Картина, която съдържа текст, графична колекция

Описанието е генерирано автоматичноТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА

Факултет по изчислителна техника и автоматизация

Катедра „Софтуерни и интернет технологии“

**КУРСОВ ПРОЕКТ**

по дисциплината „Обектно-ориентирано програмиране 1 част”

на тема: „Контекстно-свободна граматика ”

Вариант 5

|  |  |
| --- | --- |
| Изготвил:  Павел Георгиев Георгиев | Проверил:  ас. В. Колесниченко |
| Специалност: СИТ |  |
| Група: 1а |  |
| Факултетен номер: 22621621 |  |

2024

Съдържание

[Глава 1. Увод 4](#_Toc170054187)

[1.1 Описание и идея на проекта 4](#_Toc170054188)

[1.2 Цел и задачи на разработката 5](#_Toc170054189)

[Глава 2. Преглед на предметната област 5](#_Toc170054190)

[2.1 Основни дефиниции, концепции и алгоритми, които ще бъдат използвани 5](#_Toc170054191)

[2.2 Дефиниране на проблеми и сложност на поставената задача 6](#_Toc170054192)

[2.3 Подходи, методи за решаване на поставените проблемите 7](#_Toc170054193)

[2.4 Потребителски изисквания и качествени изисквания 9](#_Toc170054194)

[Глава 3. Проектиране 9](#_Toc170054195)

[3.1 Обща структура на проекта пакети който ще се реализират 9](#_Toc170054196)

[3.2 Диаграми/Блок схеми(на структура и поведение -по обекти, слоеве с най-важните извадки от кода) 10](#_Toc170054197)

[Глава 4. Реализация, тестване 12](#_Toc170054198)

[4.1 Реализация на класове 12](#_Toc170054199)

[4.2 Алгоритми и Оптимизации. 13](#_Toc170054200)

[4.3 Планиране, описание и създаване на тестови сценарии (създаване на примери) 14](#_Toc170054201)

[Глава 5. Заключение 22](#_Toc170054202)

[5.1 Обобщение на изпълнението на началните цели 22](#_Toc170054203)

[5.2 Насоки за бъдещо развитие и усъвършенстване 23](#_Toc170054204)

[Използвана литература 23](#_Toc170054205)

Глава 1. Увод

* 1. Описание и идея на проекта

Да се реализира програма, която поддържа операции с контекстно-свободна граматика, използваща главни латински букви за променливи (нетерминали) и малки латински букви и цифри за терминали.

Граматиките да се сериализират по разработен от Вас формат. Всяка прочетена граматика да получава уникален идентификатор.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (open, close, save, save as, help и exit):

|  |  |
| --- | --- |
| list | Списък с идентификаторите на всички прочетени граматики |
| print <id> | Извежда граматиката в подходящ формат. За всяко правило да се отпечата пореден номер |
| save <id> <filename> | Записва граматиката във файл |
| addRule <id> <rule> | Добавя правила |
| removeRule <id> <n> | Премахване на правило по пореден номер |
| union <id1> <id2> | Намира обединението на две граматики и създава нова граматика. Отпечатва идентификатора на новата граматика |
| concat <id1> <id2> | Намира конкатенацията на две граматики и създава нова граматика. Отпечатва идентификатора на новата граматика |
| chomsky <id> | Проверява дали дадена граматика е в нормална форма на Чомски |
| cyk <id> | Проверява дали дадена дума е в езика на дадена граматика (CYK алгоритъм) |
| iter <id> | Намира резултат от изпълнението на операцията “итерация” (звезда на Клини) над граматика и създава нова граматика. Отпечатва идентификатора на новата граматика |
| empty <id> | Проверява дали езикът на дадена контекстно-свободна граматика е празен |
| chomskify <id> | Преобразува граматика в нормална форма на Чомски. Отпечатва идентификатора на новата граматика |

