информатика

Компьютерные инструменты в образовании, 2024
№ 2: 72–84
УДК: 004.43
http://cte.eltech.ru
doi:10.32603/2071-2340-2024-2-72-84

ПРОГРАММНЫЙ МОДУЛЬ ДЛЯ РАБОТЫ С КОНТЕКСТНО-СВОБОДНЫМИ ГРАММАТИКАМИ

Жегалин М. А.[1], студент, ✉ michael.zhegalin.univ@gmail.com
Яндринский В. В.[1], студент, yandrinsky@gmail.com

[1]Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В. И. Ульянова (Ленина), ул. Профессора Попова, 5, 197022, Санкт-Петербург, Россия

Аннотация

В статье представлена разработка учебного программного модуля для поддержки начального этапа обучения формальным грамматикам. Разработанный модуль позволяет студенту или преподавателю описывать заданный язык посредством контекстно-свободной грамматики. В процессе выполнения учебного задания по написанию грамматики модуль в интерактивном режиме обеспечивает реакцию на отличия построенной грамматики от эталонной. Основная техническая проблема, которую решали авторы, связана с тем, что не существует алгоритма определения эквивалентности двух контекстно-свободных грамматик. Поэтому модуль должен в реальном времени генерировать достаточно большое число слов языка различной длины, чтобы с большой вероятностью обнаружить отличия предложенной грамматики от эталонной. В работе было проанализировано несколько алгоритмов генерации слов, после чего выбран алгоритм, наиболее соответствующий условиям использования модуля. Разработанный модуль можно использовать для проведения олимпиад по дискретной математике и информатике у школьников и студентов, а также для поддержки самостоятельной работы студентов над темой «Формальные языки и грамматики».

Ключевые слова: *учебный модуль, грамматики, алгоритм Эрли, контекстно-свободные грамматики, сравнение грамматик.*

Цитирование: Жегалин М. А., Яндринский В. В. Программный модуль для работы с контекстно-свободными грамматиками // Компьютерные инструменты в образовании. 2024. № 2. С. 72–84. doi:10.32603/2071-2340-2024-2-72-84

1. ВВЕДЕНИЕ

Грамматики широко применяются в различных областях компьютерных наук и информационных технологий. Они имеют фундаментальное значение при разработке кода, анализе текстов и для множества других задач, которые не могут происходить без символов и слов.

Большинство языков программирования описываются с помощью контекстно-свободных грамматик. Например, грамматику языка Си или Python можно представить в виде контекстно-свободной грамматики. Это позволяет компиляторам и интер-

претаторам языков программирования анализировать и проверять синтаксическую корректность программ. Грамматики могут использоваться для автоматической генерации кода или шаблонов. В области обработки и анализа данных грамматики могут применяться для анализа структуры и валидации входных данных.

Перед авторами стояла задача создания модуля учебной лаборатории, который бы позволял в удобном формате работать с построением контекстно-свободных грамматик, генерацией их слов и сравнением грамматик между собой. В этой лаборатории пользователь имеет возможность вводить свою грамматику, составлять по ней контрольные примеры и проверять их на принадлежность к грамматике.

В качестве критериев совпадения грамматик выбраны следующие параметры: процент принадлежности слов, которые построила грамматика составителя задания, к грамматике пользователя и процент принадлежности слов, которые построила грамматика пользователя, к грамматике составителя задания.

2. ФОРМАЛЬНЫЕ ГРАММАТИКИ

Изучение формальных грамматик было заложено трудами Ноама Хомского — американского лингвиста, публициста и теоретика. В 50-х годах XX века он совершил прорыв в лингвистике благодаря своей работе «Синтаксические структуры» [1]. Многие называли это событие «Хомскианской революцией». Книга стала основой теории порождающих грамматик и теории формальных языков. В отличие от других авторов, Хомский предложил исследовать общую теорию построения языков, а не только сосредотачивать внимание на изучении устройства какого-то конкретного языка.

Грамматикой (формальной грамматикой) мы называем любое конечное описание языка. Например, грамматика $L = \{a^n b^n\}$ (здесь n — натуральное число) задает язык L, состоящий из цепочек вида $ab, aabb, aaabbb$ и т. д. Назначение грамматики — задание языка.

В работе Хомского представлена иерархия формальных грамматик. В ней языки разделелены на четыре типа в зависимости от их условной сложности:

0 — неограниченные,

1 — контекстно-зависимые и неукорачивающие,

2 — контекстно-свободные,

3 — регулярные.

Разработанный модуль предназначен для работы с контекстно-свободными грамматиками (КС-грамматиками).

