Синтактичен анализ

1. Същност и цел на синтактичния анализ

Целта на синтактичният анализ е да провери дали дадена входна програма е валидно изречение на входния език, т.е. дали изречението се поражда и съответно разпознава в резултат на краен брой прилагания на правилата на граматиката.

2. Граматика

Синтаксисът на входния език се описва с подходяща граматика. Входният език на учебния компилатор както и повечето езици за програмиране се дефинира с контекстно-свободна граматика (граматика от тип 2).

Граматиката се състои от правила. Всяко правило се състои от лява и дясна страна. Лявата страна съдържа единствен нетерминален символ, а дясната страна съдържа низ от терминални и/или нетерминални символи или празен низ.

Нетерминален е символ, който се замества с други символи от граматиката.

Терминален е символ, който не може да се замести с други символи.

В граматиката се използват следните специални означения:

- **(**част от правило**)** – изпълнява се точно 1 път

- правило 1 **|** правило 2 – прилагане на правило 1 ИЛИ правило 2

- **?** - 0 или 1 път

- ***** - 0 или повече пъти

- **+** - 1 или повече пъти

Граматиката на входния език е дадена във файла ParserGrammar.txt, включен в проекта Compiler_students.

3. LL(1) граматика

За дефиниране на синтаксиса на входния език се използва граматика от тип LL(1). Абревиатурата LL(1) има следния смисъл:

- L (left) - прочитане на входната програма (изречение) от ляво на дясно;
- L (left) - прилагане на правилата от ляво на дясно;
- 1 – в правила с две или повече разклонения изборът на клон се изпълнява чрез прочитане на една лексема (символ) напред.

LL(1) граматиките са подмножество на контекстно свободните граматики. Всяко правило в LL(1) граматика отговаря на следните изисквания:

- Изискване 1 се отнася за всички правила с алтернативни клонове, например

A : A1 | A2 | ... | An.

За да може да се реализира такова правило с LL(1) метод, трябва да е изпълнено условието

first(Ai) ∩ first(Aj) = ∅, за всички i != j

, където first(Ai) и first(Aj) са съответно множества от терминални символи, с които е допустимо да започват нетерминалните символи Ai и Aj.

- Изискване 2 се отнася за правила, генериращи празен низ, т.е.

A : ε

За да може да се реализира такова правило с LL(1) метод, трябва да е изпълнено условието

first(A) ∩ follow(A) = ∅

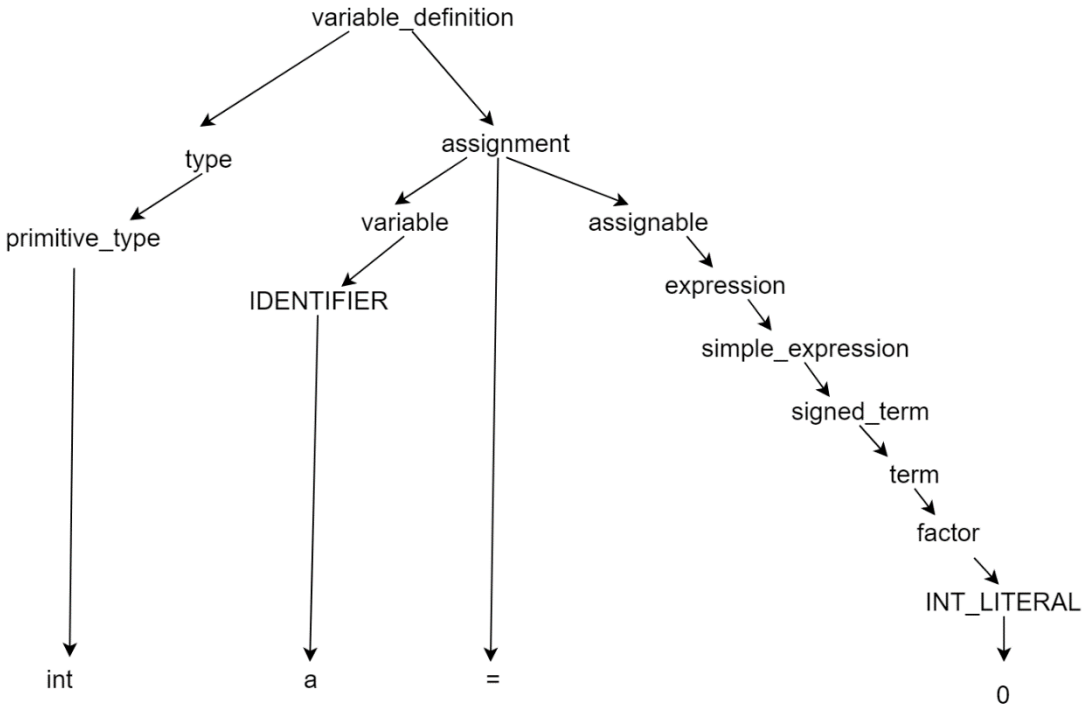
, където follow(A) е множество от терминални символи, които могат да следват непосредствено нетерминалния символ A.