* 1. Цел и задачи на разработката

Целта на разработката е да се създаде програма, която може да обработва потребителския вход чрез работа с командния ред(*Command line interface*). За целта, когато програмата се стартира, тя трябва да позволява на потребителя да въвежда команди и след това да ги изпълнява. Командата е ключова дума, която се намира на първо място в потребителския вход. Програмата трябва да позволява на потребителя да извършва операции със файлове, като: отваряне на файлове (open), записване на промените обратно в същия файл (save) или в друг, който потребителят посочи (save as). Трябва да има и опция за затваряне на файла, без записване на промените (close). Когато отворите даден файл, неговото съдържание трябва да се зареди в паметта, след което файлът се затваря. Всички промени, които потребителят направи след това трябва да се пазят в паметта, но не трябва да се записват обратно, освен ако потребителят изрично не укаже това.

Програмата трябва да поддържа и операции върху граматики, като добавяне и премахване на правила, изпълнение на операции като конкатенация, обединение и също така някои по-сложни операции като преобразуване в нормална форма на Чомски и CYK алгоритъм.

Глава 2. Преглед на предметната област

2.1 Основни дефиниции, концепции и алгоритми, които ще бъдат използвани

**Контекстно свободна граматика (КСГ)** е формална граматика, чиито правила могат да се прилагат към нетерминален символ независимо от неговия контекст. Единичният нетерминал от лявата страна винаги може да бъде заменен с терминал от дясната страна. КСГ е начин за описване на езици чрез рекурсивни правила, наречени продукции. Една КСГ се определя от следните четири компонента: набор от променливи V, набор от терминални символи T, начална променлива S и набор от продукциите P [[12]](#ref12) , [[14]](#ref14) , [[21]](#ref21) .

1.Наборът от терминали е краен набор от символи, които образуват низовете на дефинирания език [[5]](#ref5) .

2. Наборът от променливи е краен набор от променливи, като всяка променлива представлява език – набор от низове [[5]](#ref5) .

3. Начален символ - един от нетерминалите представлява езика, който се дефинира. Другите променливи представляват спомагателни класове от низове, които се използват, за да помогнат да се определи езикът на началния символ.

4.Набор от продукции - представят рекурсивното дефиниране на даден език.

**Правило(продукция)** – всяко правило се състои от:

- нетерминал, който се определя от правилото. Често се нарича глава на продукцията;

-символ на продукцията - →;

- редица от нула или повече терминали и/или нетерминали. Нарича се тяло и представлява начин за образуване на низове в езика на граматиката [[3]](#ref3) , [[14]](#ref14) .

**Звезда на Клини** - математически процес или операция върху набор от числа, знаци или символи. Ако вземем набор от числа V и създадем безкраен списък с всички възможни комбинации от числата, тогава сме създали звездата на Клини на V. При дадено множество звездата на Клини представлява множеството от всички низови комбинации на членовете на това множество [[4]](#ref4) .

**CYK алгоритъм** – алгоритъм за разбор на КСГ, използващ динамично програмиране. Проверява дали дадена дума е част от езика на граматиката. Стандартната версия на CYK работи само с КСГ, които са в нормална форма на Чомски(CNF) [[15]](#ref15) .

**Конкатенация** - операцията конкатенация взима две думи α и β и образува новата дума α · β като слепва двете думи. Например aba · bb = ababb [[2]](#ref2) , [[12]](#ref12) .

**Нормална форма на Чомски(CNF)** – една КСГ е в нормална форма на Чомски, ако всяко правило е от вида A → BC и A → a, като не могат да бъдат променливата за начало S. Освен това, позволяваме правилото S → ε [[7]](#ref7) , [[12]](#ref12) .

2.2 Дефиниране на проблеми и сложност на поставената задача

Основните проблеми при разработването на програмата са следните:

1. Имплементация на *Command line interface*;
2. Работа със файлове;
3. Представяне на граматика в програмата;
4. Сериализация на граматиките;
5. Имплементация на операциите върху граматики;
6. Обработване на грешки.

За да се имплементира *Command line interface* е нужно да се осигури обработка на потребителския вход, разпознаване на командата, която потребителя извиква и също така извикване на самата команда. Всички останали команди могат да се изпълняват само ако има успешно зареден файл.

