ЛЕКЦІЯ 12

Таблично-керований передбачаючий (прогнозуючий) розбір

На рис.1 зображена структура передбачаючого аналізатора, який визначає чергове правило з таблиці. Таку таблицю можна побудувати безпосередньо з граматики.

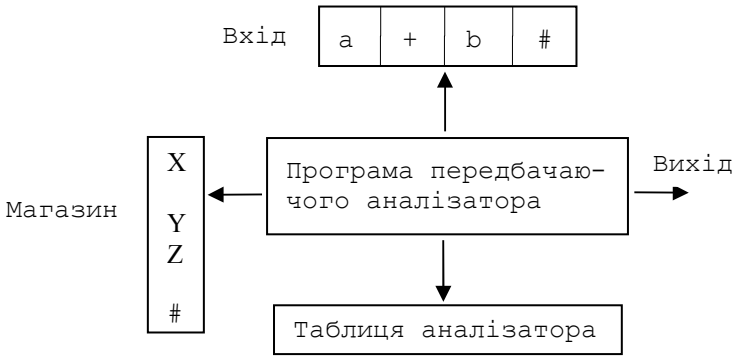


Рис 1.

Таблично-керований передбачаючий аналізатор має вхідний буфер, магазин, таблицю аналізу і вихід. Вхідний буфер містить рядок розпізнавання, за яким йде # – правий кінцевий маркер (ознаку кінця рядка). Магазин містить послідовність символів граматики з символом # на дні. Спочатку магазин містить початковий символ граматики на верхівці і символ # на дні. Таблиця аналізу – це двовимірний масив M [A,a], де A – нетермінал, і a – термінал або символ #.
Аналізатор керується програмою, яка працює наступним чином. Програма розглядає X – символ на верхівці магазину і a – поточний вхідний символ. Ці два символи визначають дію аналізатора. Є три подальші варіанти:

1. Якщо X = a = #, аналізатор зупиняється і повідомляє про успішне завершення розбору.

2. Якщо X = a ≠ #, аналізатор видаляє X з магазина і просуває покажчик входу на наступний вхідний символ.

3. Якщо X – нетермінал, програма бере з таблиці елемент M[X,a]. За цими координатами в таблиці зберігається або правило для нетерміналу X, або ознака помилки. Якщо, наприклад, M [X,a] = {X → UVW}, аналізатор заміняє X на верхівці магазину на WVU (в результаті на верхівці буде U). Будемо вважати, що аналізатор в якості виходу просто друкує використані правила виводу. Якщо M [X,a] = error, аналізатор звертається до підпрограми аналізу помилок.

Поведінка аналізатора може бути описана в термінах конфігурацій автомата розбору.

Спочатку аналізатор знаходиться в конфігурації, в якій магазин містить S# (S – початковий символ граматики) у вхідному буфері w# (w – вхідний рядок), змінна InSym містить перший символ вхідного рядка. Алгоритм, який використовує таблицю аналізатора M для здійснення розбору показано нижче (Алгоритм 1).

Алгоритм 1. Нерекурсивний передбачаючий аналіз.

```
repeat  X := верхній символ магазина;
        if X – термінал чи #
        then if X = InSym
             then
                 begin
                      видалити X з магазина;
                      InSym := черговий символ;
                 end
             else error()
        else /*X = нетермінал*/
             if M[X,InSym] = X → Y1Y2...Yk
             then
                 begin
                      видалити X з магазина;
                      помістити Yk,Yk-1,...Y1 в магазин (Y1 на верхівку);
                      вивести правило X → Y1Y2...Yk;
                 end
             else error() /*вхід таблиці M пустий*/
untilx X = #; /*магазин пустий*/
```

Приклад 1. Розглянемо граматику арифметичних виразів у такому вигляді:

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

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

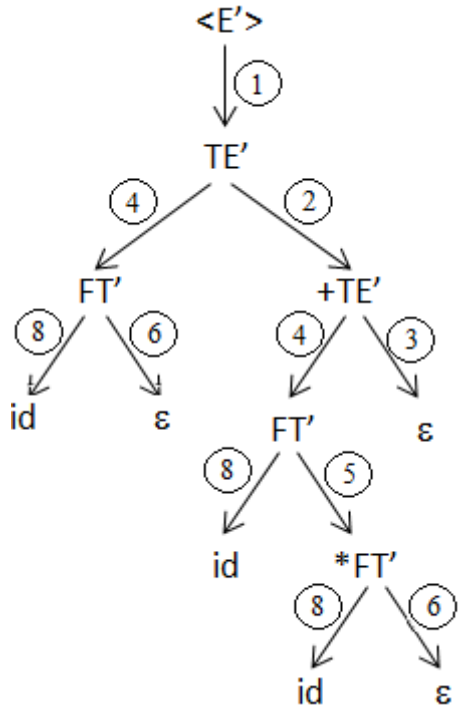
Таблиця передбачаючого аналізатора показана нижче (Таблиця 1). Тут пусті клітинки – входи помилок. Непусті клітинки містять правила, за якими виконується розгортка нетермінала.

Таблиця 1

Нетермінал	**Вхідний символ**					
	id	**+**	*****	**(**	**)**	**#**
E	E→ TE'			E→ TE'		
E'		E'→ +TE'			E'→ ε	E'→ ε
T	T→ FT'			T→ FT'		
T'		T'→ ε	T'→ *FT'		T'→ ε	T'→ ε
F	F→ id			F→ (E)		

Таблиця 2

Вхід	Магазин	Вихід (застосо-ване правило)	№ викор. правила
id+id*id#	E#		
id+id*id#	TE'#	E→TE'	1
id+id*id#	FT'E'#	T→FT'	4
id+id*id#	idT'E'#	F→id	8
+id*id#	T'E'#		
+id*id#	E'#	T'→ε	6
+id*id#	+TE'#	E'→+TE'	2
id*id#	TE'#		
id*id#	FT'E'#	T→FT'	4
id*id#	idE'T'#	F→id	8
*id#	T'E'#'		
*id#	F*T'E'#	T'→*FT'	5
id#	FT'E'#		
id#	idT'E'#	F→id	8
#	T'E'#		
#	E'#	T'→ε	6
#	#	E'→ε	3



Якщо на вхід подається рядок id + id * id, передбачаючий аналізатор виконує послідовність кроків, яка показана в таблиці 2. Вказівник вхідного рядка показує на найлівіший символ в колонці «Вхід». Якщо уважно проаналізувати дії аналізатора, то видно, що він виконує лівий вивід, тобто правила застосовуються в відповідності до лівого виводу. За вже переглянутими вхідними символами йдуть символи граматики в магазині (зверху вниз), що відповідає лівим сентенціальним формам виводу.
Дерево розбору для рядка id + id * id зображено на рис. 2.

Конструювання таблиць передбачаючого аналізатора

Для конструювання таблиць передбачаючого аналізатора за граматикою G може бути використано алгоритм, який базується на наступній ідеї. Припустимо, що $A \to \alpha$ – правило виводу граматики і $a \in FIRTS(\alpha)$. Тоді аналізатор виконує розгортку A по α, якщо вхідним символом є a. Складність виникає, коли $\alpha = \varepsilon$ чи $\alpha \Rightarrow * \varepsilon$. В цьому випадку потрібно розгорнути A в α, якщо поточний вхідний символ належить FOLLOW(A) або дорівнює #, причому $\# \in FOLLOW(A)$.

Алгоритм 2. Побудова таблиць передбачаючого аналізатора.
Для кожного правила виводу $A \to \alpha$ граматики виконати кроки 1 і 2:

Крок 1. Для кожного терміналу a з FIRST(u) додати $A \to u$ до M[A,a].