Грамматика определяет формальный язык L, который является множеством слов некоторого алфавита Σ, его элементы называют терминальными символами или *терминалами.* Чтобы задавать рекурсивные определения, мы вводим дополнительный алфавит N специальных (вспомогательных) символов. Нетерминалы используются при грамматическом разборе слов или предложений. Например, при разборе предложений русского языка используются такие нетерминалы как «подлежащее», «сказуемое», «дополнение», а при разборе слов — «приставка», «корень», суффикс», «окончание». Также в контекстно-свободных грамматиках используется понятие *стартового нетерминала*, который принято обозначать буквой S. Он соответствует всему языку в целом, с него начинается грамматический разбор. В приведенном примере русского языка это будут нетерминалы «предложение» и «слово» соответственно. Главным элемен-

том контекстно-свободной грамматики является конечное множество P — *правила грамматики.*

Правила контекстно-свободной грамматики выглядят следующим специальным образом:

— левая часть каждого правила является нетерминалом,
— правая часть правила — цепочка терминалов и нетерминалов.

Более формально: всякое правило из P имеет вид $A \to \beta$, где $A \in N, \beta \in \{\Sigma \cup N^*\}$.

Приведем пример описания КС-грамматикой правильной скобочной последовательности (в приведенных ниже записях запятая не относится к множеству терминальных символов):

$\Sigma = \{(,)\}$;
$N = \{S\}$ — стартовый нетерминал;
$P = \{S \to SS, S \to (S), S \to \varepsilon\}, \varepsilon$ — пустой символ.

Пример получения скобочной последовательности (())() из правил грамматики:

$$S \to SS \to (S)S \to ((S))S \to (())S \to (())(S) \to (())().$$

Языки, созданные на основе КС-грамматик, распознаются с помощью автоматов с магазинной памятью (pda — pushdown automaton) [2]. Магазинный автомат имеет рабочую память — магазин, в который записываются символы из алфавита магазинных символов. Каждый шаг работы магазинного автомата определяется текущим состоянием управления, входным символом, либо без него (ε-шаги) и верхним символом в стеке магазина. При выполнении одного шага магазинного автомата верхний символ стека заменяется указанной цепочкой символов, в том числе пустой (таким образом, стирается верхний символ стека), и происходит переход в новое состояние управления.

Текущим входным символом становится следующий символ на входной ленте, если происходит шаг, зависящий от входного символа, или текущий входной символ остается без изменений в случае ε-шага.

Изначально в стеке магазина находится только один символ — начальный символ. Цепочка считается принятой, если магазинный автомат, начиная с начального состояния управления, с начальным символом в стеке и считав данную цепочку на входе, завершает работу в одном из конечных состояний или опустошает магазин. Каждый конкретный магазинный автомат принимает входную цепочку, используя только один из указанных признаков принятия цепочки.

3. ОБЗОР СРЕДСТВ ДЛЯ РАБОТЫ С ГРАММАТИКАМИ

Рассмотрим существующие средства для работы с грамматиками и языками. Перед тем как перейти к ним, дадим определение *DSL (Domain-Specific Languages)* — предметно-ориентированных языков. Это компьютерные языки, специализированные для конкретной области применения. В отличие от широко применимых языков программирования Python, Java, C++ и прочих, DSL учитывают особенности конкретной сферы знаний, для работы с которой они создавались. Например, среди DSL есть множество широко употребляемых языков, таких как TeX/LaTeX для подготовки (компьютерной вёрстки) текстовых документов, SQL для СУБД, HTML для разметки документов, Verilog и VHDL для описания аппаратного обеспечения, Excel для работы с электронными таблицами.

Одним из основных инструментов для работы с языками и грамматиками является **JetBrains MPS (Meta Programming System)** — метаязыковая платформа компании JetBrains для создания языков программирования и моделирования, а также для разработки генераторов кода и других инструментов автоматизации [3]. Этот инструмент базируется на концепции Language-Oriented Programming (LOP) и использует метамоделирование для создания DSL. Разработчики создают метамодели, описывающие синтаксис и семантику языков программирования, и на их основе генерируют код. MPS предоставляет графический редактор для создания DSL, интегрированный с IDE.

Код хранится в так называемом *абстрактном синтаксическом дереве (AST)*, структуре данных, состоящей из узлов с набором атрибутов (свойств, дочерних элементов и ссылок), которые полностью описывают программный код. Задача редактора MPS состоит в том, чтобы визуализировать AST в удобной для пользователя форме и предоставить средства для его эффективного редактирования.

При создании языка в MPS разработчик определяет правила редактирования кода и указывает, как код отображается пользователю. Также можно указать систему типов языка и ограничения как набор правил, применимых к языку. MPS использует генеративный подход. Это означает, что разработчик может определить генераторы для своего языка, чтобы преобразовывать вводимые конечным пользователем данные в более традиционный язык, обычно это язык общего назначения. В настоящее время MPS особенно хорош в создании Java-кода, но не ограничивается этим, также можно использовать C, C#, XML, FHTML, PDF, LaTeX, JavaScript и т. д.

Существуют и другие программные продукты, ориентированные на работу с грамматиками.

