Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование
информационных систем

Системное программирование

Винник Екатерина Петровна

Реализация алгоритма построения машины Тьюринга по булевым грамматикам

Отчет по учебной практике

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

Санкт-Петербург
2021

Оглавление

Введение

Теория формальных языков лежит в основе таких направлений системного программирования, как разработка трансляторов и статических анализаторов. В связи с большой актуальностью этих направлений исследование формальных языков очень важно.

На данный момент среди формальных языков контекстно-свободные языки нашли наибольшее применение, к примеру, с их помощью задается основная грамматическая структура большого числа языков программирования. Однако уже с появлением самых первых языков программирования у контекстно-свободных языков не хватало выразительной силы, чтобы обеспечить некоторые их ограничения. Например, Роберт Флойд в статье [2] показал, что язык Алгол-60 не может быть полностью определен с помощью контекстно-свободных грамматик, приведя в пример следующую программу

$$\text{begin real } \mathrm{x}^{(\mathrm{n})};\ \mathrm{x}^{(\mathrm{n})} :=\ \mathrm{x}^{(\mathrm{n})};\ \text{end}$$

(где $\mathrm{x}^{(\mathrm{n})}$ есть n вхождений x), которая соответствует элементу языка $\{a^{(n)}b^{(n)}c^{(n)}|n \geq 0\}$). Данный язык не является контекстно-свободным, но существует булева грамматика, порождающая данный язык [5].

Для разрешения подобных описанной выше проблем разрабатывались различные обобщения контекстно-свободных грамматик – например, таким обобщением были двухуровневые грамматики, использовавшиеся в языке Алгол-68 [8]. Но для таких грамматик задача определения принадлежности слова языку является неразрешимой, тогда как для булевых грамматик, также являющихся обобщением контекстно-свободных, существует алгоритм кубической сложности, решающий данную задачу.

Так как булевы грамматики были представлены сравнительно недавно [4], многое в них подлежит исследованию. В последнее время для изучения формальных языков исследователи все чаще стали прибегать к самым различным областям математики, и в частности, перспективной областью, которая может применяться для исследований, являет-

ся теория групп. Однако, применение теоретико-группового аппарата для исследования формальных языков осложняется тем, что построение по формальной грамматике представления группы – задача, трудно решаемая вручную. На данный момент существует единственный проект – LangToGroup[1], предоставляющий возможность автоматически построить представление группы по данной контекстно-свободной формальной грамматике. Так как LangToGroup не предоставляет возможности исследовать булевы языки, добавление поддержки булевых грамматик в проект является актуальным и важным улучшением.

[1]Официальная документация и исходный код проекта могут быть найдены по ссылке `https://github.com/YaccConstructor/LangToGroup`
Дата последнего обращения: 15.05.2021

Постановка задачи

Целью данной учебной практики является предоставление возможности исследовать булевы граммматики с помощью теории групп в проекте LangToGroup.

Для достижения этой цели были поставлены следующие задачи:

- осуществить поддержку ввода грамматики, подлежащей исследованию, пользователем;
- разработать архитектуру машины Тьюринга, моделирующую алгоритм распознавания булевых языков;
- реализовать продуманную архитектуру в какой-либо из имеющихся в проекте нотаций машин Тьюринга;
- провести экспериментальное исследование полученной реализации.

1. Обзор

В данном разделе будут представлены основные понятия и технологии, необходимые для дальнейшего описания работы:

- определения и теоремы из теории формальных языков, использованные в исследовании;
- обзор существующей архитектуры проекта LangToGroup для дальнейшего корректного описания изменений, вносимых в нее в процессе исследования;
- обзор существующих технологических решений, применяющихся для реализации поддержки пользовательского ввода с целью обосновать принятые в исследовании технологические решения.

1.1. Формальные языки

В данном разделе будут введены основные определения и теоремы, используемые 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 терминальных символов Σ;*
- *конечное множество нетерминальных символов N, не пересекающееся с Σ;*
- *конечное множество правил R, где каждое правило может быть описано как $(\Sigma \cup N)^* N (\Sigma \cup N)^* \to (\Sigma \cup N)^*$;*
- *S – стартовый нетерминал.*

Определение 1.1.2 *Контекстно-свободной называется формальная грамматика, у которой в левой частях правил стоят единичные нетерминалы.*

Например, правилом контекстно-свободной формальной грамматики может являться правило вида $A_i \to B_1 B_2 ... B_n$.

Следующее определение машины Тьюринга может быть найдено в статье [3].

Определение 1.1.3 *В классической формализации машина Тьюринга может быть описана как семиместный кортеж* $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, *где:*

- Q – *конечный алфавит состояний;*
- Σ – *конечный алфавит символов;*
- Γ – *алфавит ленты,* $\Sigma \in \Gamma$*;*
- δ – *функция перехода. Значение функции* $\delta(q, X)$, *где* $q \in Q$ *и* $X \in \Gamma$ – *символ на ленте, в случае если оно определено, является трехместным кортежем* (p, Y, D), *где:*
  - p – *следующее состояние,* $p \in Q$*;*
  - Y – *символ из* Γ, *который будет записан на месте символа* X *ленты;*
  - D – *направление движения головы по ленте, может принимать значения* L *и* R *(соответствует движению влево и вправо соответственно).*
- q_0 – *стартовое состояние машины,* $q_0 \in Q$*;*
- B – *пустой символ;*
- F – *множество финальных (принимающих) состояний,* $F \in Q$.

Определение 1.1.4 *Машина Тьюринга имеет* k *лент,* k *голов и может быть описана как шестиместный кортеж* $M = (X, \Gamma, Q, \Theta, \bar{s_1}, \bar{s_2})$, *где:*

- X – *входной алфавит;*

- Γ – *алфавит лент;*
- $Q = \bigcup_{i=1}^{k} Q_i$ – *множество состояний голов на лентах машины;*
- Θ – *множество команд машины;*
- $\bar{s_1}$ – *k-вектор начальных состояний машины;*
- $\bar{s_2}$ – *k-вектор конечных состояний машины.*

Следующее определение применялось в реализации машины Тьюринга, изначально использовавшейся в проекте, и может быть найдено в статье [10].

Определение 1.1.5 *Симметризация машины Тьюринга M — это машина M^{sym} , полученная из M путем добавления обратной команды τ^{-1} в множество команд для каждой τ из M.*

Нотация этой машины Тьюринга может быть описана следующим образом: крайняя левая (правая) ячейка машины обозначается α (β). В каждый момент времени голова наблюдает две ячейки на ленте. Конфигурацией ленты является слово $\alpha uqvw$, где q – текущее состояние головы на ленте u (v) – слово слева (справа) от головы.