Крок 2. Якщо $\varepsilon \in FIRST(\alpha)$, додати $A \to \alpha$ до M[A,b] для кожного терміналу b з FOLLOW(A). Якщо $\varepsilon \in FIRST(\alpha)$ і $\# \in FOLLOW(A)$, додати $A \to \alpha$ до M[A,#].

Крок 3. Всі невизначені входи прирівняти до error.

Приклад 2. Застосуємо алгоритм 2 до граматики, яка була приведена в прикладі 1.
Оскільки FIRST(TE') = FIRST(T) = { (, id }, у відповідності з правилом виводу $E \to TE'$ входи M[E,(] і M[E,id] стають рівними $E \to TE'$.
В відповідності з правилом виводу $E' \to + TE'$ вхід M[E',+] буде дорівнювати $E' \to + TE'$.
В відповідності з правилом виводу $E' \to \varepsilon$ входи M[E',)] і M[E',#] будуть дорівнювати $E' \to \varepsilon$, оскільки FOLLOW(E') = {) , # }.
Таблиця аналізу, побудована алгоритмом 2, приведена в таблиці 1.

Умови безповоротного LL(1) синтаксичного аналізу

Якщо для **LL(1)**-граматики можна побудувати синтаксичний аналізатор, що працює без повернень, то отримаємо дві наступні переваги:
1) загальний час синтаксичного аналізу буде обмежений величиною, що пропорційна довжині вхідного рядка;
2) програма синтаксичного аналізатора може читати вхідні символи по одному, не зберігаючи раніше прочитані символи.

Відомо, що правила будь-якої КВ-граматики можна подати у вигляді правил, що належать лише наступним п’яти типам:

Тип 1. $X_p \to X_q \mid X_r$, де $q<p$ і $r<p$
Тип 2. $X_p \to X_q X_r$, де $q<p$ і якщо $X_q \to + \varepsilon$, то і $r<p$
Тип 3. $X_p \to X_q$, де $q<p$
Тип 4. $X_p \to a$
Тип 5. $X_p \to \varepsilon$

Для всіх типів правил $X_p, X_q, X_r \in N$, $a \in T$.

Відношення $q<p$ в даному випадку означає таку властивість: якщо граматика не має ліворекурсивних нетерміналів, то всі нетермінали $X_1, X_2, \ldots, X_n$ можна впорядкувати так, що
$X_p \Rightarrow^L + X_q$ лише тоді, коли $q<p$.

Наприклад, для граматики

$S \to P\ \#$

$P \to LE$

$E \to +LE \mid \varepsilon$

$L \to MT$

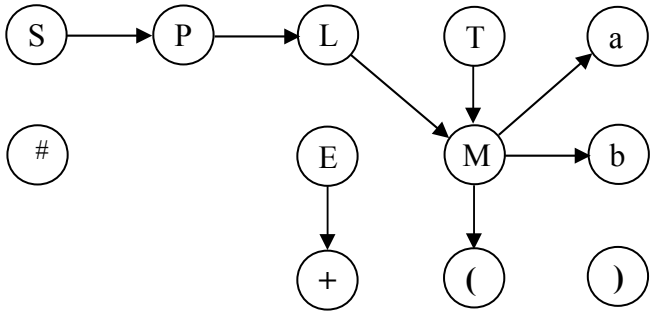
$T \to MT \mid \varepsilon$

$M \to a \mid b \mid (P),$

де $S, P, L, E, M, T \in N;$

$a, b \in T$

таким впорядкуванням буде E, M, T, L, P, S.



