Синтаксический анализатор

Синтаксический анализатор – часть компилятора, отвечающая за выявление и проверку синтаксических конструкций входного языка

Функции синтаксического анализатора

- Выделение синтаксических конструкций в тексте исходной программы
- Проверка правильности каждой синтаксической конструкции
- Представление синтаксических конструкций в виде, удобном для дальнейшей генерации текста результирующей программы
- Сообщения о выявленных ошибках

Обработка ошибок

- *Лексические* ошибки включают неверно записанные идентификаторы, ключевые слова или операторы, или отсутствие кавычек вокруг текста, являющегося строкой

- *Синтаксические* ошибки включают неверно поставленные точки с запятой или лишние или недостающие фигурные скобки

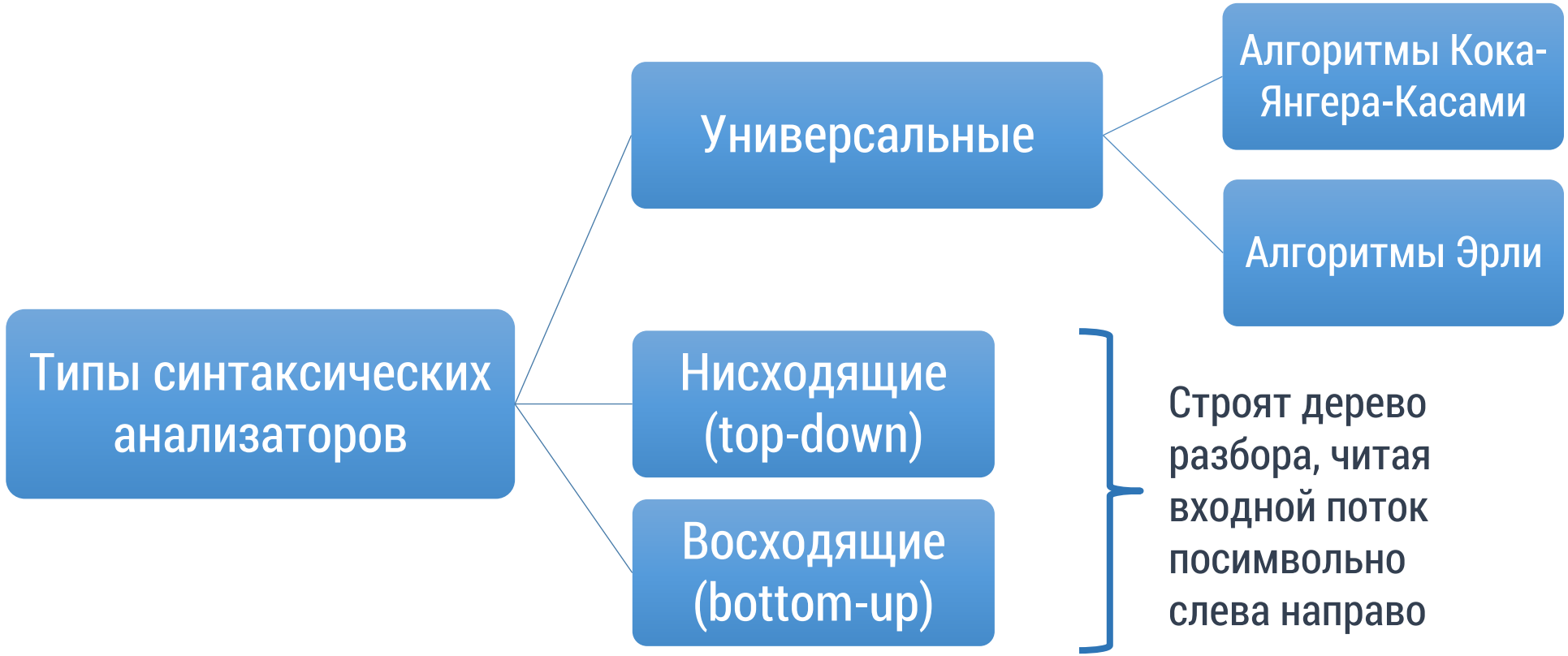
- *Семантические* ошибки включают несоответствие типов операторов и их операндов

- *Логические* ошибки могут быть любыми — от неверных решений программиста до использования, например, в программе на языке Си оператора присваивания = вместо оператора сравнения ==

Стратегии восстановления после ошибок

- Восстановление в режиме паники
- Восстановление на уровне фразы
- Правила ошибок
- Глобальная коррекция

Типы синтаксических анализаторов



Контекстно-свободные грамматики

$A \to \beta$, где $A \in$ **VN**, $\beta \in$ **V**$^+$ (неукорачивающие)

$A \to \beta$, где $A \in$ **VN**, $\beta \in$ **V*** (укорачивающие)

E → E **+** T | E**-**T | T

T → T**∗**F | T**/**F | F

F → **(**E**)** | **id**

stmt → **if** (expr) stmt **else** stmt

Почему КС-грамматики?

$$[a\ldots z]\,([a\ldots z]\,|\,[0\ldots 9])^*\quad ((+\,|\,-\,|\times|\div)\ \ [a\ldots z]\,([a\ldots z]\,|\,[0\ldots 9])^*\,)^*$$

```
a + b × c                fee ÷ fie × foe
```

Приоритет операторов?