Xtext — это инструментарий, разработанный компанией Eclipse Foundation, который позволяет создавать DSL на основе грамматики с использованием языка Xtext [4]. Xtext предоставляет возможность определения синтаксиса DSL, автоматической генерации редакторов и поддержки различных языков программирования. В отличие от стандартных генераторов парсеров, Xtext генерирует не только парсер, но и модель классов для абстрактного синтаксического дерева, а также предоставляет полнофункциональную, настраиваемую IDE на основе Eclipse. Чтобы указать язык, разработчик должен написать грамматику на языке грамматики Xtext. Эта грамматика описывает, как модель Ecore получается из текстовой записи. Из этого определения генератор кода получает анализатор и классы объектной модели. Оба могут использоваться независимо от Eclipse.

ANTLR (ANother Tool for Language Recognition) — это инструмент для создания парсеров и генерации кода на основе грамматики [5]. ANTLR преобразует контекстно-свободную грамматику в виде расширенной формы Бэкуса-Наура (РБНФ) [6] в программу на C++, Java, C#, JavaScript, Go, Swift, Python.

ANTLR по умолчанию считывает грамматику, создает распознаватель для определенного языка и проверяет входной поток на соответствие синтаксису грамматики. В случае отсутствия синтаксических ошибок, ANTLR просто завершает выполнение без вывода каких-либо сообщений. Для более разнообразного использования языка можно присоединять действия к элементам грамматики, которые реализуются на языке программирования, используемом для генерации распознавателя.

ANTLR способен генерировать лексеры, парсеры, парсеры деревьев и комбинированные лексеры-парсеры. Синтаксические анализаторы могут автоматически формировать деревья синтаксического анализа или абстрактные синтаксические деревья, которые могут быть дополнительно обработаны с использованием анализаторов деревьев. При гене-

рации распознавателя действия встраиваются в исходный код распознавателя в соответствующих точках. Например, в случае компилятора действия могут использоваться для построения и проверки таблиц символов, а также для генерации инструкций на целевом языке.

Далее перейдем к описанию разработанного программного модуля. Начнем с выбора алгоритма, который является основой принципа работы модуля.

4. АЛГОРИТМ ЭРЛИ

В книге «Алгоритмы и структуры данных» [7] Никлаус Вирт ставит задачу разработать простой и эффективный алгоритм для синтаксического анализа. Предлагается метод, который называется *нисходящим грамматическим разбором*, заключающийся в реконструировании этапов порождения от стартового символа до конечного предложения, то есть сверху вниз. Однако оказывается, что алгоритм может давать неоднозначный результат, поэтому его применение требует дополнительных ограничений на грамматику.

Далее был рассмотрен *алгоритм Кока-Янгера-Касами*, который также применяется для синтаксического анализа строк в контекстно-свободной грамматике [8]. Он используется для определения, может ли данная строка быть сгенерирована из данной грамматики, и если может, то как.

Принцип работы алгоритма заключается в том, что он строит таблицу для хранения подстрок и устанавливает, могут ли эти подстроки быть сгенерированы из нетерминалов грамматики. Затем он комбинирует эти результаты и итеративно строит более длинные строки до тех пор, пока не достигнет конечной строки, определяя таким образом, может ли вся строка быть получена из грамматики. Однако этот алгоритм ограничен тем, что требует приведения грамматики к *нормальной форме Хомского (НФХ)* [9]. КС-грамматика находится в НФХ, если все ее правила имеют вид:

$$A \to BC \text{ или } A \to \alpha \text{ или } S \to \varepsilon.$$

Соответственно, если бы этот алгоритм использовался для нашего модуля, то было бы необходимо преобразовывать задаваемую грамматику к НФХ. Поэтому был рассмотрен *алгоритм Эрли (англ. Earley algorithm)* — алгоритм синтаксического анализа предложения по контекстно-свободной грамматике, основанный на методе динамического программирования [10]. Его преимущество заключается в том, что он не накладывает ограничений на используемую для анализа контекстно-свободную грамматику.

5. ОПИСАНИЕ УСТРОЙСТВА ПРОГРАММНОГО МОДУЛЯ

Разработанный модуль используется для сравнения грамматики пользователя и эталонной грамматики. Данные грамматики поступают в качестве входных данных, далее происходит генерация контрольных примеров, принадлежащих этим грамматикам.

В задаче мы имеем набор правил.

$$S \to AB,$$
$$A \to a+,$$
$$B \to AB,$$
$$B \to a.$$

Рассмотрим этот этап подробнее (рис. 1).

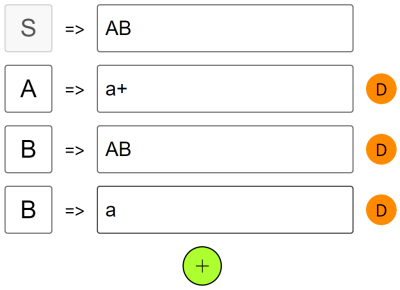

Рис. 1. Пример грамматики

Генерация начинается со стартового нетерминального символа.