За да може потребителя да работи със файлове трябва да се създадат команди за всяка една операция върху файлове и да се пази в паметта нужна информация, която осигурява гладката работа със файлове – дали е отворен файл, какъв е неговият път и какво е съдържанието му. Когато се отвори даден файл, неговото съдържание трябва да се зареди в паметта, след което файлът се затваря. Всички промени, които потребителят направи след това трябва да се пазят в паметта, но не трябва да се записват обратно, освен ако потребителят изрично не укаже това.

Програмата не може да работи с граматики без да съществува дефиниран обект, който представлява една граматика и нейните основни части – правилата. Всяка граматика също трябва да има уникален идентификатор, който да различава граматиките една от друга.

След създаване на обекта граматика трябва да се имплементират команди за всички операции върху граматики, като имплементацията на различните операции силно варира по сложност – докато операциите за добавяне и премахване на правило, извеждане на една граматика и показването на всички записани граматики са сравнително елементарни за имплементация, операциите за преобразуване на граматика в нормална форма на Чомски и CYK алгоритъма за проверка принадлежността на дума към език на граматика са логически сложни и изискват сравнително голямо количество ресурси за успешната им реализация.

По време на работа на програмата е вероятно да възникнат грешки поради въвеждане на невалидни параметри на командите, и тези грешки трябва да се обработят по подходящ начин и да се извести потребителя за настъпилият проблем.

2.3 Подходи, методи за решаване на поставените проблемите

1. Имплементация на Command line interface;

За успешното имплементиране на Command line interface е необходим интерфейс Command*,* който дефинира основното поведение на една команда – метод за изпълнение. Също така е необходим клас имплементация за всяка команда, който съдържа програмната реализация на съответната команда или извиква метод от някой друг клас, където се съдържа тази реализация.

Освен това трябва да присъстват и класове, които извършват следните действия: съхранение на всички поддържани команди, приемане и обработка на потребителският вход. В моята програмна реализация тези действия се извършват в един клас – CommandHandler. Той приема потребителския вход, разделя го на части, търси ключова дума на команда и при съвпадение извиква съответната команда.

За съхранение на всичките команди съм избрал структурата HashMap, защото съхранява елементи като комбинация от ключ и стойност – в случая ключа е думата с която се извиква съответната команда, а стойността е клас на командата.

Приемането на потребителски вход съм осъществил чрез класа Scanner*,* който позволява лесна обработка на вход и разделяне на информацията на токени. При прочитане на ред от текст разделям реда на части и използвам първата част за извикване на команда.

2. Работа със файлове;

За успешната имлементация на операциите със файлове е нужен клас, който съхранява нужната информация. В моят проект това е класа FileHandlerImpl, който съхранява състояние за отворен файл, път на текущо отворения файл и неговото съдържание. Този клас също съдържа getter и setter за файловото съдържание, тъй като почти всички команди взаимодействат с него. Също така има предвиден набор от тип граматика, който съдържа всички прочетени граматики от програмата.

3. Представяне на граматика в програмата;

За представяне на граматика в програмата е нужен клас за граматика, който дефинира основното поведение на една граматика. В моят проект това е класа Grammar. Класа граматика съдържа набор от правила, тъй като правилата са един от основните елементи на една граматика. Друго поле на класа граматика е идентификатор, който показва уникалността на всяка граматика. Също така съм включил и текстово представяне на една граматика, тъй като се изисква многократно в програмата.

Но преди създаване на този клас трябва да се направи клас, който представлява едно правило. Класа правило трябва да съдържа двете части на едно граматическо правило – нетерминал и терминали. В моят проект това е класа Rule. За полета на класа правило съм избрал нетерминала да е низ, а терминалите да са набор от низове. Освен това съм включил и текстово представяне на едно правило, тъй като то се употребява на множество места в програмата.

4. Сериализация на граматиките;

Сериализацията на граматиките съм осъществил чрез метод в класа граматика, който генерира уникално число. То е от тип long, за да може генерираното число да е възможно най-произволно.