Также в настоящий момент в проекте реализована другая нотация машины Тьюринга, более подробное описание которой можно найти в статье [9]. С использованием этой нотации строится альтернативный алгоритм преобразования формальной грамматики в представление группы. Конфигурацией ленты в данной нотации является двухместный кортеж (X_1, Q_1), где X_1 – символ, Q_1 – состояние, и допустимы только продукции двух видов:

$$(X_1, Q_1) \to (X_2, Q_2)\ ,\ (X_1, Q_1) \to (L/R, Q_2)$$

Введем формальное определение машины Тьюринга, реализующей описанную нотацию. Рассмотрим два бесконечных списка букв: $s_0, s_1, ...$ и $q_1, q_2, ...$

Определение 1.1.6 *Машина Тьюринга – конечное множество четырехместных кортежей, каждый из которых может быть одного из трех видов:*

- $q_i s_j s_k q_l$;
- $q_i s_j R q_l$;
- $q_i s_j L q_l$.

для которого верно, что ни у каких двух кортежей первые две буквы не совпадают.

Следует отметить, что введенная таким образом машина Тьюринга является одноленточной детерминированной машиной.

Изложенные далее определения, необходимые для описания алгоритма распознавания булевых языков, можно найти в статье [4].

Определение 1.1.7 *Булевой грамматикой называется четырехместный кортеж $G = (\Sigma, N, R, S)$, который состоит из следующих компонентов:*

- *конечное множество терминальных символов Σ;*
- *конечное множество нетерминальных символов N, не пересекающееся с Σ;*
- *конечное множество правил R, где каждое правило имеет вид*

  $$A \to \alpha_1 \& \alpha_2 \& ... \& \alpha_m \& \neg\beta_1 \& \neg\beta_2 \& ... \& \neg\beta_n \ \ (m+n \geq 1, \alpha_i, \beta_j \in (\Sigma \cup N)^*)$$

  Части правила $A \to \alpha_i$, $A \to \neg\beta_j$ называются положительными и отрицательными конъюнктами соответственно.
  Считается, что входное слово может быть выведено с использованием правила $S \to A\&B\&\neg C$, если оно выводимо из A, выводимо из B и не выводимо из C;
- *S – стартовый нетерминал.*

Рассмотрим произвольную булеву грамматику, предварительно перенумеровав каждое из множеств правил, содержащих нетерминал A_i в своей левой части для последующего перебора алгоритмом. В правиле $A_i \to C_1B_1\&...\&C_kB_k$ часть правила A_1B_1 называется первой конъюнкцией, а C_kB_k – k-й. Введем следующие обозначения. Определим $N^+ = \{A^+ | A \in N\}$ – нетерминалы с приписанным справа знаком "$+$", $N^- = \{A^- | A \in N\}$ – нетерминалы с приписанным справа знаком "$-$", P' – множество правил, имеющих хотя бы одну конъюнкцию.

Определение 1.1.8 *Термы определяются следующим образом над алфавитом* $\Sigma \cup N \cup N^+ \cup N^- \cup P' \cup \{\text{“(”},\text{“)”}\}$*:*

- *для любого* $u \in \Sigma_+$ *и* $A \in N$ $A(u), A^+(u), A^-(u)$ *– термы;*
- *если* p_k *есть конъюнкция вида* $A \to BC$ *или* $A \to \neg BC$ *и* t_1, t_2 *термы для* B *и* C *соответственно, то* $p_k(t_1t_2)$ *является термом для* A*.*

Определение 1.1.9 *Строковое значение* $\sigma(t)$ *терма* t *определяется следующим образом:*

- $\sigma(A(u)) = \sigma(A^+(u)) = \sigma(A^-(u)) = u$ *для любых* $u \in \Sigma^+$*;*
- $\sigma(p_k(t_1t_2)) = \sigma(t_1) \cdot \sigma(t_2)$*.*

Определение 1.1.10 *Терм называется истинным в следующих случаях:*

- $A(u)$ *всегда истинный.* $A^+(u)$ *– истинный тогда и только тогда, когда* $u \in L(A)$*.* $A^-(u)$ *– истинный тогда и только тогда, когда* $u \notin L(A)$*;*
- $t = p_k(t_1t_2)$ *– истинный тогда и только тогда, когда выполняются все следующие условия:*
  - *оба терма* t_1*,* t_2 *истинны;*

– *для каждого длинного правила r для A (где A – нетерминал в левой части правила r), которое предшествует p, верно, что $\sigma(t) \notin L(r)$;*

– *для каждого конъюнкта p_i правила p, предшествующего p_k (т.е., $i < k$), справедливо $\sigma(t) \notin L(p_k)$;*

– *для каждой факторизации $\sigma(t_1)\sigma(t_2) = uv$ такой, что условие $0 < |u| < |\sigma(t_1)|$ выполняется, верно, что $u \notin L(B)$ или $v \notin L(C)$, где p_k – это $A \to BC$ или $A \to \neg BC$.*

Алгоритм распознавания булевого языка L, изложенный в [4], определяет систему переписывания входного слова w, которое на старте переписывается как терм $S(w)$, S – стартовый нетерминал грамматики. Результатом последовательности переписываний входного слова может являться либо терм $S^+(w)$, и тогда w является элементом языка L, либо $S^-(w)$ и тогда w языку L не принадлежит.

Определение 1.1.11 *Определим следующую систему переписываний терма $A(a)$:*

1. *Терм $A(a)$, где $a \in \Sigma$, переписывается как $A^+(a)$, если $A \to a \in P$ и переписывается как $A^-(a)$ иначе.*

2. *Терм $A(u)$, где слово $u = a_1 \cdots a_m$ и $m \geq 2$, переписывается как $p_1(B(a_1)C(a_2 \cdots a_m))$, где p является первым правилом для нетерминала A, и первая его конъюнкция есть $A \to BC$ или $A \to \neg BC$.*

3. *Терм $p_k(B^+(u)C^+(v))$ переписывается следующим образом:*

   - *если k-я конъюнкция правила p имеет вид $(A \to BC)$, то $(*)$:*

     – *если k-я конъюнкция в правиле p_k является последней, нетерминал A переписывается как $A^+(uv)$;*

     – *если в правиле больше, чем k конъюнкций, k-я конъюнкция переписывается следующей: $p_{k+1}(D(a_1)E(a_2 ... a_m))$, где $uv = a_1 a_2 ... a_m$ и $(k+1)$-я конъюнкция имеет вид $A \to DE$ или $A \to \neg DE$.*

- *если k-я конъюнкция правила p имеет вид $(A \to \neg BC)$, то $(**)$:*
  - *если p есть последнее правило для A, нетерминал переписывается как $A^-(uv)$;*
  - *если r – следующее правило для A после p, нетерминал переписывается как $r_1(D(a_1)E(a_2 \cdots a_m))$, где $uv = a_1a_2...a_m$ и первая конъюнкция правила r имеет вид $A \to DE$ или $A \to \neg DE$.*

4. *Любой из термов $p_k(B^+(u)C^-(v))$, $p_k(B^-(u)C^+(v))$ или $p_k(B^-(u)C^-(v))$ переписывается следующим образом:*
   - *если $|v| > 1$, то v представляется как $v = ax$ $(a \in \Sigma, x \in \Sigma^+)$, терм переписывается как $p_k(B(ua)C(x))$;*
   - *если $|v| = 1$ и k-я конъюнкция не имеет отрицания, применяются действия, описанные в $(**)$ выше;*
   - *если $|v| = 1$ и k-я конъюнкция имеет отрицание, применяются действия, описанные в $(*)$ выше.*

Теорема 1.1.1 *Приведенная система переписываний сохраняет истинность термов в процессе переписывания, терм $A(a)$ в конечном итоге будет переписан как $A^+(a)$, или $A^-(a)$.*

Из теоремы следует, что приведенная система переписываний может быть использована как распознаватель языка, порождаемого булевой грамматикой.