Соответственно, из S можно получить только AB. Далее каждый из этих двух нетерминальных символов можно раскрыть с помощью правил, например A как "$a+$", а B как "AB" или как "a". Поэтому количество комбинаций увеличивается, что можно наблюдать на рисунке 2 в виде ветвления.

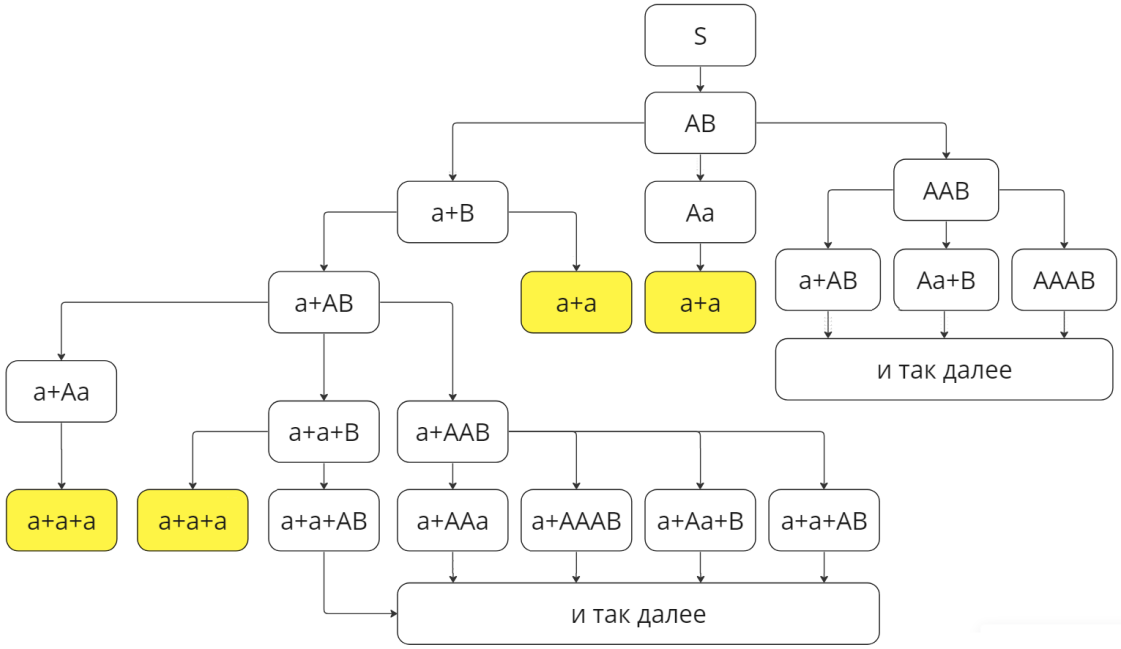

Рис. 2. Построение дерева генерации слов

Из AB можем получить три варианта: $a+B$, Aa, AAB. Далее дерево продолжает строиться в ширину, то есть каждая цепочка терминальных и нетерминальных символов раскрывается согласно правилам по уровням. Когда цепочка состоит только из терминальных символов, она становится словом, которое добавляется в множество. Благодаря этому хранятся только уникальные слова.

Для каждой из двух грамматик генерируется по 2000 тысячи слов, это число было выбрано экспериментальным путем, поскольку его превышение приводит к большим затратам времени. Уменьшать данное значение нет смысла, так как из-за этого понизится точность сравнения.

После генерации происходит сравнение контрольных примеров, принадлежащих грамматике пользователя, с примерами эталонной грамматики (каждый с каждым). Если все сгенерированные примеры совпали, то сравнение заканчивается, а на экран выводится результат сравнения. Если же обнаружено несовпадение, то примеры, которые не нашли себе пару, проверяются другим образом. Это происходит с помощью алгоритма Эрли, который был описан выше. В итоге мы получаем результат сравнения в процентах. Процент принадлежности слов, которые построены по эталонной грамматике, к грамматике пользователя, и наоборот.

Необходимо упомянуть о точности такого способа сравнения. Можно однозначно определять полное совпадение или несовпадение КС-грамматик при помощи данного способа. Однако точность частичного совпадения КС-грамматик не может быть определена окончательно, так как не существует алгоритма для полноценного сравнения КС-грамматик. Процент вхождения слов в ту или иную грамматику зависит от количества сгенерированных примеров. Тем самым точность частичного сравнения может изменяться. Увеличение числа слов грамматики приводит к уменьшению колебаний результата. На рисунке 3 представлен график изменения процента вхождения одной грамматики в другую. Как можно заметить, с увеличением количества генерируемых слов процент вхождения снижается, тем самым повышая точность совпадения грамматик.