Умови безповоротного низхідного розбору формулюються наступним чином:

1. Жоден нетермінальний символ не є ліворекурсивним.
2. Для жодного правила типу 1 із X_q і X_r не виводяться рядки, що починаються з одного і того ж термінального символа, тобто $FIRST(X_q) \cap FIRST(X_r) = \varnothing$.
3. Для будь-якого правила типу 1, де X_r може породжувати порожній рядок $X_r \Rightarrow^* \varepsilon$, із іншого нетерміналу X_q не може виводитись рядок, що починається з терміналу, який належить множині FOLLOW нетерміналу лівої частини правила X_p, тобто завжди виконується $FIRST(X_q) \cap FOLLOW(X_p) = \varnothing$. Те ж саме повинне виконуватись, якщо X_q і X_r міняються ролями.
4. В жодному правилі типу 1 символ X_q або X_r не повинні бути «безвідмовним», іншими словами йому повинна відповідати процедура аналізуючої машини Кнута, яка хоча б в одному випадку повертає значення «false».

Визначення. Нетермінальний символ X_p є «безвідмовним» тоді і тільки тоді, коли відповідне йому правило:

1) або належить типу 5 ($X_p \to \varepsilon$);
2) або належить типу 3 і X_q є безвідмовним;
3) або належить типу 2 і X_q є безвідмовним;
4) або належить типу 1 і X_q або X_r є безвідмовним.

Перевірити умову 4 можна шляхом послідовного дослідження символів X_p на безвідмовність в порядку $X_1, X_2, \ldots, X_p$, тобто від меншого до більшого.

Теорема. Довільна LL(1) – мова може бути описана граматикою, всі правила якої належать одному з двох типів:

Тип 1. $A \to a_1 \beta_1 \mid a_2 \beta_2 \mid \ldots \mid a_m \beta_m$

Тип 2. $A \to a_1 \beta_1 \mid a_2 \beta_2 \mid \ldots \mid a_m \beta_m \mid \varepsilon$

де $a_1, a_2, \ldots a_m$ – різні термінальні символи; в правилах типу 2 жоден із символів a_i не належить множині FOLLOW(A); β_i – довільні сентенціальні форми.

Ще одне визначення LL(1)-граматики

Алгоритм 2 для побудови таблиці аналізу M може бути застосовано до любої граматики. Проте для деяких граматик M може мати неоднозначно визначені входи. Наприклад, якщо граматика ліворекурсивна чи неоднозначна, M буде мати хоча б один неоднозначно-визначений вхід. Граматики, для яких таблиці аналізу не мають неоднозначно-визначених входів, повинні бути LL(k)-граматиками, особливо LL(1)-граматиками.
Можна показати, що алгоритм 2 для кожної LL(1)-граматики G будує таблиці, за якими розпізнаються всі ланцюжки з L(G). LL(1)-граматики мають декілька відмінних властивостей. Неоднозначна чи ліворекурсивна граматика не може бути LL(1). Можна також показати, що граматика G являється LL(1) тоді і тільки тоді, коли для двох правил вигляду $A \to \alpha \mid \beta$ виконується наступне:

1) ні для якого терміналу a одночасно з α і β не виводяться рядки, які починаються з терміналу a;

2) тільки з одного із рядків α чи β може виводитися пустий рядок;

3) якщо $\beta \Rightarrow^* \varepsilon$, то з α не виводиться ніякий рядок, який починається з терміналу, що належать множині FOLLOW(A).

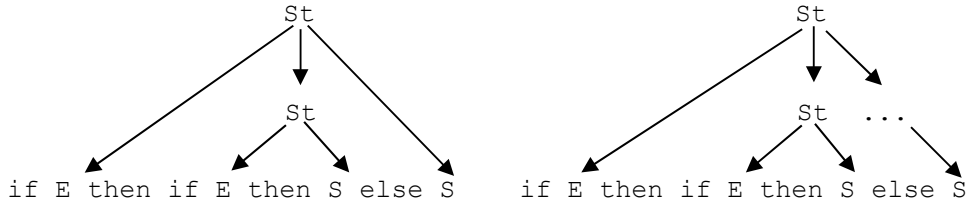
Еквівалентним є наступне визначення:
КВ-граматика називається LL(1)-граматикою, якщо з існування двох лівих виводів

$$(1)\ S \Rightarrow^* \gamma\ A\ \alpha \Rightarrow \gamma\ \beta\ \alpha \Rightarrow^* \gamma\ x,$$
$$(2)\ S \Rightarrow^* \gamma\ A\ \alpha \Rightarrow \gamma\ \delta\ \alpha \Rightarrow^* \gamma\ y,$$