1.2. Обзор функциональности проекта

В данном разделе будет представлена архитектура проекта LangToGroup и некоторые детали ее реализации.

Проект LangToGroup состоит из взаимодействующих друг с другом модулей. На вход приложению подается контекстно-свободная грамматика, подлежащая исследованию, над которой производятся преоб-

разования по построению представления группы в соответствии с одним из двух реализованных алгоритмов.

Следует отметить, что алгоритмы используют разные нотации машин Тьюринга. В случае если используется машина, определенная в 1.1.4, над грамматикой проводятся следующие преобразования:

- строится машина Тьюринга из 1.1.4, распознающая данную контекстно-свободную грамматику;
- полученная машина Тьюринга симметризуется;
- к симметризованной машине Тьюринга применяется система переписывания, называемая S-машиной;
- по полученному из предыдущего пункта результату строится представление группы.

В случае если используется машина, определенная в 1.1.6, над грамматикой проводятся следующие преобразования:

- строится машина Тьюринга из 1.1.4, распознающая данную контекстно-свободную грамматику;
- полученная машина конвертируется в машину Тьюринга из 1.1.6;
- по конвертированной машине Тьюринга строится представление полугруппы;
- по представлению полугруппы строится представление группы.

Каждое преобразование реализовано отдельным модулем, более наглядное представление совместной работы которых можно увидеть на рисунке 1. Схема также отражает актуальность улучшений, осуществляемых в исследовании – отсутствие поддержки булевых грамматик и фиксированность входных данных, использующихся для построения группы (на данный момент построение группы осуществляется только для тестовых грамматик).

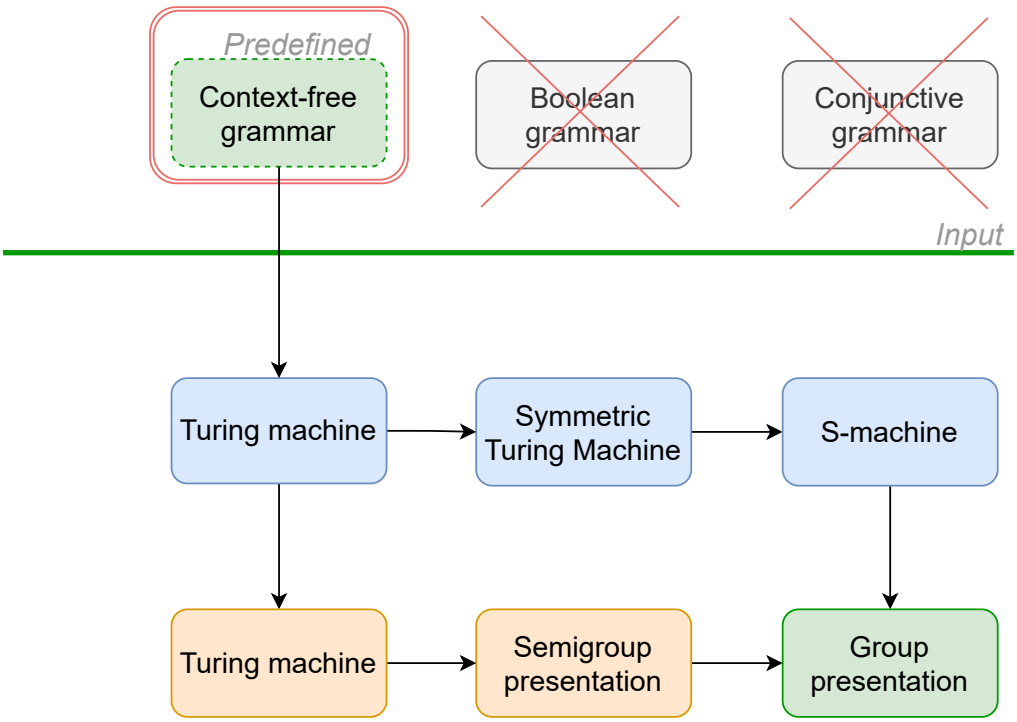

Рис. 1: Архитектура проекта LangToGroup

Разрабатывание архитектуры машины Тьюринга, моделирующей алгоритм определения принадлежности слова языку, порожденному данной булевой грамматикой, проводилось в терминах одноленточной детерминированной классической машины Тьюринга из определения 1.1.3. Так как машина Тьюринга из 1.1.6 также является одноленточной детерминированной машиной Тьюринга, и множество переходов в ней является подмножеством переходов классической машины, было принято решение выражать разработанную архитектуру в ее нотации. То есть, использовать для построения представления группы алгоритм, строящий сначала представление полугруппы.

Так как основным языком проекта является Haskell, разработки в рамках исследования проводились на этом языке.

1.3. Обзор технологий, использующихся для обеспечения ввода текстов грамматик пользователем

В данном разделе представлены существующие технологические подходы к реализации пользовательского ввода и обоснование выбранного подхода для обработки пользовательских грамматик.

Для реализации обработки пользовательского ввода используются различные синтаксические анализаторы. Существует несколько способов создания синтаксического анализатора – например, требующийся в разработке анализатор может быть сгенерирован *парсер-генератором*, либо использовать *парсер-комбинаторы*.

Парсер-генераторы удобны, так как при их использовании требуется только определить грамматику, для которой должен быть сгенерирован синтаксический анализатор. Но простота использования парсер-генератора приносит некоторые сложности в его использовании [11]: одной из них является то, что вводимая грамматика должна иметь форму, поддерживаемую парсер-генератором. Кроме того, на небольших грамматиках сгенерированный код может быть гораздо объемнее и сложнее кода, написанного разработчиком.

Утверждается [1], что использование парсер-комбинаторов является более гибким подходом, чем использование парсер-генераторов. Парсер-комбинаторы удобны в решении небольших задач: так как работа с ними заключается в использовании соответствующей библиотеки, работающей на основном языке проекта, код синтаксического анализатора является написанной разработчиком программой, которая может быть отлажена и структурирована в соответствии с имеющимися требованиями, в отличие от сгенерированного кода [7].

Так как для синтаксического анализа текстов грамматик не требуется сложная грамматика, и объем входных данных также не является большим, было принято решение использовать парсер-комбинаторы.

