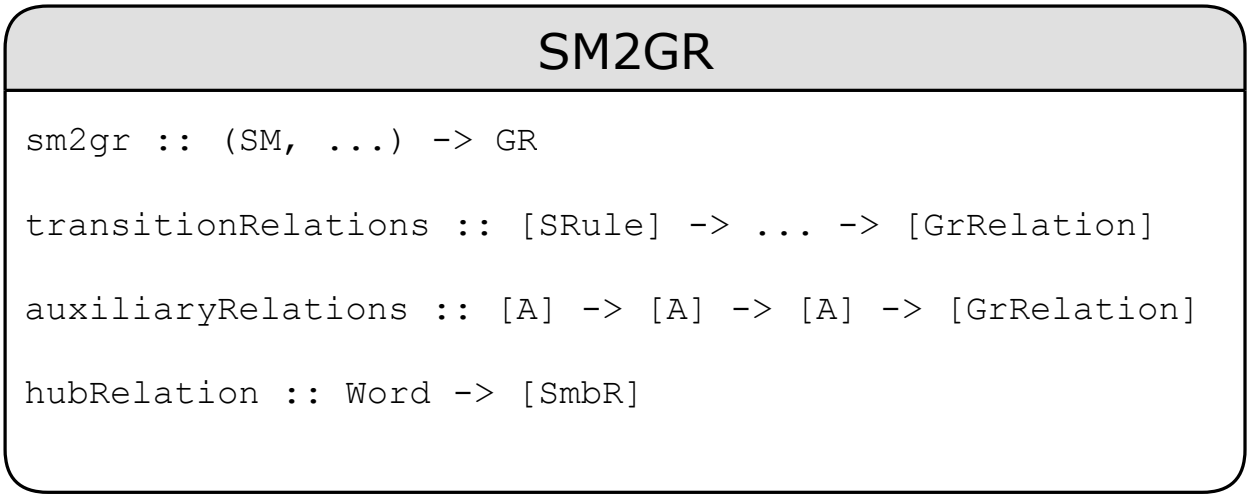

Рис. 7: Схема модуля построения представления группы по S-машине

Переходные вспомогательные отношения и отношение хаба генерируются в функциях *transitionRelations*, *auxiliaryRelations* и *hubRelation*

соответственно. Образующие представления группы генерируются в функции *sm2gr*, которая, кроме этого, объединяет все отношения.

2.6. Вспомогательные инструменты

В этом разделе будут описаны дополнительные модули приложения, разрабатываемого в рамках данной работы, которые были написаны, во-первых, с целью облегчить восприятие и, во-вторых, для проверки эквивалентности преобразований в контексте сохранения распознаваемого языка при переходах от машины к машине.

Для первого были разработаны модули, которые генерируют код LaTeX, с возможностью его генерации для отображения грамматик, машин Тьюринга, конфигураций машин Тьюринга, S-машин. Для такой генерации использовалась библиотека HaTeX[2]. Был создан класс ShowLaTeX и реализован для всех типов, которые необходимо было печатать в LaTeX. Таким образом, были реализованы модули на языке Haskell **GrammarPrinter**, **TMPrinter**, **SMPrinter** и **ConfigPrinter**, в которых реализован класс ShowLaTeX для типов *Grammar*, *TM*, *SM* и *Configs* соответственно. Примеры такой генерации можно наблюдать в приложениях A и B.

Для второго были написаны модули, которые интерпретируют отдельно машины Тьюринга и S-машины на заданных входных данных, общая концепция которых заключается в обходе дерева конфигураций в ширину с остановкой алгоритма в случае нахождения в фронте конечной конфигурации. Так, при обходе в ширину к каждой конфигурации из фронта необходимо применить команды машины Тьюринга или S-правила, которые возможно применить к данной конфигурации, а получившиеся конфигурации после применения добавить в множество фронта.