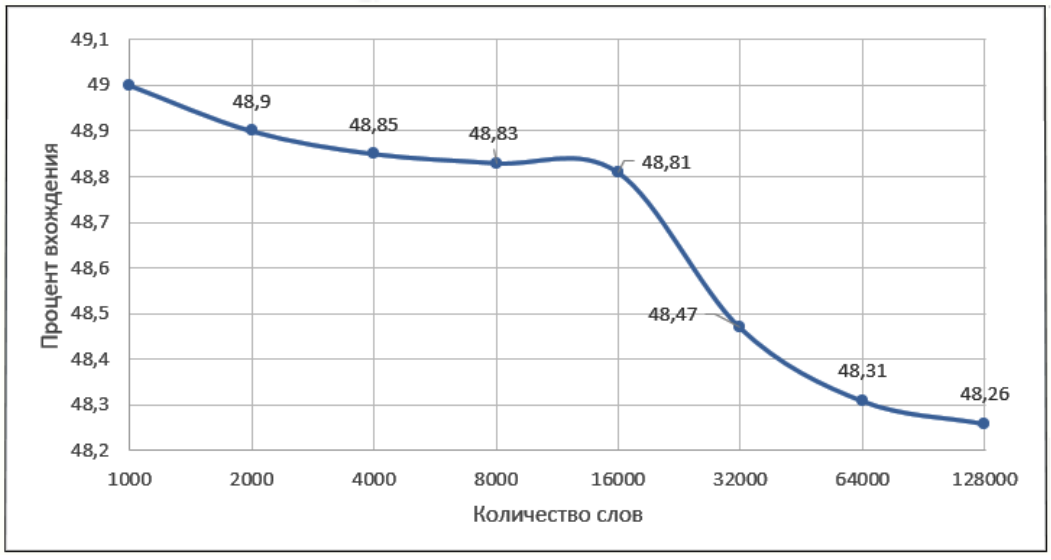

Рис. 3. График изменения процента вхождения

Однако существуют грамматики, набор правил которых приводит к построению менее 2000 тысяч слов. В таком случае можно полностью сравнить все слова грамматик и дать точный результат. Такой пример представлен на (рис. 4–5).

Результат их сравнения является точным и представлен на рис. 6.