На данный момент для языка Haskell существует несколько библиотек парсер-комбинаторов. Важным критерием для выбора библиотеки являлось наличие хорошей системы сообщений об ошибках, так

как анализатор текстов грамматик относится к модулю общения с пользователем. Согласно сравнительному анализу[2], библиотека Megaparsec является наилучшей для целей проекта, так как предназначена для обработки небольших текстов, понимаемых человеком, а также снабжена хорошей системой сообщений об ошибках. По сравнению с библиотекой Parsec[3], ответвлением которой является Megaparsec, Megaparsec обладает более хорошей документацией, что также стало дополнительным преимуществом при выборе библиотеки.

[2]Сравнение с другими библиотеками может быть найдено на официальной странице Megaparsec по ссылке `https://hackage.haskell.org/package/megaparsec`

Дата последнего обращения 15.05.2021

[3]Официальная документация Parsec может быть найдена по ссылке `https://hackage.haskell.org/package/parsec`

Дата последнего обращения 15.05.2021

2. Описание реализации

В данном разделе представлено описание технологических и архитектурных решений, принятых для достижения поставленных задач: реализации поддержки ввода текстов грамматик пользователем и создания машины Тьюринга, моделирующей алгоритм распознавания булевого языка, порождаемого данной грамматикой. Обе поставленные задачи были реализованы, обновленная архитектура проекта представлена на рисунке 2.

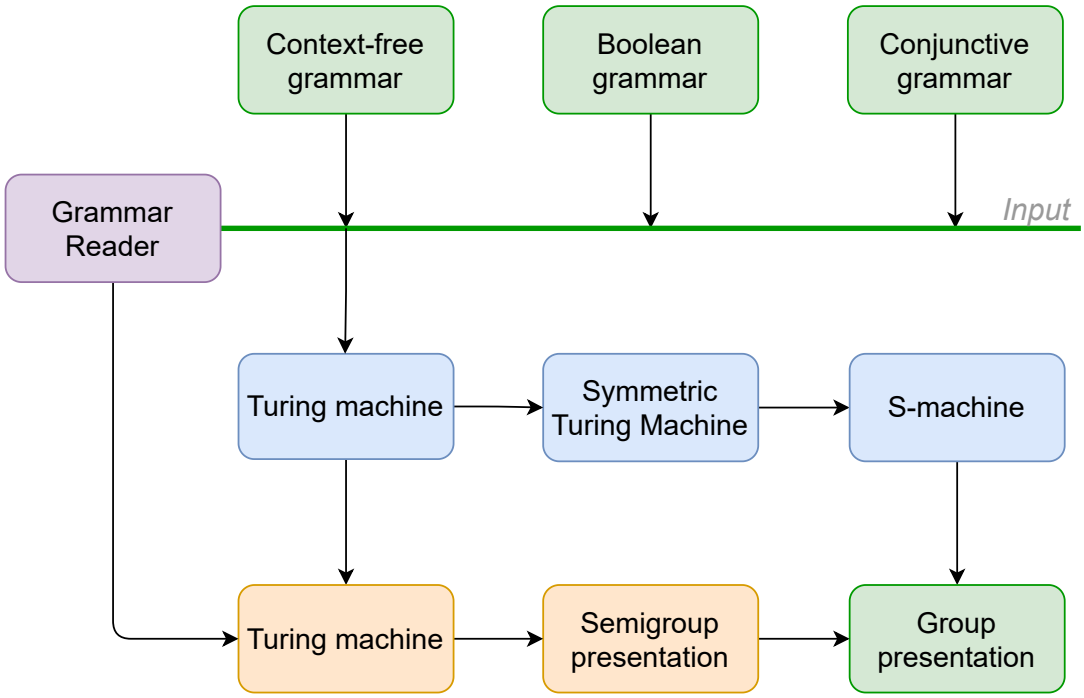

Рис. 2: Модернизированная архитектура проекта

Представленный на рисунке 2 модуль *GrammarReader* решает задачу обработки текстов грамматик. Также реализовано построение по данной грамматике машины Тьюринга из 1.1.6, распознающей язык, порождаемый грамматикой. Построение позволяет воспользоваться для исследования грамматик алгоритмом для построения представления группы через представление полугруппы и поддержать булевы, конъюнктивные грамматики в проекте. Более подробное описание реализованной функциональности будет приведено далее.

2.1. Реализация модуля обработки ввода грамматик пользователем

Данный модуль состоит из двух подмодулей – *Main* и *GrammarReader*. Модуль *GrammarReader* содержит в себе синтаксический анализатор текста грамматик. Результатом работы анализатора является грамматика, тип которой определен в ранее написанном модуле *GrammarType* следующим образом:

```
-- грамматика
newtype Grammar =
        Grammar (Set Nonterminal, -- множество нетерминалов
                 Set Terminal, -- множество терминалов
                 Set Relation, -- множество правил
                 StartSymbol) -- стартовый нетерминал
```

Рис. 3: Использующийся в проекте тип грамматики

Для поддержки булевых грамматик в систему типов, имеющуюся в *GrammarType*, был добавлен новый тип конъюнкции *Conj*, имеющий два конструктора – конструктор *PosConj {symbols :: [Symbol]}*, позволяющий задавать конъюнкции, не имеющие отрицания, и конструктор *NegConj {symbols :: [Symbol]}*, задающий конъюнкции с отрицанием в правилах булевых грамматик. Также в тип *Relation* был добавлен отдельный конструктор *BooleanRelation*, использующий введенный тип *Conj*, представленный на рисунке 4. Заданный таким образом тип *Relation* обеспечивает возможность задавать в булевой грамматике правила, соответствующие контекстно-свободной грамматике, как с помощью конструктора *BooleanRelation* (следует определить правую часть правила как список конъюнкций, состоящий из одной конъюнкции, не имеющей отрицания), так и с помощью ранее определенного конструктора *Relation*.

Синтаксический анализатор грамматик, именуемый *pGrammar*, состоит из комбинации синтаксических анализаторов, результатами работы

```haskell
-- тип, соответствующий конъюнкции
data Conj = PosConj {symbols :: [Symbol]}
          | NegConj {symbols :: [Symbol]}
    deriving (Eq, Ord, Show)
-- тип, соответствующий правилу грамматики
data Relation = Relation (Nonterminal, [Symbol])
          | BooleanRelation  (Nonterminal, [Conj])
    deriving (Eq, Ord, Show)
```

Рис. 4: Изменения, добавленные в модуль *GrammarType*

которых является множество терминалов, множество нетерминалов и множество правил соответственно:

```haskell
-- синтаксический анализатор грамматики
pGrammar :: Parser Grammar
-- синтаксический анализатор множества нетерминалов
pNonterminals :: Parser (Set Nonterminal)
-- синтаксический анализатор множества терминалов
pTerminals :: Parser (Set Terminal)
-- синтаксический анализатор множества правил
pRelations :: Parser (Set Relation)
```

Рис. 5: Комбинация синтаксических анализаторов, используемая в синтаксическом анализаторе грамматик

Main является точкой входа в приложение и осуществляет запуск синтаксического анализатора с параметрами – путями к файлу с текстом грамматики и файлу, в который следует сохранять сообщения об ошибках, возникающих в процессе обработки входной грамматики.

