ЛЕКЦІЯ 10

СИНТАКСИЧНИЙ АНАЛІЗ

Існує три основних типи синтаксичних аналізаторів (англ.: parser), що реалізують три стратегії розбору:

1) стратегія аналізу зверху вниз (низхідний аналіз);
2) стратегія аналізу знизу вверх (висхідний аналіз);
3) змішана стратегія.

Як було показано вище, загальна схема синтаксичного аналізу на основі недетермінованого МП-автомату працює з поверненнями.

Проте, якщо на граматику, що породжує мову, накласти певні обмеження, а також ефективно використати стек автомата, то можна застосовувати деякі відомі прийоми, що дозволяють побудувати ефективні синтаксичні аналізатори, що працюють без повернень.

На практиці в синтаксичних аналізаторах (СА) існуючих мов програмування використовується саме такий підхід. Зокрема було визначено чотири відносно великих класи КВ-граматик, які дозволяють побудувати СА без повернень:

1) **LL(k)**-граматики;
2) **LR(k)**-граматики;
3) **RL(k)**-граматики;
4) **RR(k)**-граматики.

Перша буква **L** чи **R** у позначенні вказує напрям перегляду вхідного рядка (зліва направо чи справа наліво).

Друга буква (**L** чи **R**) в позначенні означає вид виводу (лівий чи правий відповідно).

Буква **k** в дужках означає кількість символів вхідного рядка, що аналізуються наперед при виконанні розбору.

Іншими словами можна дати наступні спрощені визначення:

Визначення 1. LL(k)-граматика – це граматика, що допускає безповоротний синтаксичний аналіз при вводі вхідного рядка **зліва направо** і реалізує **лівий** вивід із загляданням наперед і аналізом **k** символів вхідного рядка.

Визначення 2. LR(k)-граматика – це граматика, що допускає безповоротний синтаксичний аналіз при вводі вхідного рядка **зліва направо** і реалізує **правий** вивід із загляданням наперед і аналізом **k** символів вхідного рядка.

Визначення 3. RL(k)-граматика – це граматика, що допускає безповоротний синтаксичний аналіз при вводі вхідного рядка **справа наліво** і реалізує **лівий** вивід із загляданням наперед і аналізом **k** символів вхідного рядка.

Визначення 4. RR(k)-граматика – це граматика, що допускає безповоротний синтаксичний аналіз при вводі вхідного рядка **справа наліво** і реалізує **правий** вивід із загляданням наперед і аналізом **k** символів вхідного рядка.

З використанням **LL(k)** і **RR(k)**-граматик реалізується низхідна (зверху вниз) стратегія синтаксичного розбору, а з використанням **LR(k)** і **RL(k)**-граматик – висхідна (знизу вверх) стратегія синтаксичного розбору.

На практиці використовуються в основному **LL(k)** та **LR(k)**-граматики.

Строге визначення LL(k) і LR(k)-граматик

Введемо декілька позначень:

якщо k – невід'ємне ціле число;

$\alpha \in V^*$;

$|\alpha|$ – довжина рядка α;

$\Rightarrow^{L}{}_{*}$ – лівий вивід;

$\Rightarrow^{R}{}_{*}$ – правий вивід;

то $k{:}\alpha = \begin{cases} \text{якщо } |\alpha| \geq k, \text{ то } k{:}\alpha \text{ — це перші k символів рядка } \alpha \\ \text{інакше } k{:}\alpha \text{ – це весь рядок } \alpha \end{cases}$

$\alpha{:}k = \begin{cases} \text{якщо } |\alpha| \geq k, \text{ то } \alpha{:}k \text{ — це останні k символів рядка } \alpha \\ \text{інакше } \alpha{:}k \text{ – це весь рядок } \alpha \end{cases}$

Визначення 5. КВ-граматика є LL(k)-граматикою, якщо для будь-якого $A \in N$ і будь-яких рядків $\beta, \beta', \gamma \in T^*$ і $\alpha, \alpha', \delta \in V^*$ виконується наступна умова:

якщо $S \Rightarrow^{L}{}_{*} \gamma A \delta \Rightarrow^{L} \gamma \alpha\delta \Rightarrow^{L}{}_{*} \gamma \beta$,

та $S \Rightarrow^{L}{}_{*} \gamma A \delta \Rightarrow^{L} \gamma \alpha' \delta \Rightarrow^{L}{}_{*} \gamma \beta'$,

та $k{:}\beta = k{:}\beta'$,

то $\alpha = \alpha'$

Визначення 6. КВ-граматика є LR(k)-граматикою, якщо для будь-яких A і $A' \in N$ і будь-яких рядків $\alpha, \alpha', \beta, \beta' \in V^*$ і $\gamma, \gamma' \in T^*$ виконується наступна умова:

якщо $S \Rightarrow^{R}{}_{*} \beta A \gamma \Rightarrow^{R} \beta \alpha \gamma$

та $S \Rightarrow^{R}{}_{*} \beta' A' \gamma' \Rightarrow^{R} \beta' \alpha' \gamma'$

та $(|\beta\alpha| + k){:}\ \beta \alpha \gamma = (|\beta\alpha| + k) : \beta' \alpha' \gamma'$

то $\beta = \beta'$; $A = A'$; $\alpha = \alpha'$.

Айронс запропонував метод розбору, який не є ні низхідним, ні висхідним і отримав назву «розбір з лівого кута», а відповідний йому клас граматик отримав назву **LC(k)**-граматики. Цей метод розбору працює з неповними поверненнями. **LC(k)**-граматики були досліджені Розенкранцем і Л'юісом, які показали, що всі **LC(k)**-мови є **LL(k)**-мовами і навпаки.

При розгляданні конкретних алгоритмів синтаксичного аналізу корисно використовувати множини FIRST(x) і FOLLOW(x).

Множини FIRST і FOLLOW

При побудові багатьох аналізаторів зручно використовувати дві функції, зв'язані з граматикою G. Ці функції, FIRST і FOLLOW, дозволяють побудувати таблицю передбачуваного розбору для G, якщо, звичайно, це можливо.

Множини, що формуються цими функціями, можуть, крім того, бути використані при відновленні після помилок.

Якщо u – довільний рядок символів граматики, то FIRST(u) – множина терміналів, з яких починаються рядки, що виводяться із u. Якщо $u \Rightarrow^* \varepsilon$, то ε також належить FIRST(u).

Визначимо FOLLOW(A) для нетерміналу A як множину терміналів a, що можуть з‘явитись безпосередньо справа від A в деякій сентенціальній формі, тобто множина терміналів a, таких, що для деяких u і v існує вивід виду $S \Rightarrow^* uAav$. Відмітимо, що між A і a в процесі виводу можуть з’являтись нетермінальні символи, із яких виводиться ε. Якщо A може бути найправішим символом деякої сентенціальної форми, то # (обмежувач) належить FOLLOW(A).

Для побудови FIRST(X) для всіх символів граматики X можна застосувати наступний алгоритм.

Алгоритм FIRST. Побудова множин FIRST для символів граматики.

Крок 1. Якщо X – термінал, то FIRST(X) – це {X}; якщо X – нетермінал, то FIRST(X) = {}.

Крок 2. Якщо є правило виводу $X \to \varepsilon$, то додати ε до FIRST(X).

Крок 3. Поки ще можна додавати нові елементи або ε до будь-якої множини FIRST(X):

1) все, що належить $FIRST(Y_1)$, належить також і FIRST(X).
2) включити a в FIRST(X), якщо X – нетермінал і існує правило виводу $X \to Y_1Y_2...Y_k$, а також для деякого індексу i $a \in FIRST(Y_i)$ і ε належить всім $FIRST(Y_1),...,FIRST(Y_{i-1})$, тобто $Y_1...Y_{i-1} \Rightarrow^* \varepsilon$.
3) якщо ε належить $FIRST(Y_i)$ для всіх i=1,2,...,k, то додати ε до FIRST(X).
4) якщо із Y_1 не виводиться ε, то нічого більше не додаємо до FIRST(X), але якщо $Y_1 \Rightarrow^* \varepsilon$, то додаємо $FIRST(Y_2)$, і т.д.