Такой обход в ширину хорошо работает в случае с машиной Тьюринга, а в интерпретаторе S-машины из-за ее сильной недетермини-

[2]HaTeX — библиотека, реализующая синтаксис LaTeX и несколько полезных абстракций.
Описание библиотеки: `https://hackage.haskell.org/package/HaTeX-3.22.2.0`
Дата последнего посещения: 14.05.20

рованности было также использовано множество, в котором хранятся пройденные конфигурации. Если после применения правила получившаяся конфигурация уже есть в этом множестве, то такая конфигурация не добавляется в фронт. Таким образом, исключается возможность возвращения к уже пройденным конфигурациям, что облегчает поиск конечной конфигурации.

Результатом работы интерпретаторов является история пройденных конфигураций, которые привели к конечной и из которых при необходимости можно сгенерировать LaTeX.

Функции для интерпретации машины Тьюринга содержатся в модуле **TMInterpreter**, где функция *interpretTM* производит интерпретация. Аналогично модуль интерпретатора S-машины **SMInterpreter**, где *interpretSM* производит интерпретацию. Кроме этого, в этом модуле присутствует функция *getRestrictedGraph*, которая строит граф заданной высоты. После этого этот граф можно напечатать в dot-файл с помощью функции *writeGraph* модуля **DotGraphWriter**, печатающей его с помощью библиотеки graphviz[3].

Кроме этого, были реализованы модули, которые печатают представление группы в исходный файл математических пакетов GAP[4] и Maple[5], для последующего анализа представления группы ими.

Таким образом, у нас есть возможность наглядно продемонстрировать результат работы алгоритма и проверить совпадают ли распознаваемые языки после переходов.

---

[3]graphviz — пакет инструментов с открытым исходным кодом для визуализации графов.
Описание библиотеки: `https://hackage.haskell.org/package/graphviz`
Дата последнего посещения: 14.05.20

[4]GAP (Groups, Algorithms, Programming) — программный пакет, система компьютерной алгебры с открытым исходным кодом и особым акцентом на вычислительную теорию групп.
Сайт пакета: `https://www.gap-system.org/`
Дата последнего посещения: 14.05.20

[5]Maple — программный пакет, система компьютерной математики ориентированная на сложные математические вычисления, визуализацию данных и моделирование.
Сайт пакета: `https://www.maplesoft.com/products/Maple/`
Дата последнего посещения: 14.05.20

3. Эксперименты

В данном разделе приведены результаты работы цепочек алгоритмов построения представления группы из контекстно-свободных грамматик в нормальной форме Хомского с выводом численных размеров машин на всех шагах преобразования.

Для оценки размера получившегося представления группы, запустили алгоритмы с недетерминированной и детерминированной симметризацией на трех грамматиках: грамматика из одного правила (11), грамматика языка "a*" (12) и однозначная грамматика языка Дика на одном типе скобок (13).

$$S \to a \tag{11}$$

$$\begin{aligned} S &\to AS \\ S &\to \epsilon \\ A &\to a \end{aligned} \tag{12}$$

$$\begin{aligned} S &\to AC \\ S &\to \epsilon \\ C &\to SD \\ D &\to BS \\ A &\to a \\ B &\to b \end{aligned} \tag{13}$$

Таблица 1 демонстрирует размеры каждого из множеств наших алгебраических типов, построенных в процессе работы алгоритма с недетерминированной симметризацией машины Тьюринга. По ней очевидно, что был получен алгоритм, который даже на простых исходных данных строит довольно большое представление группы. Итак, грамматика только из одного правила превращается в около 90 тысяч групповых отношений и 56 тысяч образующих.

	Grammar			**TM**			
	Σ	N	R	X	Γ	Q	Θ
1 rule	1	1	1	1	3	5	4
a^*	1	2	3	1	4	7	9
Dyck	2	4	6	2	8	11	19

SymTM				**SM**			**G**	
X	Γ	Q	Θ	Y	Q	Θ	A	R
1	14	270	206	14	88246	2363	89508	56187
1	26	547	434	26	344118	5741	347370	204903
2	54	1131	900	54	1469136	15064	1478859	957619

Таблица 1: Мощности множеств машин при использовании алгоритма симметризации недетерминированных машин Тьюринга

Но анализируя эту таблицу, можно заметить, что основной рост размеров происходит на этапах симметризации машины Тьюринга и преобразования симметричной машины Тьюринга в S-машину. Поэтому, чтобы уменьшить размер симметричной машины Тьюринга, был реализован алгоритм симметризации детерминированной машины Тьюринга, который также рассмотрен выше. Таблица 2 демонстрирует результаты его работы.