Примеры обработки пользовательских текстов можно увидеть в приложении А.

2.2. Архитектура машины Тьюринга, распознающей булевы грамматики

Так как в статье [4] не приведено описание устройства машины Тьюринга, эмулирующей алгоритм теоремы 1.1.1, создание архитектуры этой машины было важным этапом в исследовании. Разработанная машина Тьюринга является совокупностью подпрограмм, взаимодействующих друг с другом и изображена на схеме 6.

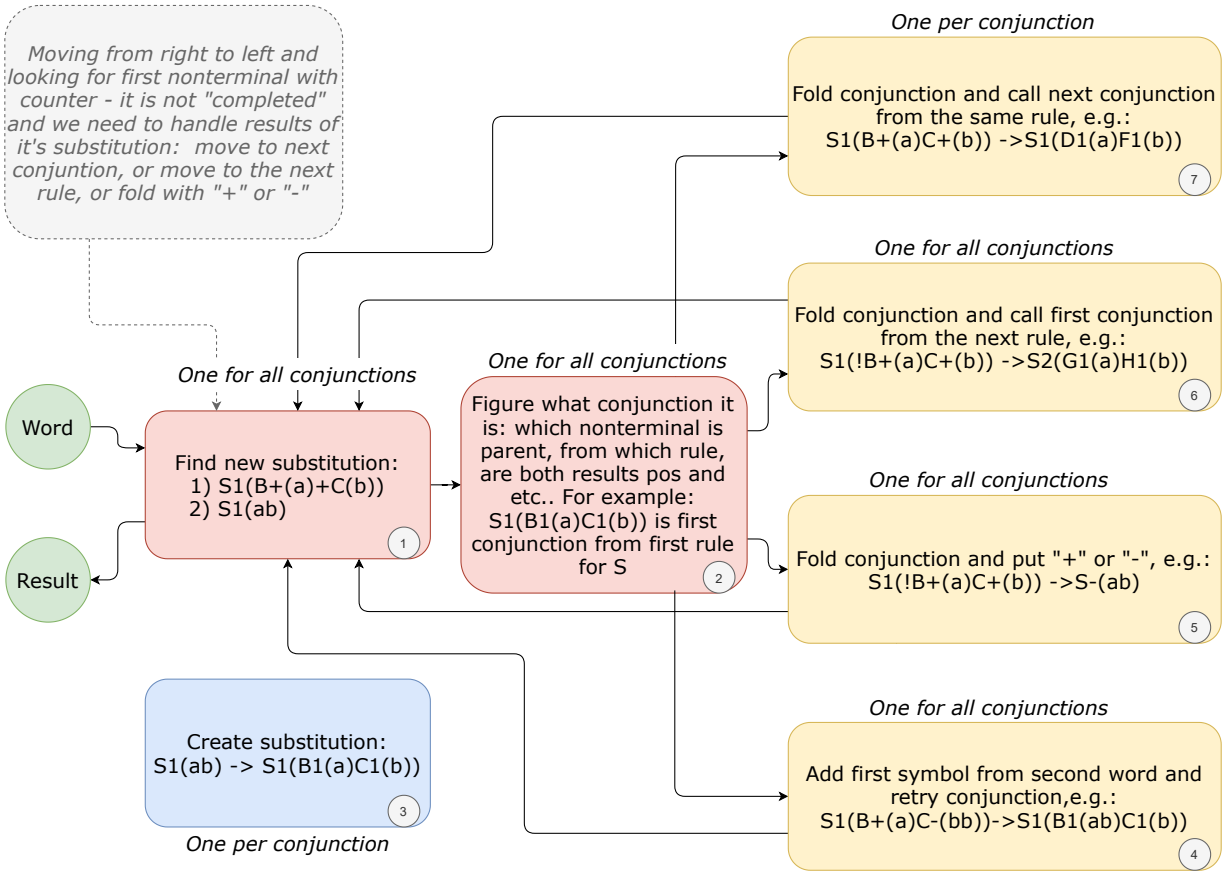

Рис. 6: Архитектура машины Тьюринга

Каждый из блоков, изображенный на схеме 6, соответствует определенной подпрограмме в машине Тьюринга. Некоторые из блоков, например, соответствующие номерам 1 и 2 на схеме, являются уникальными для машины Тьюринга и генерируются один раз. Также существуют блоки, соответствующие подпрограммам, которые генерируются в соответствии с некоторым параметром. Например, подпрограмма вызова первой конъюнкции в следующем правиле (соответствует номеру 6 на

схеме 6) генерируется для всех номеров правил, имеющихся у каждого из нетерминалов грамматики. Рассмотрим подробнее взаимодействие сгенерированных подпрограмм в процессе распознавания принадлежности данного слова булеву языку.

Представленная машина Тьюринга является одноленточной машиной-переписывателем, в точности повторяющей действия, описанные в алгоритме. При этом для каждого нетерминала S, выводимость из которого данного слова w проверяется, на соседней правой ячейке хранится счетчик, определяющий номер правила, применяемого в данный момент. Например, считая один символ соответствующим одной ячейке ленты машины, следующая запись

$$Si(w)$$

означает, что для слова w будет проверяться выводимость с помощью i-го правила для нетерминала S. Каждый из символов $S, i, (, w,)$ записан на отдельной ячейке. В дальнейшем для большей наглядности обозначений будем записывать последовательность Si двух ячеек из ячейки нетерминала и ячейки счетчика как S_i. В случае, если выводимость данного слова из нетерминала определена, счетчик на соседней с нетерминалом ячейке стирается, и ставится знак $"+"$, либо $"-"$. Для большей наглядности обозначений будем записывать последовательность из двух ячеек нетерминала и результата выводимости как $S^+(w)$, либо $S^-(w)$ по аналогии с $S_i(w)$.

Номер конъюнкции по номеру правила и участвующим в ней нетерминалам определяется однозначно.

Каждой конъюнкции i-го правила грамматики соответствует отдельная программа, помеченная номером 3 на схеме 6, переписывающая терм $S_i(w)$, где слово $w = ab$, этой конъюнкцией. Основной программой, запускающей другие программы по переписыванию слова, является *findNewSubstitution*, помеченная на схеме номером 1.

