Лекція 11
Розробка синтаксичного аналізатора, що реалізує низхідну стратегію методом рекурсивного спуску

Метод рекурсивного спуску

Метод рекурсивного спуску реалізує безповоротний аналіз за рахунок обмежень на правила граматики:

1) правила не повинні містити лівобічної рекурсії. Якщо такі правила є, то вони замінюються правилами з правосторонньою рекурсією або представляються в ітераційній формі.
2) якщо є декілька правил з однаковою лівою частиною, то права частина повинна починатися з різних термінальних символів (нормальна форма Грейбах).

Суть методу: кожному нетерміналу граматики ставиться у відповідність окрема процедура або функція.

Приклад програмування синтаксичного аналізатора для наступної граматики:

1. <блок> → begin <список операторів> end
2. <список операторів> → <оператор>
3. <список операторів> → <оператор>;<список операторів>

} < список операторів > → <оператор> {; <оператор>}

4. <оператор> → <оператор присвоєння>
5. <оператор> → <оператор введення>
6. < оператор присвоєння > → <змінна>: =<вираз>
7. <вираз> → <терм>
8. <вираз> → <терм> + <вираз>

} <вираз> → <терм>{+ <терм>}

9. <терм> → <змінна>
10. <терм> → (<вираз>)
11. <оператор введення> → Read (<список введення>)
12. <список введення> → <змінна>
13. <список введення> → <змінна>,<список введення>

} <список введення> → <змінна>{, <змінна>}

14. <змінна> → i

Визначимо назви процедур, що відповідають нетерміналам граматики таким чином:

<блок> → BLK
<список операторів> → LST
<оператор> → STM
<оператор присвоєння> → ASN
<оператор введення> → GET
<вираз> → EXP
<терм> → TRM
<змінна> → VAR
<список введення> → SPW

Допоміжні процедури:

SCN - сканування лексем (код лексеми записується у змінну TS)
ERR - обробка помилок.

Примітка. У наведеній нижче програмі порівняння вигляду TS = ‘символи’ означають, що TS порівнюється з кодами лексем вказаних символів.

Структура синтаксичного аналізатора буде такою:

```
Program PARSER;
      <опис даних>;
      <опис процедур>;
begin
      <початкові установки>
      SCN; BLK
      <завершення аналізу>
end.
```

Розглянемо рекурсивні процедури, що реалізують синтаксичний аналізатор:

```
Procedure BLK;
  begin
       if TS <> 'begin' then ERR
                        else begin
                               SCN; LST
                             end
      if TS <> 'end' then ERR
 end;

Procedure LST;
    begin
      STM;
       while TS = ';' do begin
             SCN; STM
        end;
 end;

Procedure STM;
    begin
      if TS = 'Read' then begin SCN; GET end;
      else ASN;
 end;

Procedure ASN;
    begin
      VAR;
   if TS <> ': =' then ERR else begin SCN; EXP; end;
 end;

Procedure EXP;
    begin
      TRM;
       while TS = ' +' do begin
             SCN; TRM
        end;
 end;
```

```
Procedure TRM;
   begin
      if TS = '(' then begin
                          SCN; EXP;
                          if TS <> ')' then ERR ;
                         end
                  else VAR;
end;

Procedure GET;
   begin
      if TS <> '(' then ERR
                  else begin
                         SCN; SPW;
                         If TS <> ')' then ERR
                        end;
      SCN;
end;

Procedure SPW;
   begin
      VAR;
       while TS = ' , ' do begin
             SCN; VAR
        end;
 end;

Procedure VAR;
   begin
      if TS <> ' i ' then ERR else SCN;
 end;
```

Розробка синтаксичного аналізатора, що реалізує стратегію знизу вгору без повернення

Всі алгоритми аналізу без повернення будуються на основі спеціальних граматик з певними обмеженнями.

Розглянемо алгоритм висхідного аналізу на основі граматик простого передування.

Метод граматик простого передування розроблений Віртом і Вебером і є висхідним аналогом аналізуючої машини Кнута без повернень.