Тепер множину FIRST для будь-якого рядка $X_1 X_2 ... X_n$ можна сформувати наступним чином:

Крок 1. $FIRST(X_1X_2...X_n) = \{\}$.

Крок 2.

1) Додаємо до $FIRST(X_1X_2...X_n)$ всі символи із $FIRST(X_1)$, які є відмінними від ε.
2) Додаємо також символи із $FIRST(X_2)$, відмінні від ε, якщо $\varepsilon \in FIRST(X_1)$; додаємо символи із $FIRST(X_3)$, відмінні від ε, якщо ε належить як $FIRST(X_1)$, так і $FIRST(X_2)$ і т.д.
3) Наостанок додамо ε до $FIRST(X_1X_2...X_n)$, якщо $\varepsilon \in FIRST(X_i)$ для всіх i.

Для обчислення FOLLOW(A) для нетерміналу A можна застосувати наступний алгоритм.

Алгоритм FOLLOW. Побудова FOLLOW(X) для всіх X, які є нетерміналами граматики.

Крок 1. FOLLOW(X) = {}.

Крок 2. Помістити # в FOLLOW(S), де S – початковий символ(аксіома) і # – правий кінцевий маркер.

Крок 3. Якщо є правило виводу $A \to uBv$, то все із FIRST(v), за виключенням ε, додати до FOLLOW(B).

Крок 4. Поки ще можна додавати символи до будь-якої з множин FOLLOW(X): якщо є правило виводу $A \to uB$ або $A \to uBv$, де FIRST(v) містить ε (тобто $v \Rightarrow^* \varepsilon$), то всі символи із FOLLOW(A) додати до FOLLOW(B).

Приклад. Розглянемо граматику:

1. E → T E'
2. E' → + T E'
3. E' → ε
4. T → F T'
5. T' → * F T'
6. T' → ε
7. F → (E)
8. F → id

1. <вираз> → <доданок><рядок доданків>
2. <рядок доданків> → + <доданок><рядок доданків>
3. <рядок доданків> → ε
4. <доданок> → <множник><рядок множників>
5. <рядок множників> → * <множник><рядок множників>
6. <рядок множників> → ε
7. <множник> → (<вираз>)
8. <множник> →i d

Для неї

```
FIRST(E)=FIRST(T)=FIRST(F)={(,id}
FIRST(E')={+, ε}
FIRST(T')={*, ε}
FOLLOW(E)=FOLLOW(E')={),#}
FOLLOW(T)=FOLLOW(T')={+,),#}
FOLLOW(F)={+,*,),#}
```

1. E → T E'
2. E' → + T E'
3. E' → ε
4. T → F T'
5. T' → * F T'
6. T' → ε
7. F → (E)
8. F → id

Наприклад, id та ліва дужка додаються до FIRST(F) на кроці 3 при i=1, оскільки FIRST(id) = {id} і FIRST('(') = {'('} відповідно до кроку 1. На кроці 3 при i=1, відповідно до правила виводу T → FT' до FIRST(T), додаються також id та ліва дужка. На кроці 2 в FIRST(E') включається ε.

На кроці 1 для формування множин FOLLOW в FOLLOW(E) включаємо #. На кроці 2, використовуючи правило виводу F → (E), до FOLLOW(E) додається також права дужка. На кроці 3, відповідно до правила E → TE', в FOLLOW(E') включаються # та права дужка. Оскільки E' ⇒* ε, вони (тобто # та права дужка) також попадають в FOLLOW(T). У відповідності з правилом виводу E → TE', на кроці 2 в FOLLOW(T) включаються всі символи із FIRST(E'), відмінні від ε.