для яких FIRST(x) = FIRST(y), випливає, що $\beta = \delta$. Це означає, що для даного ланцюжка $\gamma A \alpha$ і першого символу, який виводиться з $A\alpha$ (чи #), існує не більше одного правила, яке може бути застосовано до A, щоб отримати вивід деякого термінального ланцюжка, який починається з γ і продовжується цим першим символом. Мова, для якої можна побудувати LL(1)-граматику, називають LL(1)-мовою. Якщо таблиця аналізу має неоднозначно-визначені входи, то граматика не являє собою LL(1). Прикладом може служити наступна граматика:

```
St → if Ex then St
   | if Ex then St else St
   | Cont
Ex → ...
```

Ця граматика неоднозначна, що ілюструється на рис.3. Оскільки граматика неоднозначна, то вона не є LL(1)-граматикою. Проблема, чи породжує граматика LL-мову, не має алгоритмічного розв'язку.



а) б)

Рис.3.

Видалення лівої рекурсії

Основна складність при використанні передбачаючого аналізу – це написання такої граматики для вхідної мови, щоб за нею можна було б побудувати передбачаючий аналізатор. Іноді за допомогою деяких простих перетворень не LL(1)-граматику можна привести до LL(1)-вигляду. Серед цих перетворень найбільш очевидні являються ліва факторизація і видалення лівої рекурсії. Тут необхідно зробити два зауваження: по-перше, не всяка граматика після цих перетворень стає LL(1), по-друге, після видалення лівої рекурсії і лівої факторизації отримана граматика може стати важкою для розуміння. Граматика ліворекурсивна, якщо в ній міститься нетермінал A такий, що існує вивід $A \Rightarrow^+ Au$ для деякого рядка u. Ліворекурсивні граматики не можуть аналізуватися методами зверху-вниз, тому необхідно видалення лівої рекурсії.
Безпосередню ліву рекурсію (рекурсію вигляду $A \rightarrow Au$) можна видалити наступним чином. Спочатку групуємо A-правила:

```
A → Au1 | Au2 | ... | Aum | v1 | v2 | .... | vn
```

де ніякий із рядків vi не починається з A. Потім заміняємо A-правила на

```
A → v1A' | v2A' | .... | vnA'
A'→ u1A' | u2A' | .... | umA' | ε
```

Нетермінал A породжує ті ж рядки, що і раніше, але тепер немає лівої рекурсії. За допомогою цієї процедури видаляються всі безпосередні ліві рекурсії, але не видаляється ліва рекурсія, яка включає два чи більше кроки. Приведений нижче алгоритм 3 дозволяє видалити всі ліві рекурсії з граматики.

Алгоритм 3. Видалення лівої рекурсії.

Крок 1. Впорядковуємо нетермінали в довільному порядку.
Крок 2.
```
for i := 1 to n do
 for j := 1 to i-1 do
    нехай Aj → v1 | v2 | ... | vk - всі поточні правила для Aj;
    замінити всі правила вигляду Ai →Aju на правила
    Ai →v1u | v2u | ... | vkU;
 end;
видалити безпосередню ліву рекурсію в правилах для Ai;
end
```
Після (i-1)-ї ітерації зовнішнього циклу на кроці 2 для любого правила вигляду $Ak \rightarrow Alu$, де kk. В результаті на наступній ітерації (по i) внутрішній цикл (по j) послідовно збільшує нижню межу по m в любому правилі $Ai \rightarrow Amu$, поки не буде виконано $m >= i$. Потім, викидаючи безпосередню ліву рекурсію для Ai-правил, робимо m більше i.
Алгоритм 3 застосовується, якщо граматика не має циклів (виводів вигляду $A \Rightarrow + A$) і ε-правил (правил вигляду $A \rightarrow \varepsilon$). Як цикли, так і ε-правила можуть бути видалені попередньо. Отримувана граматика без лівої рекурсії може мати ε-правила.