4. Низходящ синтактичен анализ (LL(1))
   4.1. Рекурсивно спускане

Синтактичният анализ представлява построяване на синтактично дърво.

За разпознаване на изречението (входната програма) в учебния компилатор е използван низходящ (top-down) синтактичен анализ. При него синтактичното дърво се построява отгоре надолу, като се започва от корена на дървото, обозначен със стартовия символ в граматиката, а следващите върхове съответстват на синтаксиса на анализираното изречение. Граматичните правила се прилагат отляво надясно. Построяването на дървото приключва, когато в листата се получи разпознаваното изречение.

Фиг. 1 показва синтактичното дърво за нетерминален символ variable_definition.



Фиг. 1. Синтактично дърво за variable_definition

За реализация на низходящия синтактичен анализ е използван метод на рекурсивното спускане (LL(1) – анализ). Синтактичният анализатор се състои от отделни функции. В общия случай на всеки нетерминален символ от граматиката съответства една функция. Действието на всяка функция се състои в разпознаване на подниз, започващ с текущата лексема, съответстващ на дясната страна на правилото, дефиниращо нетерминалния символ. Дадена функция може да извиква други функции или самата себе си. След като функцията разпознае подниза, свързан с реализирания нетерминален символ, тя завършва работата си и предава управлението на извикващата функция. Текуща става следващата лексема от входния поток. Ако функцията не разпознае нетерминалния символ, се извежда съобщение за грешка.

4.2. Примери за реализация на синтактични функции

Обработване на терминален символ става в метод accept. Този метод сравнява текущо прочетената с очакваната лексема според правилото за разпознавания нетерминален символ. Ако лексемите не съвпадат, се генерира изключение, което извежда съобщение за синтактична грешка. При съвпадение на прочетената и очакваната лексема анализът продължава с прочитане на следващата лексема от потока.

```
protected void accept(TokenType tokenType) {
    if (currentToken.getTokenType() != tokenType) {
        throw new SyntaxException("Token doesn't match! Expected " + tokenType + ", Got " + currentToken.getTokenType(), currentToken);
    }
```

```
        currentToken = lexer.nextToken();
    }
```

Примери за синтактични функции:

- Метод, разпознаващ стартовия нетерминален символ program:

```
public AST entryRule() {
    accept(TokenType.PROGRAM);
    accept(TokenType.LBRACKET);
    programBody();
    accept(TokenType.RBRACKET);
    return currentNode;
}
```

- Метод, разпознаващ оператор while (правило whileStatement):

```
void whileStatement() {
    accept(TokenType.WHILE);
    accept(TokenType.LPAREN);
    expression();
    accept(TokenType.RPAREN);
    block();
}
```

В правила, които съдържат множество клонове, избор на клон се реализира чрез поглеждане една лексема напред.

Например за оператор (правило statement) се проверява дали следващата лексема е стартов символ от прост оператор (правило simple_statement). В този случай се избира прост оператор (simple_statement), в противен случай – съставен оператор (compound_statement).

- Метод, разпознаващ оператор (правило statement):

```
void statement() {
if (TokenType.isSimpleStatementTerminal(currentToken.getTokenType())){
        simpleStatement();
```

```
        accept(TokenType.SEMICOLON);
    } else {
        compoundStatement();
    }
}
```

Задачи:

1. Във файла ParserImpl.java да се довършат телата на синтактичните функции, маркирани с коментар /* ToDo Syntax Analysis … */.

2. Да се тества работата на синтактичния анализатор с различни входни програми.