Алгоритми синтаксичного розбору зверху вниз

Розглянемо три алгоритми синтаксичного аналізу, що працюють по низхідній стратегії:

1) аналізуюча машина Кнута;
2) алгоритм, що працює по методу рекурсивного спуску;
3) алгоритм таблично-керованого передбачаючого розбору.

Нагадаємо, що стратегія розбору зверху вниз може бути реалізована для **LL(k)**-граматик, причому, чим менше **k**, тим ефективніший аналізатор можна побудувати.

Тому особливої уваги заслуговують **LL(1)**-граматики і умови відповідності довільної граматики **G** класу **LL(1)**, оскільки вони допускають безповоротний аналіз зверху вниз із загляданням наперед всього на один символ.

Аналізуюча машина Кнута (АМК)

Розглянемо універсальний алгоритм низхідного синтаксичного аналізу, запропонований Дональдом Кнутом, який отримав назву «аналізуюча машина Кнута (АМК)».

Спочатку розглянемо роботу АМК з поверненнями для не **LL(k)-**граматики, а потім умови, яким повинна задовольняти граматика, щоб належати класу **LL(1)**-граматик і допускати безповоротний розбір.

Розглянемо наступну граматику мови, що нагадує мову «логічних виразів».

<логічний вираз> : : = <відношення> | (<логічний вираз>)

<відношення> : : = <вираз> = <вираз>

<вираз> : : = a | b |(<вираз> + <вираз>)

Запишемо цю граматику для компактності в модифікованій формі БНФ, у якій замість символу «::=» використовується символ «→», а синтаксичні класи позначаються великими літерами, а не беруться в кутові дужки.

$$\begin{aligned} &B \rightarrow R \mid (B) \\ &R \rightarrow E = E \\ &E \rightarrow a \mid b \mid (E+E) \end{aligned} \qquad (2.1)$$

Відповідна програма для аналізуючої машини Кнута буде такою:

Адреса операції	**Код Операції**	**AT (Адреса True)**	**AF (Адреса False)**
B	[R]	T	
	(		F
	[B]		F
	)	T	F
R	[E]		F
	=		F
	[E]	T	F
E	a	T	
	b	T	
	(		F
	[E]		F
	+		F
	[E]		F
	)	T	F
Start	[B]		**ПОМИЛКА**
	#	**ОК**	**ПОМИЛКА**

Варто звернути увагу на відповідність між граматикою і АМК-програмою. Останні два рядки програми відповідають ще одному граматичному правилу

$$Start \rightarrow B \#$$

де «#» – спеціальний символ-обмежувач, який зустрічається лише в самому кінці рядка, що аналізується.

Аналізуюча машина – це абстрактна машина для аналізу рядків в деякому алфавіті. Вона читає із «вхідного рядка» кожен раз по одній літері згідно з деякою програмою. Програма аналізуючої машини – це набір процедур, що рекурсивно викликають одна одну; сама програма по суті є однією з таких процедур. Кожна процедура намагається виявити у вхідному рядку присутність деякої конкретної синтаксичної конструкції і по закінчені роботи повертає значення «true» чи «false» в залежності від того, чи був пошук успішним.

Нехай $S_1 S_2 \ldots S_n$ – вхідний рядок, і S_h – «поточна» літера, що читається машиною.

Команда цієї машини складається із трьох полів: поля коду операції і двох адрес, AT (Адреса True) і AF (Адреса False). Процедури записуються з використанням двох типів команд, що відповідають двом різним видам кодів операцій.

Тип 1: Код операції – літера *a* із алфавіту.

Тип 2: Код операції – адреса процедури, що стоїть в квадратних дужках [A].

Ці команди виконуються наступним чином:

Тип 1: якщо $S_h = a$, то пропустити *a* (тобто встановити $h := h + 1$) і перейти до AT, інакше перейти до AF.