5. Имплементация на операциите върху граматики;

Проблем при имплементацията на операциите върху граматики е промяната на обекта граматика по време на обхождане на самия обект. При опит за промяна се хвърля грешка ConcurrentModificationException*,* която се получава при модифициране на колекция от обект, различен от итератора.

За избягването на тази грешка аз съхранявам в паметта частите от терминалите на даденото правило, които трябва да се променят, и след като приключи обхождането на терминалите на правилото осъществявам нужните промени. Ако се наложи промяна на цели правила това се осъществява след приключването на обхождането на всички правила.

6. Обработване на грешки.

Ако потребителят не е добре запознат с документацията на програмата и/или с теорията на граматиките е неизбежно това да бъдат въведени невалидни параметри на команда. За това е нужно създаването на специфични грешки които се случват при работата с файлове и граматики. Всички допълнително създадени класове за грешки включват параметър съобщение, защото потребителят трябва да бъде известен за тяхното настъпване и да бъде информиран за това как да избегне съответната грешка.

Създал съм следните класове:

* клас за грешка при липсата на отворен файл, тъй като повечето команди изискват отворен файл за да работят.
* класове за грешка при въвеждане на невалидни данни, тъй като това със сигурност ще се случи по време на работа на програмата.
* клас за липса на съвпадение на даден текст във файловото съдържание, тъй като някои команди изискват намирането и промяната на част от файловото съдържание.

2.4 Потребителски изисквания и качествени изисквания

За да се изпълни програмата е нужно потребителя да има инсталирана версия 21.0.2 на JAVA SE. Необходимо е и потребителският акаунт да има право да създава и редактира файлове.

Глава 3. Проектиране

3.1 Обща структура на проекта пакети който ще се реализират

Общата структура на проекта е разделена в следните пакети:

* bg.tu\_varna.sit.a1.f22621621 – основен пакет на проекта. Съдържа всички останали пакети и главният клас на програмата – Application
* commands – пакет с команди. Съдържа всички команди, които програмата поддържа.
* exceptions – пакет с грешки. Съдържа всички класове на грешки, дефинирани от програмиста.
* interfaces – пакет с интерфейси. Съдържа всички интерфейси, които са дефинирани
* models – пакет с имплементации. Съдържа имплементациите на повечето интерфейси
* utils – пакет с помощни класове. Съдържа класове, които подпомагат реализацията на програмата

3.2 Диаграми/Блок схеми(на структура и поведение -по обекти, слоеве с най-важните извадки от кода) Картина, която съдържа текст, екранна снимка, софтуер, Мултимедиен софтуер

Описанието е генерирано автоматично

Фигура 1. Interfaces

Картина, която съдържа текст, екранна снимка, диаграма

Описанието е генерирано автоматично

Фигура 2. Models

Картина, която съдържа текст, екранна снимка, софтуер, Мултимедиен софтуер

Описанието е генерирано автоматично

Фигура 3. Utils

Картина, която съдържа текст, екранна снимка, Мултимедиен софтуер, софтуер

Описанието е генерирано автоматично

Фигура 4. Commands

Картина, която съдържа текст, екранна снимка, Мултимедиен софтуер, Графичен софтуер

Описанието е генерирано автоматично

Фигура 5. Exceptions

Глава 4. Реализация, тестване

4.1 Реализация на класове

Реализацията на класовете е описана чрез JavaDocs система за генериране на документация от коментари в изходния код. В реализацията на проекта са използвани помошни методи и структури от интернет. Препратки:

Структура Triple [[1]](#ref1) , честота на подниз в низ [[6]](#ref6) , намиране на всички комбинации – *power set* [[16]](#ref16) , общи елементи на 2 набора [[18]](#ref18) , *Command line interface* [[19]](#ref19) , n-та на брой поява на подниз [[20]](#ref20) .

Линк към проекта в GitHub: <https://github.com/pavelg122/sitoop1project22621621>

4.2 Алгоритми и Оптимизации.