Відношення простого передування

Існує 3 види відношень передування:

а) $\ldots S_i \doteq S_j \ldots$

основа

б) $\ldots S_i \lessdot S_j \ldots$ S_j – голова основи

основа

в) $\ldots S_i \gtrdot S_j \ldots$ S_i – хвіст основи

основа

$S_i, S_j \in V = T \cup N$

Наприклад: $\ldots S_{i-1} \lessdot S_i \doteq S_{i+1} \doteq S_{i+2} \doteq S_{i+k} \gtrdot S_{i+k+1}$

основа

На кожному кроці аналізу шукаємо основу і замінюємо її відповідним нетерміналом, поки не отримаємо аксіому граматики.

Визначення граматики простого передування

Граматика G називається граматикою простого передування, якщо

1) між символами граматики існують наступні відношення передування:
   1) $S_i \doteq S_j$, якщо існує правило $U \to \alpha S_i S_j \beta$
   2) $S_i \lessdot S_j$, якщо існує правило $U \to \alpha S_i D \beta$ та виведення $D \Rightarrow S_j \delta$
   3) $S_i \gtrdot S_j$, якщо існує правило $U \to \alpha C S_j \beta$ і виведення $C \Rightarrow \delta S_i$
      або існує правило $U \to \alpha C D \beta$ і виведення $C \Rightarrow \delta S_i$, $D \Rightarrow S_j \gamma$,
      де $U, C, D \in N$, тобто нетермінальні символи граматики G,
      а $\alpha, \beta, \gamma, \delta \in V^*$ – рядки
2) не допускаються правила з однаковою правою частиною, а також наявність більше одного відношення передування між будь-якою парою символів.

Приклад

Задано граматику:
T = {x, a, (,) }
N = {<A>, <терм>, <вираз>}

Множина правил цієї граматики має наступний вигляд:

1. <A> → x<терм>x
2. <терм> → a
3. <терм> → (<вираз>
4. <вираз> → <терм>a)

При реалізації МП-автоматів синтаксичних аналізаторів використовуються обмежувальні символи, які ставляться ліворуч і праворуч від символів вхідного рядка T[1..n]. В якості такого символу візьмемо символ '#'.
Для обмежувальних символів T[0]='#' та T[n+1]='#' мають місце наступні відношення:

- T[0] ⋖ 'будь-який символ рядка'
- 'будь-який символ рядка' ⋗ T[n+1].

Побудуємо матрицю відношень передування:

S_i \ S_j

	x	a	(	)	<A>	<терм>	<вираз>
x		⋖	⋖			≐	
a	⋗	⋗		≐			
(		⋖	⋖			⋖	≐
)	⋗	⋗					
<A>							
<терм>	≐	≐					
<вираз>	⋗	⋗					

Для автоматичної побудови матриці передування доцільно спочатку побудувати таблицю лівих і правих символів для нетерміналів граматики:

	L(u)	R(u)
<A>	x	x
<терм>	a, (	a, <вираз>,)
<вираз>	<терм>, a, (	)

Кодування символів передування може бути, наприклад, таке:

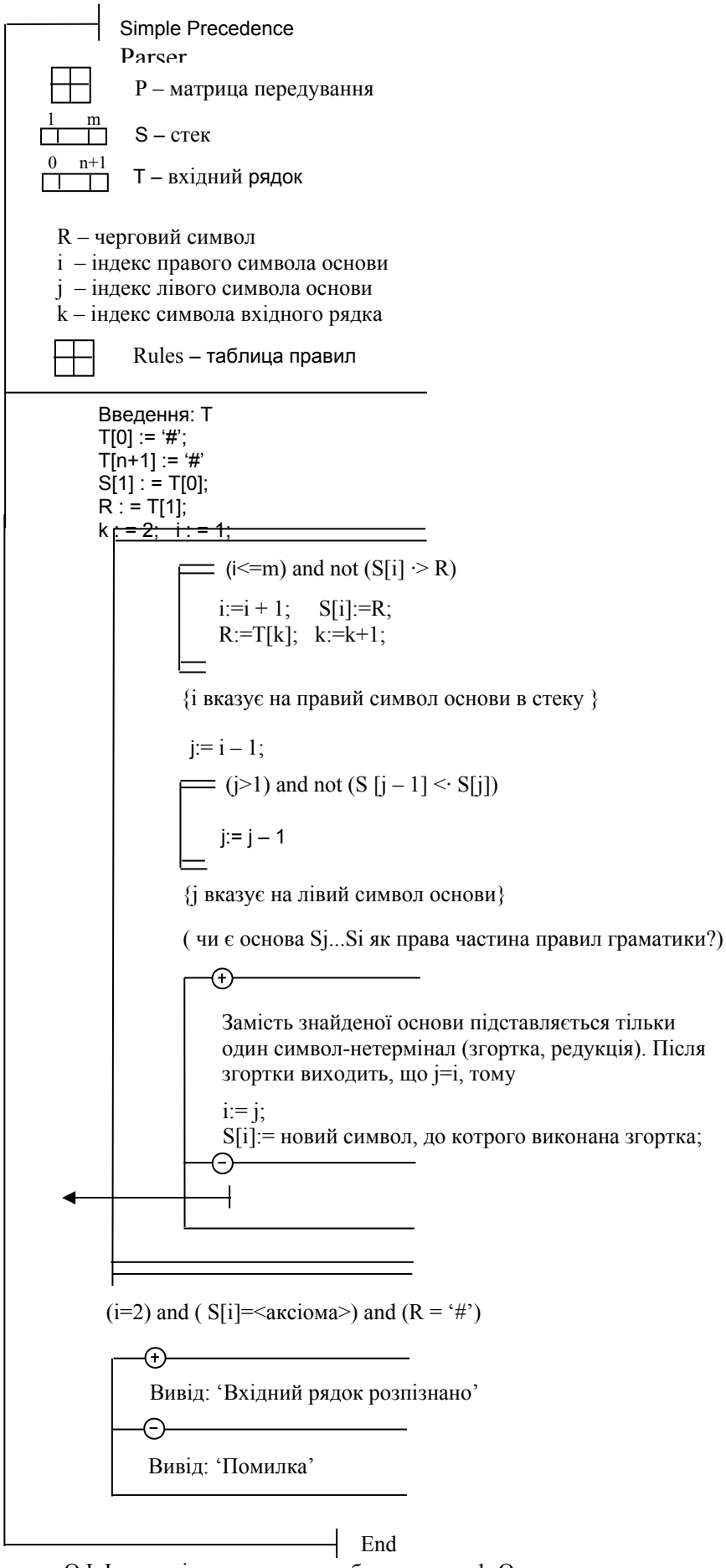
⫤ – 255		⫤ – 0	
⩵ – 0	або	⋖ – 1	
⋖ – –1		⩵ – 2	
⋗ – +1		⋗ – 3	

Розглянемо алгоритм синтаксичного аналізатора.

Для його реалізації потрібні наступні дані:

- таблиця правил;
- матриця відносин передування P;
- стек S (наприклад, у вигляді вектора);
- вхідний рядок T (вектор лексем);
- к – поточний індекс вектора T (T[k] - елемент вхідного рядка);
- i – індекс правого символу основи в стеку S;
- j – індекс лівого символу основи в стеку S;
- черговий символ R.

У цих позначеннях основа завжди матиме вигляд $\underbrace{S_j \ \ldots \ S_i}$



№ кроку	S							Відношення	R + залишок вхідного рядка T	
	1	2	3	4	5	6	7		R	залишок
0	#							<·	x	(a a) x #
1	#	x						<·	(	a a) x #
2	#	x	(					<·	a	a) x #
3	#	x	(	a				·>	a	) x #
4	#	x	(	<терм>				≐	a	) x #
5	#	x	(	<терм>	a			≐	)	x #
6	#	x	(	<терм>	a	)		·>	x	#
7	#	x	(	<вираз>				·>	x	#
8	#	x	<терм>					≐	x	#
9	#	x	<терм>	x				·>	#	
10	#	<a>							#	

 # x (a a) x #