Тип 2: викликати процедуру, що починається в комірці A (рекурсивно); якщо вона повернула значення true, то перейти до AT, інакше, якщо вона повернула значення false, то перейти до AF.

Кожне з полів AT і AF може містити або адресу команди, або один зі спеціальних символів: *T*, *F* або порожнє поле. Порожнє поле адреси відповідає адресі команди, що записана в наступному рядку. Якщо він містить *T*, здійснюється вихід із процедури із значенням «true». Якщо воно містить *F*, то відбувається вихід із процедури із значенням «false» і змінна *h* знову набуває того значення, яке вона мала до входу в процедуру.

Таким чином, мається на увазі, що при виклику процедури по коду операції типу 2, завжди разом з адресою повернення зберігається і значення змінної *h*.

Процедура Start виконає перехід на мітку «ОК», лише якщо весь рядок, що аналізується, є логічним виразом B, за яким слідує символ «#», інакше вона виконає перехід на мітку «ПОМИЛКА».

Приклад.

Використовуючи розглянуту вище АМК програму, виконаємо розбір наступного вхідного рядка:

(	a	=	(	b	+	a	)	)	#
S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}

Якщо почати з адреси Start, то виконається така послідовність дій (спочатку h = 1, тобто оброблюється перша літера S_1).

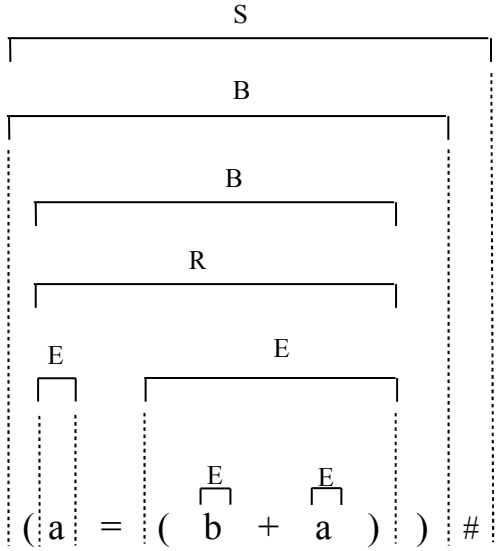
```
Виклик B (h = 1)
   Виклик R (h = 1)
      Виклик E (h = 1)
         Пошук a: ні
         Пошук b: ні
         Пошук (: так, встановити h : = 2
         Виклик E (h = 2).
            Пошук a: так; встановити h : = 3
            Повернення, true.
         Пошук + : ні
         Повернення, false; встановити h : = 1
      Повернення, false; встановити h : = 1
   Пошук (: так, встановити h : = 2
Виклик B (h = 2)
   Виклик R (h = 2)
і т.д.
```

Подібне трасування роботи АМК машини для вказаного вхідного рядка приведене в таблиці 1.