Ліва факторизація

Основна ідея лівої факторизації в тому, коли незрозуміло яку з двох альтернатив потрібно використовувати для розгортки нетерміналу A, потрібно переробити A-правила так, щоб відкласти рішення до того часу, коли не буде достатньо інформації, щоб прийняти правильне рішення.

Якщо A → uv1 | uv2 – два A-правила і вхідний рядок починається з непустого рядка, який виводиться з u, ми не знаємо, чи розгортати по uv1 чи по uv2. Проте можна відкласти рішення, розгорнувши A → uA'. Тоді після аналізу того, що виводимо з u, можна розгорнути A' → v1 або A' → v2.
Лівофакторизовані правила приймають вигляд:

```
A → u A'
A' → v1 | v2
```

Алгоритм 4. Ліва факторизація граматики.

Для кожного нетерміналу A шукаємо самий довгий префікс u, спільний для двох чи більше його альтернатив. Якщо u # ε, тобто існує нетривіальний спільний префікс, замінюємо всі A-правила

```
A → uv1 | uv2 | ... | uvn | z,
```

де z – всі альтернативи, які не починаються з u,

на

```
A → uA' | z
A' → v1 | v2 | ... | vn
```

Тут A' – новий нетермінал. Повторно застосовуємо це перетворення, поки дві альтернативи не будуть мати спільного префікса.

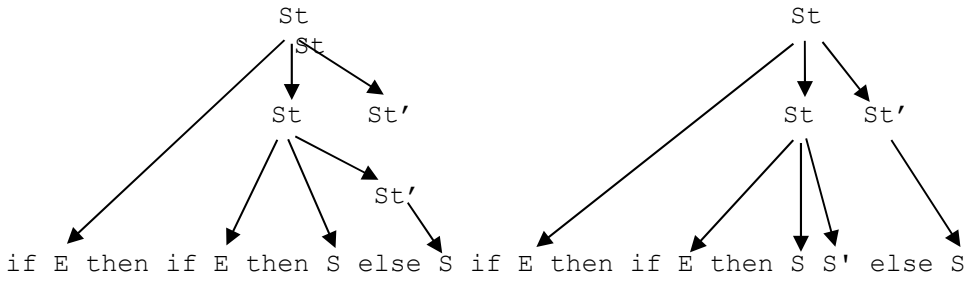
Пример 3. Розглянемо знову граматику умовних операторів:

```
St → if Ex then St
   | if Ex then St else St
   | Cont
Ex → ...
```

Після лівої факторизації граматика приймає вигляд

```
St  → if Ex then St St'
    | Cont
St' →  else St | ε
Ex  → ...
```

На жаль, граматика залишається неоднозначною, і значить не є LL(1), що ілюструється на рис.4.



а) б)

Рис.4.

Відновлення після синтаксичних помилок

В наведених програмах використовувалась процедура реакції на синтаксичні помилки error(). В найпростішому випадку ця процедура видає діагностику и завершує роботу аналізатора. Але можна спробувати деяким чином продовжити роботу. Для розбору зверху вниз можна запропонувати наступний простий алгоритм.

Якщо в момент виявлення помилки на верхівці магазину опинився нетермінальний символ N і для нього нема правила, яке відповідає вхідному символу, то скануємо вхід доти, доки не зустрінемо символ або з FIRST(N), або з FOLLOW(N). В першому разі розгортаємо N за відповідним правилом, в другому – видаляємо N з магазину.

Якщо на верхівці магазину термінальний символ, то можна викинути всі термінальні символи з верхівки магазину аж до першого (зверху) нетермінального символу і продовжувати так, як це було описано вище.