	Grammar			**TM**			
	Σ	N	R	X	Γ	Q	Θ
1 rule	1	1	1	1	3	5	4
a^*	1	2	3	1	4	7	9
Dyck	2	4	6	2	8	11	19

SymTM				**SM**			**G**	
X	Γ	Q	Θ	Y	Q	Θ	A	R
1	6	39	34	6	6058	501	6410	7637
1	8	73	72	8	15888	1024	16565	17657
2	16	161	158	16	67754	2837	69772	71533

Таблица 2: Мощности множеств машин при использовании алгоритма симметризации детерминированных машин Тьюринга

Заметим, что действительно использование этого алгоритма уменьшило в несколько раз размер симметричной машины Тюринга, S-машины и представления группы по сравнению с использованием алгоритма симметризации недетерминированных машин Тьюринга. Понятно, что чем меньше представление группы, тем проще найти слова, которые равны групповой единице. Из этого можно сделать вывод, что в случае детерминированных грамматик, нужно использовать симметризацию для детерминированных машин.

Также были предприняты попытки проверить на равенство единице некоторые слова из полученных представлений групп с помощью математических пакетов GAP и Maple. К сожалению, из-за сложности вычислений математические пакеты, с которыми проводились эксперименты, не завершали свою работу в приемлемое время.

Заключение

В ходе выполнения данной работы были достигнуты следующие результаты:

1. Реализован алгоритм преобразования контекстно-свободной грамматики в машину Тьюринга.

2. Разработан алгоритм построения представления группы по машине Тьюринга.

3. Разработаны интерпретаторы машины Тьюринга и S-машины.

4. Проведен ряд экспериментов.

Так как описанное решение на данный момент не имеет своего применения, существует необходимость в его улучшении, а именно, если хочется получить представление группы удовлетворительных размеров, то в дальнейшей работе необходимо узнать, как более оптимальным образом преобразовать симметричные машины Тьюринга в S-машины.

Мы можем утверждать, что возможно найти более оптимальный алгоритм построения S-машины по симметричной машине Тьюринга по ряду причин. Во-первых, потому что существуют альтернативные способы построения S-машины по симметричной машине Тьюринга [13]. Во-вторых, так как по некоторым языкам можно построить представление группы, например, для языка $L = \{a\}$ можно построить представление группы и отношение из теоремы 1.4.5, такое что

$$G = \langle k_1, k_2, a \mid k_1 a k_2 = 1 \rangle$$

и $K(a) = k_1 a k_2$. Понятно, что такое построение работает только для конечных языков, но это наталкивает на мысль, что, возможно, существует более простое построение и в общем случае.

Кроме этого, актуальным остается вопрос интерпретации представления группы, так как существующие математические пакеты не смогли справится с этой задачей.

A. Преобразование грамматики

В этом разделе приведен пример работы вышеизложенного алгоритма преобразования элементарной грамматики в машину Тьюринга с последующей ее интерпретацией.

Из следующей грамматики, состоящей из одного правила:

Nonterminals

S

Terminals

a

Rules

$S \to a$

Была получена следующая машина Тьюринга с помощью алгоритма реализованного в рамках данной работы:

Input alphabet

a

Tape alphabets

1. a
2. S, a'

States

1. q_0^1, q_1^1
2. q_0^2, q_1^2, q_2^2

Start states

$q_0^1,\ q_0^2$

Access states

$q_1^1,\ q_2^2$

Commands

$$\begin{bmatrix} aq_0^1\omega & \to & aq_0^1\omega \\ Sq_1^2\omega & \to & a'q_1^2\omega \end{bmatrix} \begin{bmatrix} aq_0^1\omega & \to & q_0^1\omega \\ a'q_1^2\omega & \to & q_1^2\omega \end{bmatrix} \begin{bmatrix} \alpha q_0^1\omega & \to & \alpha q_1^1\omega \\ \alpha q_1^2\omega & \to & \alpha q_2^2\omega \end{bmatrix} \begin{bmatrix} q_0^1\omega & \to & q_0^1\omega \\ q_0^2\omega & \to & Sq_1^2\omega \end{bmatrix}$$