Таблиця 1

Символ що аналізується	Адреса Операції	Операція	Результат	Дія
S_1 = "("	Start	[B]		Перехід на адресу операції **B**
S_1 = "("	B	[R]		Перехід на адресу операції **R**
S_1 = "("	R	[E]		Перехід на адресу операції **E**
S_1 = "("	E(first)	a	AF = ' '	Перехід на наступний рядок
S_1 = "("		b	AF = ' '	Перехід на наступний рядок
S_1 = "("		(	розпізнано AT = ' '	Перехід на наступний рядок
S_2 = "a"		[E]		Перехід на адресу операції **E**
S_2 = "a"	E(second)	a	розпізнано AT = T	Повернення в попереднє **E** з "T"
S_3 = "="	E(first)	Обр. рез-ту	AT = ' '	Перехід на наступний рядок
S_3 = "="		+	AF = F	Повернення в **R** з "F"
S_1 = "("	R	Обр. рез-ту	AF = F	Повернення в **B** з "F"
S_1 = "("	B		AF = ' '	Перехід на наступний рядок
S_1 = "("		(	розпізнано AT = ' '	Перехід на наступний рядок
S_2 = "a"		[B]		Перехід на адресу операції **B**
S_2 = "a"	B(second)	[R]		Перехід на адресу операції **R**
S_2 = "a"	R	[E]		Перехід на адресу операції **E**
S_2 = "a"	E	a	розпізнано AT = 'T'	Повернення на **R** з "T"
S_3 = "="	R	Обр. рез-ту	AT = ' '	Перехід на наступний рядок
S_3 = "="		=	розпізнано AT = ' '	Перехід на наступний рядок
S_4 = "("		[E]		Перехід на адресу операції **E**
S_4 = "("	E(first)	a	AF = ' '	Перехід на наступний рядок
S_4 = "("		b	AF = ' '	Перехід на наступний рядок
S_4 = "("		(	розпізнано AT = ' '	Перехід на наступний рядок
S_5 = "b"		[E]		Перехід на адресу операції **E**
S_5 = "b"	E(second)	a	AF = ' '	Перехід на наступний рядок
S_5 = "b"		b	розпізнано AT = 'T'	Повернення в попереднє **E** з "T"
S_6 = "+"	E(first)	Обр. рез-ту	AT = ' '	Перехід на наступний рядок
і так далі				

По завершенні роботи АМК управління буде передано на адресу «ОК», при цьому історія викликів процедур, повернення з яких відбулось із значенням «true», відповідатиме наступній діаграмі «розбору» вхідного рядка:



Цю діаграму можна уявити собі побудованою зверху вниз в процесі виконання програми для аналізуючої машини.

Ліва частина кожної горизонтальної дужки в цій діаграмі будується при виклику процедури, а права добудовується при поверненні із процедури.

Проте існують граматики, для яких відповідна АМК-програма не буде працювати правильно.

Тому потрібно дослідити загальне питання: «Для яких граматик відповідна АМК-програма буде працювати правильно?»

Формування таблиці АМК (програмування АМК)

Припустимо спочатку, що як і в наведеному вище прикладі, всі БНФ-правила граматики записані в стандартній формі

$$X \to Y_1 \mid Y_2 \mid \ldots \mid Y_m \mid Z_1 Z_2 \ldots Z_n$$

де m, $n \geq 0$, $m + n > 0$ і всі $Y, Z \in V$, тобто це або термінальні символи (тобто є буквами алфавіту), або нетермінальні символи (тобто представляють синтаксичні класи).

Права частина правила містить m+1 варіантів; якщо m = 0, то вона приймає простий вигляд:

$$X \to Z_1 Z_2 \ldots Z_n$$

Якщо n = 0, то рядок $Z_1 Z_2 \ldots Z_n$ розглядається як порожній рядок.

АМК-програма, що відповідає правилу, записаному в стандартній формі, складається з наступних m+n інструкцій:

Адреса	Код операції	AT	AF
X	$[Y_1]$	T	
	$[Y_2]$	T	
	⋮	⋮	
	$[Y_m]$	T	*
	$[Z_1]$		F
	⋮		⋮
	$[Z_{n-1}]$		F
	$[Z_n]$	T	F

Якщо Y_i або Z_j є термінальними символами, то квадратні дужки при вказуванні відповідного коду операції потрібно видалити. Якщо n = 0, то адресу, що позначена через «*», потрібно замінити на F; в іншому випадку потрібно залишити пробіл.

Якщо БНФ-правило записане не в стандартній формі, тоді його можна привести до стандартної форми, через введення нових нетермінальних символів $Y_1 \ldots Y_m$ і додавання правила

$$Y_1 \to a_1$$

. . . .

$$Y_m \to a_m$$

Якщо, наприклад, наша БНФ-граматика містить правило

$$X \to AB \mid CD$$

то ми замінюємо його двома правилами

$$X \to Y \mid CD$$

$$Y \to AB$$