Проект по Обектно-ориентирано програмиране

Тема: “Контекстно-свободна граматика”

Изготвил:

Кристина Бориславова Попова, ф.н. 81933, Компютърни науки, втори курс, втори поток, шеста група

1.Увод

1.1. Описание и идея на проекта

Основната идея на проекта “Контекстно-свободна граматика” е реализация на програма, която поддържа операции с контекстно-свободна граматика. Приложението отваря и прочита данните за дадена контекстно-свободна граматика от текстов файл (с разширение .txt), позволявайки на потребителя да използва създадените функции. След указание на потребителя направените промени могат да се запишат в първоначално отворения файл или да се запазят в нов файл, указан от потребителя. Ако потребителят не е указал какво да се направи с новите промени, те не се запазват никъде.

1.2. Цел и задачи на разработката

Целта на проекта е създаването на програма, която да поддържа съответните функционалности с контекстно-свободна граматика, с които позволява на потребителя да:

- отваря съдържанието на даден файл;
- затваря отворения файл;
- записва направените промени в нов, указан от потребителя, файл;
- вижда кратка информация за поддържаните от програмата команди;
- излиза от програмата;
- извежда списък с идентификаторите на всички прочетени граматик;
- извежда граматиката по подаден идентификатор;
- записва граматиката във файл;
- добавя правила;
- премахва правило по пореден номер;
- проверява дали дадена граматика е в нормална форма на Чомски;
- преобразува граматика в нормална форма на Чомски;
- проверява дали езикът на дадена контекстно-свободна граматика е празен;
- намира резултат от изпълнението на операцията “итерация”’;
- проверява дали дадена дума е в езика на дадена граматика;
- намира обединението на две граматики и създава нова граматика;
- намира конкатенацията на две граматики и създава нова граматика;

2. Преглед на предметната област

2.1. Основни дефиниции и концепции

Обектно-ориентираното програмиране е сравнително нова програмистка парадигма, принуждаваща програмистите да преразгледат мисленето си за програмирането, за смисъла на изпълнение на програмата и за това как информацията да бъде структурирана в компютъра.

Обектно-ориентираното програмиране е съсредоточено върху обектите, които капсулират състояние и поведение. Тази дефиниция оприличава обектите на променливи величини от абстрактни типове данни. Възможността за дефиниране на класове позволява да се създават и обработват типове данни, които липсват в езика и да се създават специфични приложения.

Средствата за дефиниране на производни класове и наследяването са сред най-важните характеристики на обектно-ориентираното програмиране. Като се използва механизмът на наследяване от съществуващ клас може да се създаде нов клас, който да използва вече създадените компоненти и поведение на базовия/те си клас/ове. Така се избягва повторното дефиниране на тези компоненти и поведение.

Терминът абстрактен тип данни е много важен в компютърната наука и особено в обектно-ориентираното програмиране. Абстрактните типове данни са пример за реализирането на пакетни данни и операции в едно цяло, позволяващо тяхната взаимосвързаност. Такива средства предоставят класовете. Информационното скриване на реализацията на типа данни се изразява в това, че програмист, използващ величина от някакъв абстрактен тип данни, не трябва да бъде напълно запознат с вътрешната структура на типа. Начинът, по който типът е реализиран, трябва да бъде скрит от потребителя. Открити трябва да са единствено свойствата на типа, които се изразяват чрез неговите операции.

Контекстно-свободна граматика е четворка от вида $G=(V,\Sigma,R,S)$, където

- V е крайно множество от променливи.
- Σ е крайно множество от букви, $\Sigma \cap V = \emptyset$
- $R \subseteq (V \cup \Sigma)^*$, крайно множество от правила
- $S \in V$ е начална променлива

Нормална форма на Чомски

- Една безконтекстна граматика е в нормална форма на Чомски, ако всяко правило е от вида $A \rightarrow BC$ и $A \rightarrow a$, като B, C не могат да бъдат променливата за начало S.

3.Проектиране

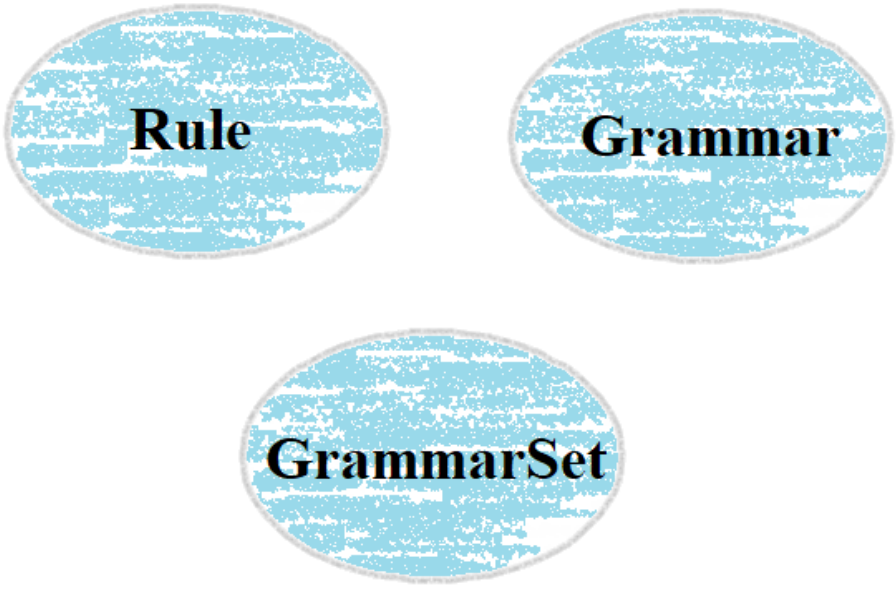
3.1.Обща архитектура – ООП дизайн

За проекта „Контекстно-свободна граматика“ бяха реализирани класовете `Rule`, `Grammar` и `GrammarSet`. За работата с командите, беше използван поведенческият шаблон за дизайн - *command pattern*, за който бяха необходими следните класове :