Пусть над словом была произведена некоторая последовательность переписываний. Тогда у нетерминалов, для которых выводимость данного слова была определена, на соседней правой ячейке ленты записан

знак "+" или "−". У нетерминалов, выводимость из которых слова еще требуется определить, на соседней правой ячейке находится счетчик, отмечающий номер применяемого правила. Программа *findNewSubstitution* двигает голову машины от конца слова к началу и, найдя первый нетерминал, выводимость из которого слова не определена, запускает далее одну из следующих программ:

- в случае, если найденный нетерминал имеет вид $S_i(w)$, запускается программа *createNewSubstitution*, помеченная номером 3 на схеме;

- в случае, если найденный нетерминал имеет вид $S_i(B^+(a)C^-(b))$, $S_i(B^-(a)C^-(b))$, $S_i(B^-(a)C^+(b))$, либо $S_i(B^+(a)C^+(b))$, запускается вспомогательная программа (соответствует блоку с номером 2 на схеме) по определению номера конъюнкции, вида конъюнкции ($S_i(B^+(a)C^-(b))$, или др.) и длины слова b второго нетерминала. В зависимости от собранной информации вызывается одна из следующих программ:

  - программа записи результата в терм S_i, соответствующая блоку с номером 5 на схеме, переписывающая рассматриваемую конъюнкцию $S_i(B^+(a)C^-(b))$ результатом $S^+(ab)$, либо $S^-(ab)$. Данная программа запускается в следующих случаях:

    * если конъюнкция не имеет отрицания, имеет вид

      $$S_i(B^+(a)C^-(b)), S_i(B^-(a)C^-(b)), S_i(B^-(a)C^+(b)),$$

      правило S_i – последнее правило для рассматриваемого нетерминала и $|b| = 1$, запускается переписывание $S^-(ab)$ (см. систему переписываний 1.1.11, пункт 4);

    * если конъюнкция имеет вид $S_i(B^+(a)C^+(b))$, не имеет отрицания и является последней в правиле, запускается переписывание $S^+(ab)$ (см. систему переписываний 1.1.11, пункт 3);

* если конъюнкция имеет отрицание, является конъюнкцией вида $S_i(B^+(a)C^+(b))$ и применяемое правило является последним, запускается переписывание $S^-(ab)$ (см. систему переписываний 1.1.11, пункт 3).

– программа перебора слов a и b, помеченная номером 4 на схеме, которая запускается в случае если конъюнкция не имеет отрицания, является конъюнкцией одного из видов

$$S_i(B^+(a)C^-(b)), S_i(B^-(a)C^-(b)), S_i(B^-(a)C^+(b)),$$

и $|b| > 1$ (см. систему переписываний 1.1.11, пункт 4);

– программа, осуществляющая переход к следующей конъюнкции того же правила, соответствующая блоку с номером 7 на схеме. Запускается в следующих случаях:

* если конъюнкция имеет вид $S_i(B^+(a)C^-(b))$, $S_i(B^-(a)C^-(b))$ или $S_i(B^-(a)C^+(b))$, не имеет отрицания и $|b| = 1$;
* если конъюнкция имеет вид $S_i(B^+(a)C^+(b))$ и не имеет отрицания.

– программа, осуществляющая переход к первой конъюнкции следующего правила, помеченная номером 6 на схеме. Запускается в следующих случаях:

* если конъюнкция вида $S_i(B^+(a)C^-(b))$, $S_i(B^-(a)C^-(b))$ или $S_i(B^-(a)C^+(b))$ не имеет отрицания, правило S_i не является последним для рассматриваемого нетерминала и $|b| = 1$;
* если конъюнкция вида $S_i(B^+(a)C^+(b))$ имеет отрицание и правило S_i не является последним.

После выполнения каждая из перечисленных программ снова вызывает *findNewSubstitution*, порождая поиск новых нетерминалов, которые необходимо заменить. Если в процессе выполнения программы *findNewSubstitution* голова машины была сдвинута до начала слова,

нетерминалов, требующих замены в слове больше нет, все подстановки применены, алгоритм заканчивает свою работу.

Для большей наглядности далее приведена диаграмма последовательностей 7 данного алгоритма. Блоки с названиями *program1*, *program2*, *program3*, *program4* представляют не одну программу, а совокупность сгенерированных программ для каждого из случаев, описанных в блоках с номерами 4, 5, 6, 7 на предыдущей схеме 6.

Пример работы алгоритма приведен в приложении В.

2.3. Детали реализации

В процессе реализации поставленных задач использовались дополнительные инструменты для улучшения качества программного кода, не упомянутые в предыдущих разделах. Так, для проверки корректности реализованных модулей обработчика пользовательских текстов грамматик и построения машины Тьюринга по данной булевой грамматике было сгенерировано более двухсот пятидесяти тестов с использованием HUnit[4], как модульных, так и интеграционных. Также в сборку был добавлен HLint[5], проверяющий имеющийся в проекте код на возможные улучшения, например, на возможность использования альтернативных функций, удаления избыточностей, упрощения кода.

[4]Официальная документация может быть найдена по ссылке `https://hackage.haskell.org/package/HUnit`
Дата последнего обращения 15.05.2021

[5]Официальная документация и исходный код проекта могут быть найдены по ссылке `https://github.com/ndmitchell/hlint`
Дата последнего обращения 15.05.2021