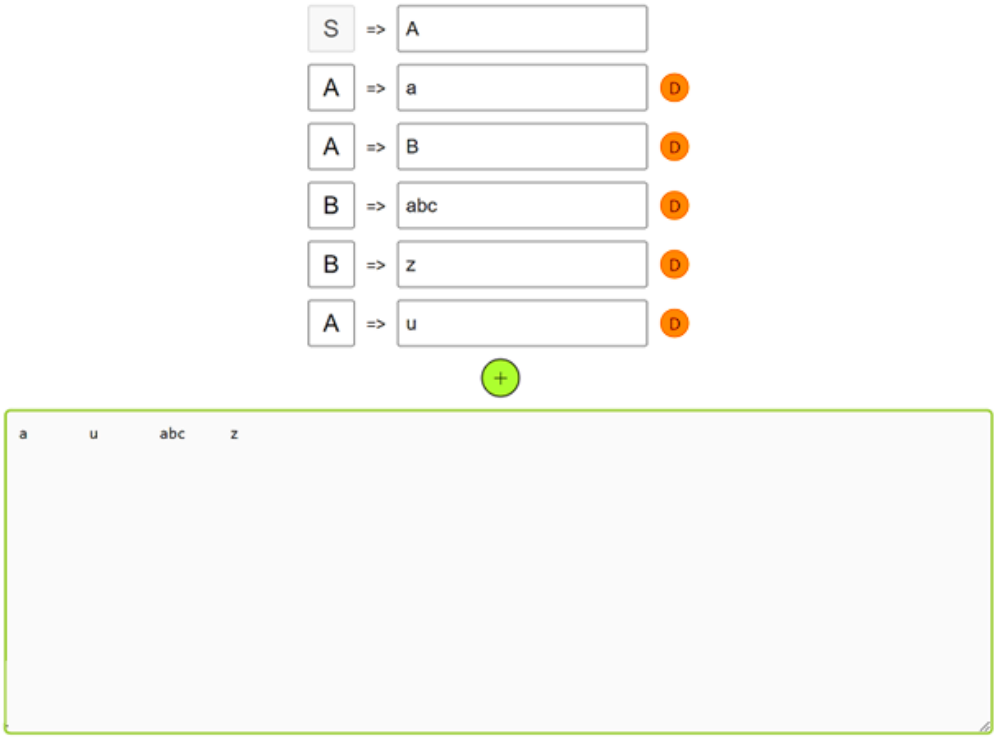

Рис. 4

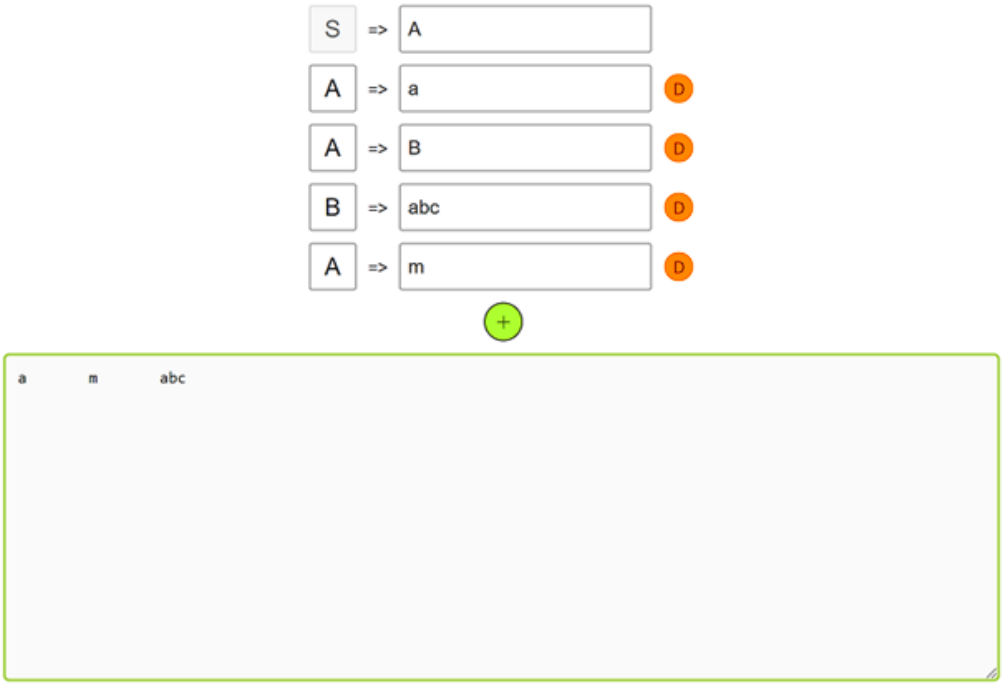

Рис. 5

50|66.67

Рис. 6. Частичное совпадение грамматик (точный результат)

6. РАБОТА С ПРОГРАММНЫМ МОДУЛЕМ

Работа с приложением интуитивно понятна. От пользователей необходимо лишь ввести свои грамматики и нажать кнопку «Сравнить». Стоит заметить, что входной символ всегда "S", изменить его нельзя. Для терминальных и нетерминальных символов можно использовать только английские буквы, цифры и специальные знаки.

Рассмотрим, как в программе происходит отображение описанных выше этапов работы модуля (рис. 7).

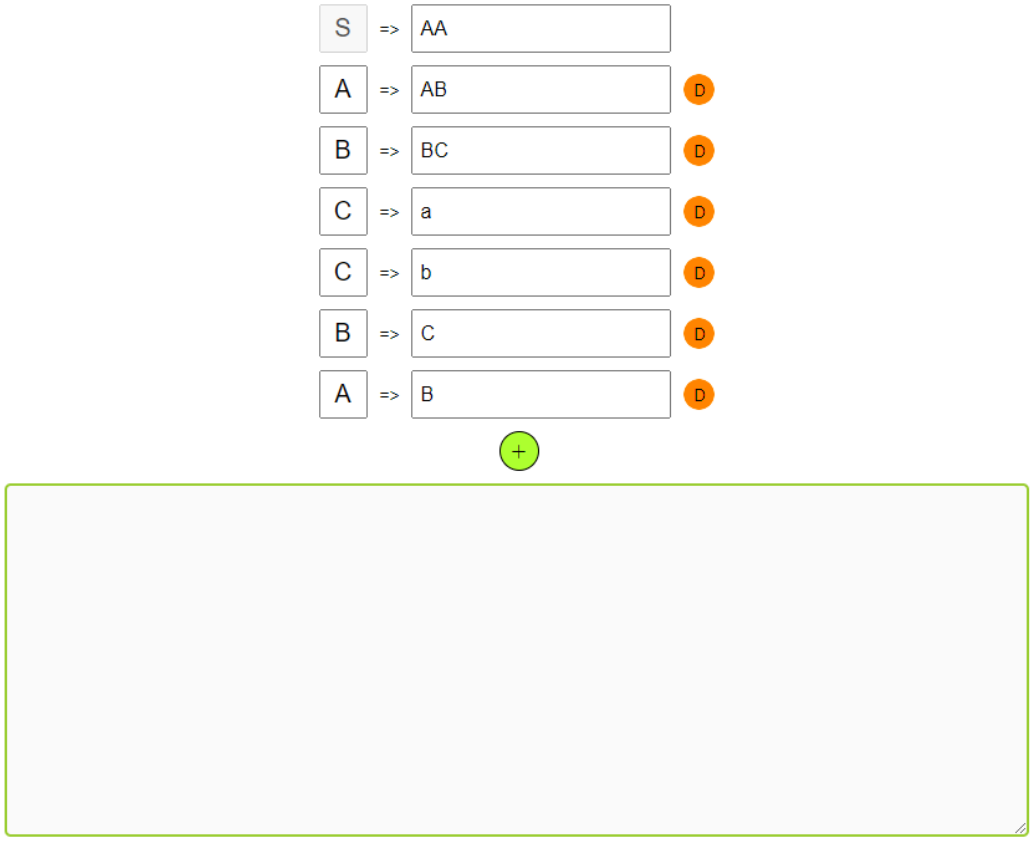

Рис. 7. Форма для задания грамматики

Также есть аналогичная форма для ввода второй грамматики. После задания грамматики происходит их генерация, о чем программа нам сообщает (рис. 8, 9).