После чего проинтерпретировали получишуюся машину Тьюринга на входной строке a и получили историю изменения конфигураций:

Configurations

№	Tape 1			Tape 2		
1	αa	q_0^1	ω	α	q_0^2	ω
2	αa	q_0^1	ω	αS	q_1^2	ω
3	αa	q_0^1	ω	$\alpha a'$	q_1^2	ω
4	α	q_0^1	ω	α	q_1^2	ω
5	α	q_1^1	ω	α	q_2^2	ω

B. Преобразование машины Тьюринга

В данном разделе рассмотрен пример работы трех фаз алгоритма симметризации недетерминированных машин Тьюринга, а именно функции *threePhaseProcessing*, и также алгоритма симметризации детерминированной машины Тьюринга.

Из машины Тьюринга, содержащей одно правило и принимающей язык из одного слова a:

Input alphabet

a

Tape alphabets

1. a

States

1. q_0^1, q_1^1

Start states

q_0^1

Access states

q_1^1

Commands

$$\left[aq_0^1\omega \;\rightarrow\; q_1^1\omega\right]$$

После трех фаз симметризации алгоритмом для недетерминированных машин получаем симметричную машину Тьюринга, которая не удовлетворяет 4 свойству теоремы 1.4.2:

Input alphabet

a

Tape alphabets

1. a
2. $\left[aq_0^1\omega \ \rightarrow \ q_1^1\omega\right]$

States

1. q_0^1, q_1^1, q_2^1
2. q_0^2, q_1^2, q_2^2, q_3^2

Start states

q_2^1, q_0^2

Access states

q_1^1, q_3^2

Commands

$$\left[\begin{array}{rcl} aq_0^1\omega & \rightarrow & q_1^1\omega \\ \left[aq_0^1\omega \ \rightarrow \ q_1^1\omega\right] q_2^2 & \rightarrow & q_2^2\left[aq_0^1\omega \ \rightarrow \ q_1^1\omega\right] \end{array}\right] \left[\begin{array}{lcl} \alpha q_1^1\omega & \rightarrow & \alpha q_1^1\omega \\ \alpha q_2^2\omega & \rightarrow & \alpha q_3^2\omega \end{array}\right]$$

$$\left[\begin{array}{rcl} \alpha q_1^1\omega & \rightarrow & \alpha q_1^1\omega \\ q_2^2\left[aq_0^1\omega \ \rightarrow \ q_1^1\omega\right] & \rightarrow & \left[aq_0^1\omega \ \rightarrow \ q_1^1\omega\right] q_2^2 \end{array}\right]$$

$$\left[\begin{array}{rcl} \alpha q_1^1\omega & \rightarrow & \alpha q_1^1\omega \\ \left[aq_0^1\omega \ \rightarrow \ q_1^1\omega\right] q_2^2\omega & \rightarrow & q_2^2\omega \end{array}\right]$$

$$\left[\begin{array}{lcl} q_2^1\omega & \rightarrow & q_0^1\omega \\ q_1^2\omega & \rightarrow & q_2^2\omega \end{array}\right] \left[\begin{array}{lcl} q_2^1\omega & \rightarrow & q_2^1\omega \\ q_0^2\omega & \rightarrow & \left[aq_0^1\omega \ \rightarrow \ q_1^1\omega\right] q_1^2\omega \end{array}\right]$$

А применяя алгоритм симметризации детерминированной машины Тьюринга к той же элементарной машине, получаем симметричную машину Тьюринга:

Input alphabet

a

Tape alphabets

1. a
2. a''