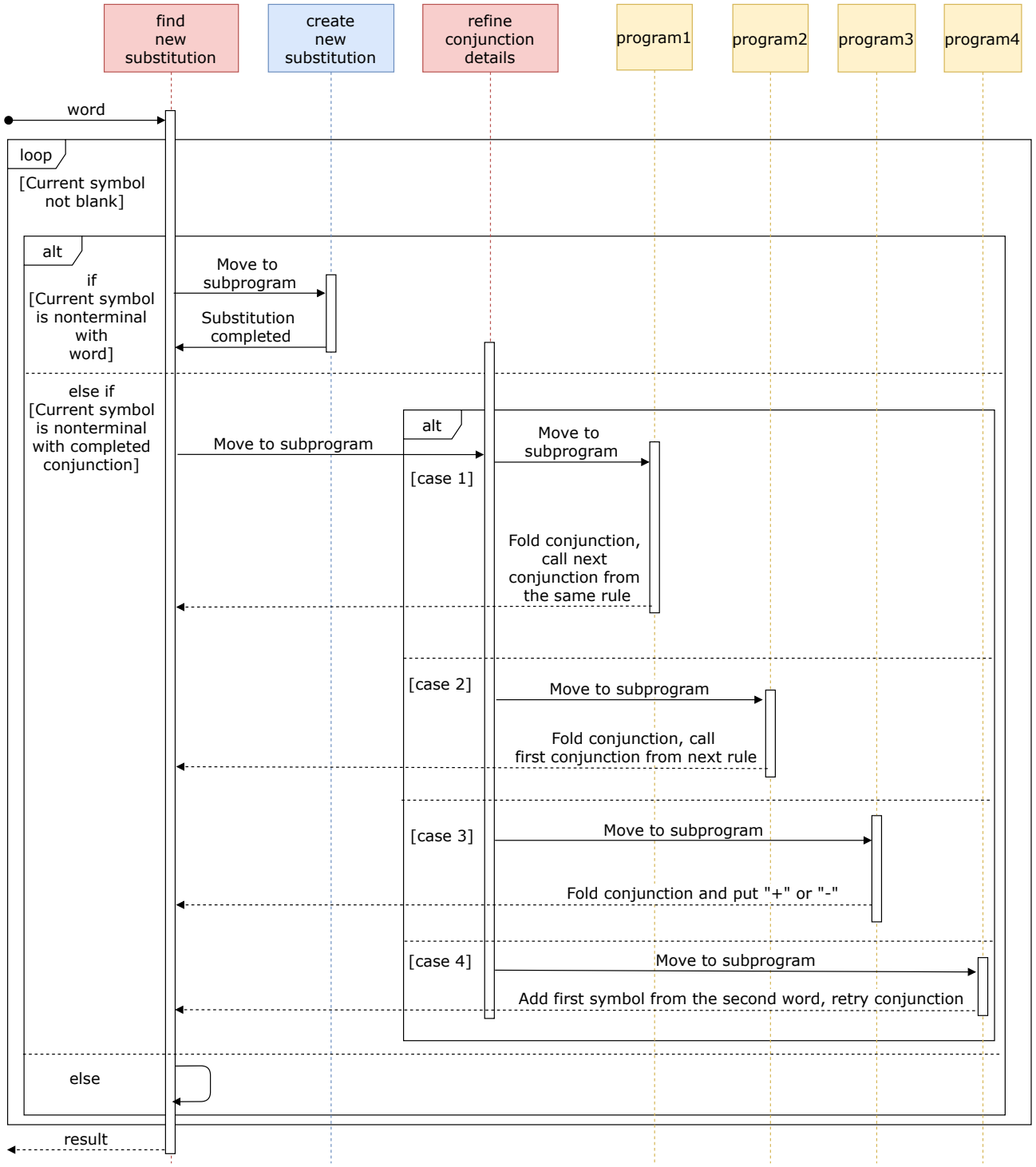

Рис. 7: Диаграмма последовательностей алгоритма построения машины Тьюринга

3. Эксперименты

Для того, чтобы оценить, насколько применима в исследованиях формальных языков с помощью теории групп реализованная поддержка булевых грамматик, был проведен ряд экспериментов. Для нескольких булевых грамматик, взятых из статьи [6], была построена машина Тьюринга, распознающая язык, порождаемый данной грамматикой, и представление группы.

Результаты работы алгоритма представлены в таблице 1. В столбце N представлено количество правил в грамматике, в столбце TM количество правил в сгенерированной машине Тьюринга, в столбце A количество групповых отношений.

Язык	Тип грамматики	N	TM	A
$\{a^n \mid n \in \mathbb{N}\}$	контекстно-свободная	2	494	150360
$\{a^n b^n \mid n \in \mathbb{N}\}$	контекстно-свободная	5	1717	719700
$\{a^n b^n c^n \mid n \in \mathbb{N}\}$	конъюнктивная	14	8236	5620800
$\{ww \mid w \in \{a,b\}^*\}$	булева	14	6110	3692580
$\{a^m b^n c^n \mid (m \neq n), m, n \in \mathbb{N}\}$	булева	14	8247	6000819

Таблица 1: Количества правил в машинах Тьюринга и количества групповых отношений для разных грамматик

Из таблицы 1 видно, что даже для небольших грамматик строятся машины Тьюринга, имеющие большое количество правил, по которым получается огромное представление группы. Следует отметить, что дополнительную сложность вносит приведение грамматики к нормальной форме, с которой работает алгоритм построения машины, так как приведение к нормальной форме увеличивает размер грамматики, и вследствие, количество правил построенной по ней машины.

Так как построенные представления группы велики, не представляется возможности исследовать их с помощью какого-либо математического пакета.

Заключение

В ходе выполнения данной работы в проекте LangToGroup[6] были достигнуты следующие результаты:

- осуществлена поддержка ввода грамматики, подлежащей исследованию, пользователем;
- разработана архитектура машины Тьюринга, моделирующей алгоритм распознавания булевого языка, порождаемого данной грамматикой;
- реализована генерация машины Тьюринга, моделирующей алгоритм распознавания булевого языка, порождаемого данной грамматикой;
- проведено экспериментальное исследование полученной реализации.

Так как описанное решение в данный момент не имеет своего применения ввиду чрезмерно большой мощности получаемых машин, существует потребность в его улучшении: можно модернизировать уже созданную архитектуру с целью избавления от возможных избыточностей, либо пересмотреть алгоритм 1.1.11, данный в статье [4] и попробовать выразить его в терминах недетерминированной одноленточной машины Тьюринга. Второй из предложенных подходов обладает потенциальным недостатком: так как на данный момент для построения представления группы по булевой грамматике используется алгоритм, использующий машину Тьюринга из 1.1.6, которая является одноленточной детерминированной машиной, для недетерминированной машины потребуются дополнительные преобразования, приводящие ее к виду машины, использующейся в алгоритме построения представления группы, что может привести к неуменьшению количества правил в получаемых машинах.

[6]Официальная документация и исходный код проекта могут быть найдены по ссылке `https://github.com/YaccConstructor/LangToGroup`

Следует отметить, что хотя булевы грамматики в проекте на данный момент не могут быть исследованы с помощью теории групп, остается возможность использовать реализованную функциональность для определения принадлежности слов языку. Так как из-за новизны булевых грамматик существует немного интерпретаторов, позволяющих распознавать языки, порождаемые булевыми грамматиками, реализованное решение может оказаться весьма полезным.

Список литературы

[1] Easy parsing with parser combinators. — Access mode: https://www.lihaoyi.com/post/EasyParsingwithParserCombinators.html (online; accessed: 2021-05-15).

[2] Floyd Robert W. On the Nonexistence of a Phrase Structure Grammar for ALGOL 60 // Commun. ACM. — 1962. — Sep. — Vol. 5, no. 9. — P. 483–484. — Access mode: https://doi.org/10.1145/368834.368898.

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

[4] Okhotin Alexander. Boolean Grammars // Proceedings of the 7th International Conference on Developments in Language Theory. — DLT'03. — Berlin, Heidelberg : Springer-Verlag, 2003. — P. 398–410.

[5] Okhotin Alexander. On the existence of a Boolean grammar for a simple programming language. — 2005. — 01.

[6] Okhotin Alexander. Conjunctive and Boolean grammars: The true general case of the context-free grammars // Computer Science Review. — 2013. — 08. — Vol. 9. — P. 27–59.

[7] Per criterion comparison of Parser Generators and Parser Combinators. — Access mode: https://gist.github.com/chshersh/27844477752359735dfa41ac184d3bf2 (online; accessed: 2021-05-15).

[8] Revised Report on the Algorithmic Language ALGOL 68 / A. van Wijngaarcien, B. J. Mailloux, J. E. L. Peck et al. // SIGPLAN Not. — 1977. — May. — Vol. 12, no. 5. — P. 1–70. — Access mode: https://doi.org/10.1145/954652.1781176.

[9] Rotman Joseph J. An Introduction to the Theory of Groups. —

Springer, New York, NY, 1995. — ISBN: 978-1-4612-4176-8. — Access mode: https://doi.org/10.1007/978-1-4612-4176-8.

[10] Sapir Mark, Birget Jean-Camille, Rips Eliyahu. Isoperimetric and Isodiametric Functions of Groups // Annals of Mathematics. — 1998. — 12. — Vol. 156.