Алгоритъм за преобразуване на граматика в нормална форма на Чомски [[8]](#ref8) , [[13]](#ref13) :

Този алгоритъм се състои от следните 5 трансформации:

1. START – премахване на началния символ от терминалите на правилата. Това се постига чрез добавяне на ново правило в началото на граматиката и промяна на началния символ, ако началният символ се среща в някой терминал на правило
2. TERM – премахване на *term* правила. Едно *term* правило има вида: A → X1...a...Xn. Този тип правила се премахва чрез първо откриване на всички правила от този тип и за всеки терминален символ се създава ново правило с терминал този символ. След това правилото, което съдържа този терминален символ се променя по следния начин: терминалния символ се заменя с нетерминалния символ на новото правило.
3. BIN – премахване на *bin* правила. Едно *bin* правило има вида: A → X1X2...Xn. Този вид правило се премахва чрез намирането на правила, които имат повече от два нетерминални символа от дясната страна(в терминала). Създава се ново правило, чиито терминал са първите два нетерминални символа на bin правилото. След това те се заменят с лявата страна(нетерминала) на новосъздаденото правило. Това продължава докато се елиминират всички правила, които имат повече от два нетерминални символа от дясната страна.
4. DEL – премахване на празни правила. Едно празно правило има вида: A → ε. Този вид правила се премахва от дадено правило като се намерят всички терминали на правилото, които съдържат нетерминала на празното правило и след това се добави целият набор от комбинации, в които някоя инстанция на нетерминалният символ е премахната.
5. UNIT – премахване на *unit* правила. Едно *unit* правило има вида: A → B. Този вид правила се премахва като символа от дясната страна се замени с терминалите на правило, което има този символ за лява страна. В примера A → B: символът B ще бъде заменен от терминалите на правилото, което има B от лявата страна( B → xx). Правилото A → B ще стане A → xx .

За реализацията на този алгоритъм реда на трансформации е от голямо значение, тъй като някои трансформации могат да унищожат резултата от други. Реда на трансформации също оказва влияние върху размера на граматиката по време на преобразуване. Подредбите START,TERM,BIN,DEL,UNIT и START,BIN,DEL,UNIT,TERM водят до най-малко увеличение на размера, но аз избрах поредността START,DEL,UNIT,TERM,BIN, тъй като бях намерил постъпково преобразование на граматика в този ред.

CYK алгоритъм за откриване на принадлежност на дума към език на граматика [[15]](#ref15), [[17]](#ref17) :

Този алгоритъм разглежда всички възможни поднизове на входния низ и задава P[l,s,v] да е истина, ако поднизът с дължина l, започващ от s, може да бъде резултат от правило с нетерминал Rv . След като се разгледат поднизове с дължина 1 се разглеждат поднизове с дължина 2 и т.н. За поднизове с дължина 2 и повече се разглежда всяко възможно разделяне на подниза на две части и проверява дали има някакво правило A → BC такова, че B съответства на първата част и C съответства на втората част. Ако има такова правило, то А се записва като съвпадащ с целия подниз. След като този процес приключи, входният низ се генерира от езика на граматиката, ако поднизът, съдържащ целия входен низ съвпада със началният символ.

4.3 Планиране, описание и създаване на тестови сценарии (създаване на примери)

въвеждане на невалидна команда:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

open – зарежда съдържанието на даден файл

успешно изпълнение:



изпълнение на командата при вече отворен файл:



невалиден брой аргументи:



close – затваря текущо отворения документ

успешно изпълнение:



невалиден брой аргументи:



не е отворен файл:



save – записва направените промени обратно в същия файл, от който са били прочетени данните

успешно изпълнение:



друг вариант на командата save със параметри,където потребителя оказва коя граматика да се запише във файл

успешно изпълнение:



не е намерена граматика:



невалиден брой аргументи:



не е отворен файл:



saveas – записва направените промени във файл, като позволява на потребителя да окаже неговия път

успешно изпълнение:



невалиден брой аргументи:



не е отворен файл:



help – извежда кратка информация за поддържаните от програмата команди

успешно изпълнение:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

exit – излиза от програмата

успешно изпълнение:

Картина, която съдържа текст, Шрифт, екранна снимка

Описанието е генерирано автоматично

list - Списък с идентификаторите на всички прочетени граматики

успешно изпълнение:

Картина, която съдържа текст, Шрифт, екранна снимка, дизайн

Описанието е генерирано автоматично

невалиден брой аргументи:



не е отворен файл:



print - Извежда граматиката в подходящ формат. За всяко правило да се отпечата пореден номер

успешно изпълнение:

Картина, която съдържа текст, екранна снимка, Шрифт, дизайн

Описанието е генерирано автоматично

невалиден брой аргументи:



не е отворен файл:



не е намерена граматика:



addRule - Добавя правила

успешно изпълнение:

Картина, която съдържа текст, Шрифт, екранна снимка

Описанието е генерирано автоматично

невалиден брой аргументи:



не е отворен файл:



не е намерена граматика:



removeRule - Премахване на правило по пореден номер

успешно изпълнение:

Картина, която съдържа текст, Шрифт, екранна снимка, дизайн

Описанието е генерирано автоматично

невалиден брой аргументи:



не е отворен файл:



не е намерена граматика:



union - Намира обединението на две граматики и създава нова граматика. Отпечатва идентификатора на новата граматика

успешно изпълнение:



невалиден брой аргументи:



не е отворен файл:



не е намерена граматика:

Картина, която съдържа текст, Шрифт, екранна снимка, типография

Описанието е генерирано автоматично

concat - Намира конкатенацията на две граматики и създава нова граматика. Отпечатва идентификатора на новата граматика

успешно изпълнение:



невалиден брой аргументи:



не е отворен файл:



не е намерена граматика:

Картина, която съдържа текст, Шрифт, екранна снимка, типография

Описанието е генерирано автоматично

chomsky - Проверява дали дадена граматика е в нормална форма на Чомски

граматиката е в нормална форма на Чомски:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

граматиката не е в нормална форма на Чомски:



невалиден брой аргументи:



не е отворен файл:



не е намерена граматика:



cyk - Проверява дали дадена дума е в езика на дадена граматика (CYK алгоритъм)

думата е част от езика:



думата не е част от езика:



граматиката не е в нормална форма на Чомски:



невалиден брой аргументи:



не е отворен файл:



не е намерена граматика:



iter - Намира резултат от изпълнението на операцията “итерация” (звезда на Клини) над граматика и създава нова граматика. Отпечатва идентификатора на новата граматика

успешно изпълнение:



невалиден брой аргументи:



не е отворен файл:



не е намерена граматика:



empty - Проверява дали езикът на дадена контекстно-свободна граматика е празен

граматиката е с празен език:



граматиката не е с празен език:



невалиден брой аргументи:



не е отворен файл:



не е намерена граматика:



chomskify - Преобразува граматика в нормална форма на Чомски. Отпечатва идентификатора на новата граматика

успешно изпълнение:

Картина, която съдържа текст, екранна снимка, Шрифт, дизайн

Описанието е генерирано автоматично Картина, която съдържа текст, екранна снимка, дизайн

Описанието е генерирано автоматично

невалиден брой аргументи:



не е отворен файл:



не е намерена граматика:



граматиката е вече в нормална форма на Чомски:



Глава 5. Заключение

* 1. Обобщение на изпълнението на началните цели

Началните цели са успешно изпълнени, като те са:

* Обработване на потребителския вход чрез работа с командния ред(Command line interface);
* Имплементиране на команди за работа с файлове;
* Имплементиране на команди за работа с граматики.

Програмата позволява на потребителя да избира какви операции да се извършват като използва ключови думи за извикване на съответната команда, съдържаща програмната имплементация на операцията. Програмата поддържа команди за работа с файлове и команди за операции върху граматики.

5.2 Насоки за бъдещо развитие и усъвършенстване

За бъдещо развитие е възможно да се добавят и останалите основни елементи на една граматика за пълна реализация на обекта граматика – начален символ, набор от терминали и набор от нетерминали.