- `Invoker`
- `ICommand`
- `Validator` – в който се валидира въведения вход от потребителя.
- `OpenCommand, SaveCommand, ExitCommand, SaveAsCommand, CloseCommand, HelpCommand, ListCommand, PrintCommand, AddRuleCommand, ChomskyCommand, IterCommand`

3.2. Диаграми (на структура и поведение - по обекти, слоеве с най-важните извадки от кода)

На фиг.1 са изобразени класовете, които са използвани в архитектурата на проекта „Контекстно-свободна граматика“. Класът `Rule` е необходим, защото всяка граматика си има правила. Класът `Grammar` е самата граматика, а класът `GrammarSet` съдържа всички граматики и функционалности, с които работи програмата.



Фигура 1

4.Реализация и тестване

4.1. Реализация на класовете

Класът Rule е правилото, което ни е необходимо за граматиките, защото всяка граматика има множество от правила. Член-данните на класа са променлива и множество от продукции. Част от реализираните член-функции са:

- `std::string getVariable()const;` - гетър връщащ променливата(нетерминала) на правилото.
- `std::vector<std::string> getProduction()const;` - гетър връщащ множеството от продукциите на правилото.
- `void print(std::ostream& os)const;` - метод принтиращ дадено правило по подаден поток;

Класът Grammar е граматиката. Член-данните на класа са множество от правила, идентификатор, множество от променливи(нетерминали, големи латински букви), азбука - множество от терминални символи (малки латински букви), начална променлива(наричана още стартов нетерминален символ). По-важните реализирани член-функции са:

- `bool checkVariable(const std::string ch)const;` - член-функция, проверяваща дали началният символ на стринг е главна латинска буква.
- `bool checkTerminal(const char ch)const;` - член-функция, проверяваща дали даден символ е малка латинска буква или цифра.
- `bool checkTerminalSet(const char ch)const;` - член-функция, проверяваща дали дадена буква е от азбуката на граматиката.
- `bool checkUpper(const char ch) const;` - член-функция, проверяваща дали даден символ е главна латинска буква.
- `void createId();` - член-функция, създаваща на всяка граматика идентификатор от вида “пореден номер създадена граматика – начална променлива”.
- `std::string getId()const;` - *гетър* връщащ идентификатора на граматиката.
- `std::vector<std::string> getVariables()const;` - *гетър* връщащ множеството от нетерминали.
- `std::vector<char> getTerminals()const;` - *гетър* връщащ азбуката на дадена граматика.
- `std::vector<Rule*> getRules()const;` - *гетър* връщащ множеството от правила на дадена граматика.
- `char getStartVariable()const;` - *гетър* връщащ стартовата променлива на дадена граматика.
- `void addRule(const Rule* r);` - член-функция, добавяща ново валидно правило. Функцията прави поредица от проверки преди да добави правилото. Правило се добавя, само ако е валидно.
- `void removeRule(int index);` - член-функция, която премахва правило на подаден номер(индекс)
- `void save(std::ostream& os)const;` - член-функция, записваща граматиката.

- `void chomsky()const;` - член-функция, която проверява дали граматика е в нормлна форма на Чомски.
- `void iter();` - член-функция, която намира резултат от изпълнението на операцията “итерация” (звезда на Клини) над граматика.

Класът GrammarSet съдържа множеството от всички граматики, с които работи системата. Допълнителните член-данни на класа са `std::string fileName`, в която помним името на последно отворения файл, и `bool isOpen`, която е `true`, когато сме отворили даден файл, и `false`, когато не сме отворили даден файл или сме затворили вече отворения такъв.

4.2. Планиране, описание и създаване на тестови сценарии (създаване на примери)

Системата беше тествана върху следните тестови сценарии:

- добавяне на невалидни правила
- добавяне на невалидни променливи
- добавяне на невалидни терминали
- проверяване дали дадена граматика е в нормална форма на Чомски
- премахване на правило с невалиден индекс

5.Заключение

5.1. Обобщение на изпълнението на началните цели

В проекта са реализирани голяма част от исканите функционалности. Бяха създадени, разработени и разширени (с допълнителни член-функции и член-данни) класове, които удовлетворяват поставената задача. Системата беше тествана и коригирана, когато не отговаряше на изискванията.

5.2. Насоки за бъдещо развитие и усъвършенстване

Нещата, които биха усъвършенствали проекта са реализиране на всички искани функционалности и подобряване на реализацията на член-функциите, които валидират дадена граматика.

Използвана литература

1. Тодорова, Магдалина. “Обектно-ориентирано програмиране на базата на езика C++”,
2. Вътев, Стефан, „Езици, автомати, изчислимост“, 2015