[11] Why I do not use Parser generator. — Access mode: https://mortoray.com/2012/07/20/why-i-dont-use-a-parser-generator/ (online; accessed: 2021-05-15).

A. Обработка текста грамматики

В данном разделе представлены примеры обработки пользовательских грамматик, поддерживаемых в проекте: булевой и контекстно-свободной.

Входная грамматика

$S;\ S\ Sa;\ c\ v\ b$

$S \rightarrow\ c\&!\ v$

$Sa \rightarrow!\ b$

$S \rightarrow\ Sa\&!\ Eps$

Результат обработки

```
Grammar (
    fromList [
        Nonterminal {nonterminalValue = "S"},
        Nonterminal {nonterminalValue = "Sa"}],
    fromList [
        Terminal {terminalValue = "b"},
        Terminal {terminalValue = "c"},
        Terminal {terminalValue = "v"}],
    fromList [
    BooleanRelation (Nonterminal {nonterminalValue = "S"},
        [PosConj {symbols =
            [T (Terminal {terminalValue = "c"})]},
        NegConj {symbols =
            [T (Terminal {terminalValue = "v"})]}]),
    BooleanRelation (Nonterminal {nonterminalValue = "S"},
        [PosConj {symbols =
            [N (Nonterminal {nonterminalValue = "Sa"})]},
        NegConj {symbols =
            [N (Nonterminal {nonterminalValue = "Eps"})]}]),
    BooleanRelation (Nonterminal {nonterminalValue = "Sa"},
        NegConj {symbols =
            [T (Terminal {terminalValue = "b"})]}])],
    Nonterminal {nonterminalValue = "S"})
```

Входная грамматика

$S;\ S\ B;\ a\ b$

$S \rightarrow\ a\ B$

$B \rightarrow\ b$

Результат обработки

```
Grammar (
    fromList [
        Nonterminal {nonterminalValue = "B"},
        Nonterminal {nonterminalValue = "S"}],
    fromList [
        Terminal {terminalValue = "a"},
        Terminal {terminalValue = "b"}],
    fromList [
        Relation (Nonterminal {nonterminalValue = "B"},
            [T (Terminal {terminalValue = "b"})]),
        Relation (Nonterminal {nonterminalValue = "S"},
            [T (Terminal {terminalValue = "a"}),
            N (Nonterminal {nonterminalValue = "B"})])],
    Nonterminal {nonterminalValue = "S"})
```

В. Пример работы алгоритма

Входная грамматика

$S \to AB\&!DC$
$A \to A_1A$
$A \to a$
$A_1 \to a$
$B \to B_1B_2$
$B_1 \to b$
$B_2 \to BC_1$
$C_1 \to c$
$B \to B_1C_1$
$C \to C_1C$
$C \to c$
$D \to A_1D_1$
$D_1 \to DB_1$
$D \to A_1B_1$

Рассматриваемая грамматика порождает язык $\{a^m b^n c^n \mid m \neq n\}$.

Для данной грамматики будут сгенерированы следующие подпрограммы:

- программа по поиску новой подстановки, соответствующая блоку с номером 1 на схеме 6;
- программа по уточнению деталей рассматриваемой конъюнкции, соответствующая блоку с номером 2 на схеме 6;
- программы, осуществляющие подстановку конъюнкции, соответствующие блоку с номером 3 на схеме 6 для каждой конъюнкции каждого правила – в данном примере будет сгенерировано 10 программ;
- программы, осуществляющие переход от i к $i+1$ конъюнкции в правиле k, соответствующие блоку с номером 7 на схеме 6 – в данном примере будет сгенерирована одна такая программа для конъюнкции AB;
- программы, осуществляющие переход от i конъюнкции к первой конъюнкции в следующем правиле, либо к правилу вида $A \to a$, соответствующие блоку с номером 6 на схеме 6 – в данном примере будет сгенерировано 4 таких программы (для конъюнкций $A_1A, B_1B_2, C_1C, A_1D_1$);

- программа, заменяющая подстановку $S_i(A^{+/-}(w_l)B^{+/-}(w_m))$ на результат $S^{+/-}(w_l w_m)$, соответствующая блоку с номером 5 на схеме 6;

- программа, меняющая разбиение слова на два подслова, соответствующая блоку с номером 4 на схеме 6.

Рассмотрим пример работы алгоритма на слове *aabc*. Для краткости шаги по переписыванию символов ленты опущены и оставлены только результаты подстановок. Следует отметить, что в выбранных обозначениях:

- B_1 – нетерминал;

- Bi – две ячейки ленты, первая из которых – нетерминал, для которого проверяется выводимость слова из i-го правила, i – счетчик, размещенный на одну ячейку правее на ленте от ячейки с нетерминалом;

- $B_1 i$ – две ячейки ленты, где первая ячейка – нетерминал B_1, для которого проверяется выводимость слова из i-го правила, счетчик i записан на второй ячейке.

Ниже представлена работа алгоритма по распознаванию слова *aabc*.
$S1(A1(a)B1(abc))$
$S1(A1(a)B1(B_1 1(a)B_2 1(bc)))$
$S1(A1(a)B1(B_1 1(a)B_2 1(B1(b)C_1 1(c))))$
$S1(A1(a)B1(B_1 1(a)B_2 1(B-(b)C_1+(c))))$
$S1(A1(a)B1(B_1 1(a)B_2-(bc)))$
$S1(A1(a)B1(B_1-(a)B_2-(bc)))$
$S1(A1(a)B2(B_1 1(a)C_1 1(bc)))$
$S1(A1(a)B2(B_1 1(a)C_1-(bc)))$
$S1(A1(a)B2(B_1-(a)C_1-(bc)))$
$S1(A1(a)B2(B_1-(a)C_1-(bc)))$
$S1(A1(a)B-(abc))$
$S1(A+(a)B-(abc))$

$S1(A1(aa)B1(bc))$
$S1(A1(aa)B1(B_1 1(b)B_2 1(c)))$
$S1(A1(aa)B1(B_1 1(b)B_2 -(c)))$
$S1(A1(aa)B1(B_1 +(b)B_2 -(c)))$
$S1(A1(aa)B2(B_1 1(b)C_1 1(c)))$
$S1(A1(aa)B2(B_1 1(b)C_1 +(c)))$
$S1(A1(aa)B2(B_1 +(b)C_1 +(c)))$
$S1(A1(aa)B+(bc))$
$S1(A1(A_1 1(a)A1(a))B+(bc))$
$S1(A1(A_1 1(a)A+(a))B+(bc))$
$S1(A1(A_1 +(a)A+(a))B+(bc))$
$S1(A+(aa)B+(bc))$
$S1(A+(aa)B+(bc))$
$S1(!D1(a)C1(abc))$
$S1(!D1(a)C1(C_1 1(a)C1(bc)))$
$S1(!D1(a)C1(C_1 1(a)C-(bc)))$
$S1(!D1(a)C1(C_1 -(a)C-(bc)))$
$S1(!D1(a)C-(abc))$
$S1(!D-(a)C-(abc))$
$S+(aabc)$