States

1. q_0^1, q_1^1
2. $q_0^{1.5}$, $q_1^{1.5}$, $q_2^{1.5}$

Start states

q_0^1, $q_0^{1.5}$

Access states

q_1^1, $q_1^{1.5}$

Commands

$$\begin{bmatrix} aq_0^1\omega & \to & q_1^1\omega \\ q_0^{1.5}\omega & \to & q_2^{1.5}\omega \end{bmatrix} \begin{bmatrix} q_1^1\omega & \to & aq_0^1\omega \\ q_2^{1.5}\omega & \to & q_0^{1.5}\omega \end{bmatrix} \begin{bmatrix} q_1^1\omega & \to & q_1^1\omega \\ \alpha q_1^{1.5}\omega & \to & \alpha q_2^{1.5}\omega \end{bmatrix}$$
$$\begin{bmatrix} q_1^1\omega & \to & q_1^1\omega \\ \alpha q_2^{1.5}\omega & \to & \alpha q_1^{1.5}\omega \end{bmatrix}$$

Список литературы

[1] Anisimov A. V. Languages over free groups // Mathematical Foundations of Computer Science 1975 4th Symposium, Mariánské Lázně, September 1–5, 1975 / Ed. by Jíří Bečvář. — Berlin, Heidelberg : Springer Berlin Heidelberg, 1975. — P. 167–171.

[2] Azimov Rustam, Grigorev Semyon. Context-free path querying by matrix multiplication // GRADES-NDA '18. — 2017.

[3] Birget Jean-Camille. Time-Complexity of the Word Problem for Semigroups and the Higman Embedding Theorem // International Journal of Algebra and Computation. — 1998. — Vol. 08, no. 02. — P. 235–294. — https://doi.org/10.1142/S0218196798000132.

[4] Chomsky N. Some methodological remarks on generative grammar. — Aspects of the Theory of Syntax, 1961.

[5] Chomsky N. Syntactic structures. — Walter de Gruyter, 2002.

[6] Hellings Jelle. Querying for Paths in Graphs using Context-Free Path Queries. — 2015.

[7] Hopcroft John E, Motwani Rajeev, Ullman Jeffrey D. Introduction to Automata Theory, Languages, and Computation: Pearson New International Edition. — Pearson Higher Ed, 2013.

[8] Lewis Harry R., Papadimitriou Christos H. Symmetric space-bounded computation // Theoretical Computer Science. — 1982. — Vol. 19, no. 2. — P. 161 – 187. — Access mode: `http://www.sciencedirect.com/science/article/pii/0304397582900585`.

[9] Muller David, Schupp Paul. Context-free languages, groups, the theory of ends, second-order logic, tiling problems, cellular automata, and vector addition systems // Bulletin of the American Mathematical Society. — 1981. — 07. — Vol. 4.

[10] Okhotin Alexander. Conjunctive Grammars // J. Autom. Lang. Comb. — 2001. — Apr. — Vol. 6, no. 4. — P. 519–535.

[11] Okhotin Alexander. Boolean grammars // Information and Computation. — 2004. — Vol. 194, no. 1. — P. 19 – 48. — Access mode: http://www.sciencedirect.com/science/article/pii/S0890540104001075.

[12] Okhotin Alexander. Conjunctive and Boolean grammars: The true general case of the context-free grammars // Computer Science Review. — 2013. — Vol. 9. — P. 27 – 59. — Access mode: http://www.sciencedirect.com/science/article/pii/S157401371300018X.

[13] Olshanskii Alexander. Space functions of groups // Transactions of the American Mathematical Society. — 2010. — 10. — Vol. 364.

[14] Olshanskii Alexander, Sapir Mark. Groups with small Dehn functions and bipartite chord diagrams. — 2004. — math/0411174.

[15] Sapir Mark V., Birget Jean-Camille, Rips Eliyahu. Isoperimetric and Isodiametric Functions of Groups // Annals of Mathematics. — 2002. — Vol. 156, no. 2. — P. 345–466. — Access mode: http://www.jstor.org/stable/3597195.

[16] Ольшанский А. Ю. Геометрия определяющих соотношений в группах. — Наука, 1989.