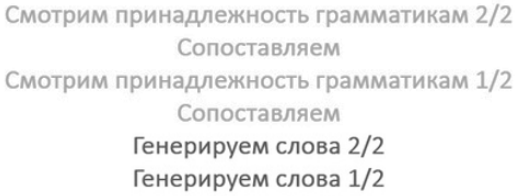

Рис. 8. Генерация слов грамматики

Рис. 9. Сгенерированные слова для грамматики

В итоге можно увидеть процент совпадения грамматик. Исходя из полученных результатов, понимаем, что гипотеза о совпадении грамматик не опровергнута, и можно предположить, что грамматики совпадают (рис. 10).

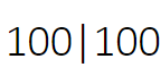

Рис. 10. Совпадение грамматик

Если ввести несовпадающие грамматики, то полученный результат будем другим (рис. 11).

Рис. 11. Несовпадение грамматик

Также совпадение грамматик может быть частичным (рис. 12).

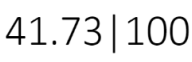

Рис. 12. Частичное совпадение грамматик

Кроме того, в приложении реализована возможность вводить произвольные примеры, чтобы проверить — могут ли они быть порождены эталонной грамматикой и грамматикой пользователя (рис. 13).

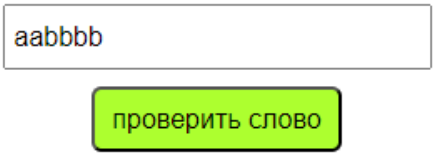

Рис. 13. Проверка слова на принадлежность грамматике

Результат проверки на принадлежность может быть разным, в зависимости от заданных грамматик и самого слова (рис. 14–17).

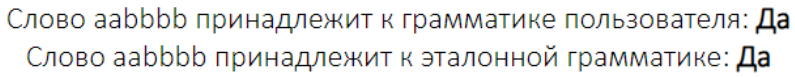

Рис. 14. Проверка слова на принадлежность грамматике

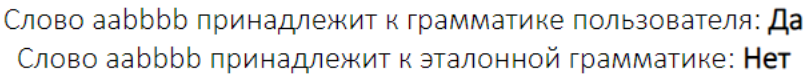

Рис. 15. Проверка слова на принадлежность грамматике

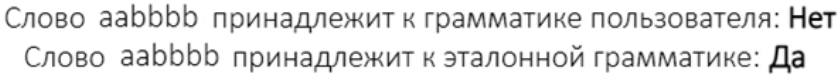

Рис. 16. Проверка слова на принадлежность грамматике

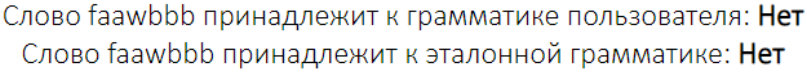

Рис. 17. Проверка слова на принадлежность грамматике

7. ЗАКЛЮЧЕНИЕ

Результатом работы является разработанный программный модуль, который представляет собой инструмент для изучения контекстно-свободных грамматик и работы с ними. Он позволяет пользователям проводить эксперименты, создавать и анализировать контекстно-свободные грамматики, что способствует овладению принципами работы с грамматиками и их применением.

В ходе работы были рассмотрены различные алгоритмы для обеспечения сравнения грамматик в реальном времени. Модуль может быть применен в Олимпиаде по дискретной математике и теоретической информатике (ДМиТИ), которая базируется на конструировании таких объектов как переключательные схемы, графы, алгоритмы, автоматы,

регулярные выражения. Включение разработанного модуля в систему поддержки Олимпиады ДмиТИ позволит добавить ещё один важный класс объектов — грамматики, на которых можно ставить конструктивно-исследовательские задачи.

Список литературы

1. *Хомский Н.* Синтаксические структуры // Новое в лингвистике. М., 1962. Вып. II. С. 412–527.
2. *Мартыненко Б.К.* Языки и трансляции: Учеб. пособие. Изд. 2-е, испр. и доп. СПб.: Изд-во С.-Петерб. ун-та, 2013. 265 с.
3. Jetbrains MPS [Электронный ресурс] URL: https://www.jetbrains.com/mps/ (дата обращения: 01.04.2024).
4. Xtext Eclipse [Электронный ресурс] URL: https://eclipse.dev/Xtext/ (дата обращения: 01.04.2024).
5. ANTLR. [Электронный ресурс]. URL: https://www.antlr.org/ (дата обращения: 01.04.2024).
6. Расширенная форма Бэкуса Наура [Электронный ресурс] URL: https://ru.wikipedia.org/w/index.php? title=Расширенная_форма_Бэкуса_—_Наура&stable=1 (дата обращения: 01.04.2024).
7. *Вирт Н.* Алгоритмы и структуры данных. М.: ДМК Пресс, 2010. 272 с.
8. Алгоритм Кока Янгера Касами [Электронный ресурс] URL: https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Кока-Янгера-Касами_разбора_грамматики_в_НФХ (дата обращения: 01.04.2024).
9. Нормальная форма Хомского [Электронный ресурс] URL: https://neerc.ifmo.ru/wiki/index.php?title=Нормальная_форма_Хомского (дата обращения: 01.04.2024).
10. *Сорокин А.* Алгоритм Эрли [Электронный ресурс] URL: https://homepage.mi-ras.ru/ sk/lehre/fivt2013/Earley.pdf (дата обращения: 01.04.2024).