Друго възможно подобрение е създаване на интерфейси за класовете Grammar, Rule, CommandHandler, GrammarUtils за пълно спазване на ООП принципа абстракция. Препоръчително е и прилагането на различни дизайнерски модели за допълнително оптимизиране на работата на програмата.

Допълнителна добра практика ще бъде пълното разделяне на програмната имплементация на командите и съответният клас наследник на интерфейса Command, за да се спази напълно поведенческият шаблон Команда.

Използвана литература

[1] Storing Data Triple in a List in Java. 2024. <https://www.baeldung.com/java-list-storing-triple>

[2] Concatenation – Wikipedia. 2024. <https://en.wikipedia.org/wiki/Concatenation>

[3] Production (computer science) – Wikipedia. 2024. <https://en.wikipedia.org/wiki/Production_(computer_science)>

[4] Kleene star – Simple English Wikipedia, the free encyclopedia. 2024. <https://simple.wikipedia.org/wiki/Kleene_star>

[5] Terminal and nonterminal symbols – Wikipedia. 2024. <https://en.wikipedia.org/wiki/Terminal_and_nonterminal_symbols>

[6] Frequency of a Substring in a String – GeeksforGeeks. 2023. <https://www.geeksforgeeks.org/frequency-substring-string/>

[7] Chomsky normal form – Wikipedia. 2023. <https://en.wikipedia.org/wiki/Chomsky_normal_form>

[8] Simplifying Context Free Grammars – GeeksforGeeks. 2022. <https://www.geeksforgeeks.org/simplifying-context-free-grammars/?ref=lbp>

[9] Explain the context free language closure under union operation?. 2021. <https://www.tutorialspoint.com/explain-the-context-free-language-closure-under-union-operation>

[10] Explain the context free language closure under concatenation?. 2021. <https://www.tutorialspoint.com/explain-the-context-free-language-closure-under-concatenation>

[11] Explain the Closure Under Kleene Star of CFL in TOC?.2021. [https://www.tutorialspoint.com/explain-the-closure-under-kleene-star-of-cfl-in-toc]( https://www.tutorialspoint.com/explain-the-closure-under-kleene-star-of-cfl-in-toc)

[12] Вътев, Стефан. Езици, автомати, изчислимост. 2015. <https://store.fmi.uni-sofia.bg/fmi/logic/static/eai/eai.pdf>

[13] Converting Context Free Grammar to Chomsky Normal Form – GeeksforGeeks. 2019. <https://www.geeksforgeeks.org/converting-context-free-grammar-chomsky-normal-form/?ref=lbp>

[14] Hopcroft J., Motwani R., Ullman J., Introduction to Automata Theory, Languages, and Computation(second edition). 2001. <https://www-2.dc.uba.ar/staff/becher/Hopcroft-Motwani-Ullman-2001.pdf>

[15] CYK algorithm – Wikipedia. 2024. <https://en.wikipedia.org/wiki/CYK_algorithm>

[16]Software Everything: Shortcut to Calculating a Power Set (using Java). 2007. <https://jvalentino.blogspot.com/2007/02/shortcut-to-calculating-power-set-using.html>

[17] Cocke–Younger–Kasami (CYK) Algorithm – GeeksforGeeks. 2023. <https://www.geeksforgeeks.org/cocke-younger-kasami-cyk-algorithm/>

[18]java - What is the best way to find common elements from 2 sets? – Stack Overflow. 2019. <https://stackoverflow.com/questions/56488685/what-is-the-best-way-to-find-common-elements-from-2-sets>

[19] Handling continuous user input in a CLI app in Java – Stack Overflow. 2015. <https://stackoverflow.com/questions/30677981/handling-continuous-user-input-in-a-cli-app-in-java>

[20] Finding the N-th Occurrence of a Substring in Java | Baeldung. 2023. <https://www.baeldung.com/java-locate-nth-match-substring>

[21] Context-free grammar – Wikipedia. 2024. <https://en.wikipedia.org/wiki/Context-free_grammar>