Поступила в редакцию 02.05.2024, окончательный вариант — 27.06.2024.

Жегалин Михаил Александрович, студент 4 курса кафедры вычислительной техники СПбГЭТУ «ЛЭТИ», ✉ **michael.zhegalin.univ@gmail.com**

Яндринский Владислав Вадимович, студент 4 курса кафедры вычислительной техники СПбГЭТУ «ЛЭТИ», **yandrinsky@gmail.com**

Computer tools in education, 2024

№ 2: 72–84

http://cte.eltech.ru

doi:10.32603/2071-2340-2024-2-72-84

A Software Module for Working with Context-Free Grammars

Zhegalin M. A.[1], Student, ✉ michael.zhegalin.univ@gmail.com
Yandrinski V. V.[1], Student, yandrinsky@gmail.com

[1]Saint Petersburg Electrotechnical University, 5, st. Professora Popova, 197022, Saint Petersburg, Russia

Abstract

The article presents the development of an educational software module to support the initial stage of learning formal grammars. The developed module allows the student or teacher to describe a given language through context-free grammar. In the process of

completing a grammar assignment, the module provides an interactive response to the differences between the constructed grammar and the reference one. The main technical problem that the authors solved is related to the fact that there is no algorithm for determining the equivalence of two context-free grammars. Therefore, the module must generate in real time a sufficiently large number of words of a language of different lengths in order to detect differences between the proposed grammar and the reference one with a high probability. In the work, several word generation algorithms were analyzed, after which the algorithm most appropriate to the conditions of use of the module was selected. The developed module can be used for conducting Olympiads in discrete mathematics and computer science for schoolchildren and students, as well as to support students' independent work on the topic "Formal languages and grammars".

Keywords: *learning module, grammars, Earley's algorithm, context-free grammars, grammar comparison.*

Citation: M. A. Zhegalin and V. V. Yandrinski, "A Software Module for Working with Context-Free Grammars," *Computer tools in education*, no. 2, pp. 72–84, 2024 (in Russian); doi:10.32603/2071-2340-2024-2-72-84

References

1. N. Chomsky, "Syntactic Structures," *New in linguistics*, vol. 2, pp. 412–527, 1962 (in Russian).
2. B. K. Martynenko, *Languages and translations*, St. Petersburg, Russia: SPbGU Publ., 2013 (in Russian).
3. Jetbrains MPS, "Meta Programming System," in *www.jetbrains.com*, 2024. [Online]. Available: https://www.jetbrains.com/mps/
4. Xtext Eclipse, "Language Engineering For Everyone!," in *eclipse.dev*, 2024. [Online]. Available: https://eclipse.dev/Xtext/
5. ANTLR, "Quick Start," in *www.antlr.org*, 2024. [Online]. Available: https://www.antlr.org/
6. "Extended Backus–Naur Form," in *ru.wikipedia.org*, 2024 (in Russian). [Online]. Available: https://ru.wikipedia.org/w/index.php?title=Расширенная_форма_Бэкуса_—_Наура&stable=1
7. N. Wirth, *Algorithms and Data Structures*, Moscow: DMK press, 2010 (in Russian).
8. ITMO University, "Cocke-Younger-Kasami algorithm," in *neerc.ifmo.ru*, 2022 (in Russian). [Online]. Available: https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Кока-Янгера-Касами_разбора_грамматики_в_НФХ
9. ITMO University, "Chomsky normal form," in *neerc.ifmo.ru*, 2022 (in Russian). [Online]. Available: https://neerc.ifmo.ru/wiki/index.php?title=Нормальная_форма_Хомского
10. A. Sorokin, *Earley algorithm*, in *homepage.mi-ras.ru*, 2013 (in Russian). [Online]. Available: https://homepage.mi-ras.ru/~sk/lehre/fivt2013/Earley.pdf

Received 02-05-2024, the final version — 27-06-2024.

Mikhail Zhegalin, 4th year Student of the bachelor's degree program of the Computer Science Department, Saint Petersburg Electrotechnical University, ✉ michael.zhegalin.univ@gmail.com

Vladislav Yandrinski, 4th year Student of the bachelor's degree program of the Computer Science Department, Saint Petersburg Electrotechnical University, yandrinsky@gmail.com