$$(\underline{(}\,|\epsilon)\ [a\ldots z]\,([a\ldots z]\,|\,[0\ldots 9])^*$$
$$((+\,|\,-\,|\times|\div)\ \ [a\ldots z]\,([a\ldots z]\,|\,[0\ldots 9])^*\ (\underline{)}\,|\epsilon)\ )^*$$

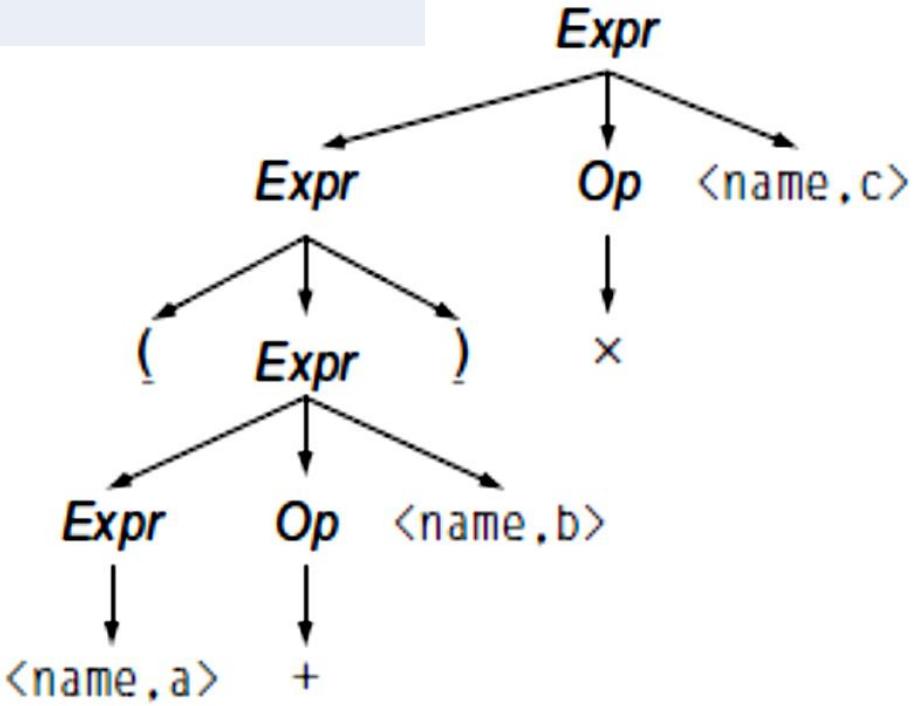
```
( a + b ) × c   a + ( b × c   a + b ) × c )
```

Почему КС-грамматики?

1	Expr → (Expr) |
2	Expr Op name |
3	name
4	Op → +
5	-
6	×
7	÷

Правило	Сентенциальная форма
	Expr
2	Expr Op name
6	Expr × name
1	(Expr) × name
2	(Expr Op name) × name
4	(Expr + name) × name
3	(name + name) × name

Правосторонний вывод для (a + b) × c



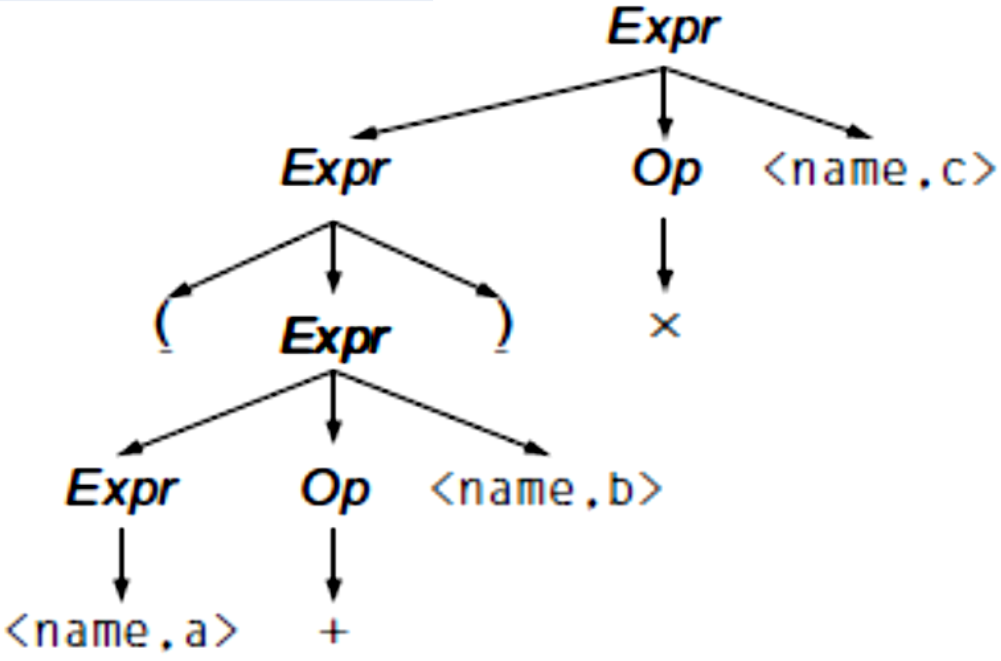
Синтаксическое дерево разбора

Почему КС-грамматики?

1	Expr → (Expr) |
2	Expr Op name |
3	name
4	Op → +
5	-
6	×
7	÷

Правило	Сентенциальная форма
	Expr
2	Expr Op name
1	(Expr) Op name
2	(Expr Op name) Op name
3	(name Op name) Op name
4	(name + name) Op name
6	(name + name) × name

Левосторонний вывод для (a + b) × c

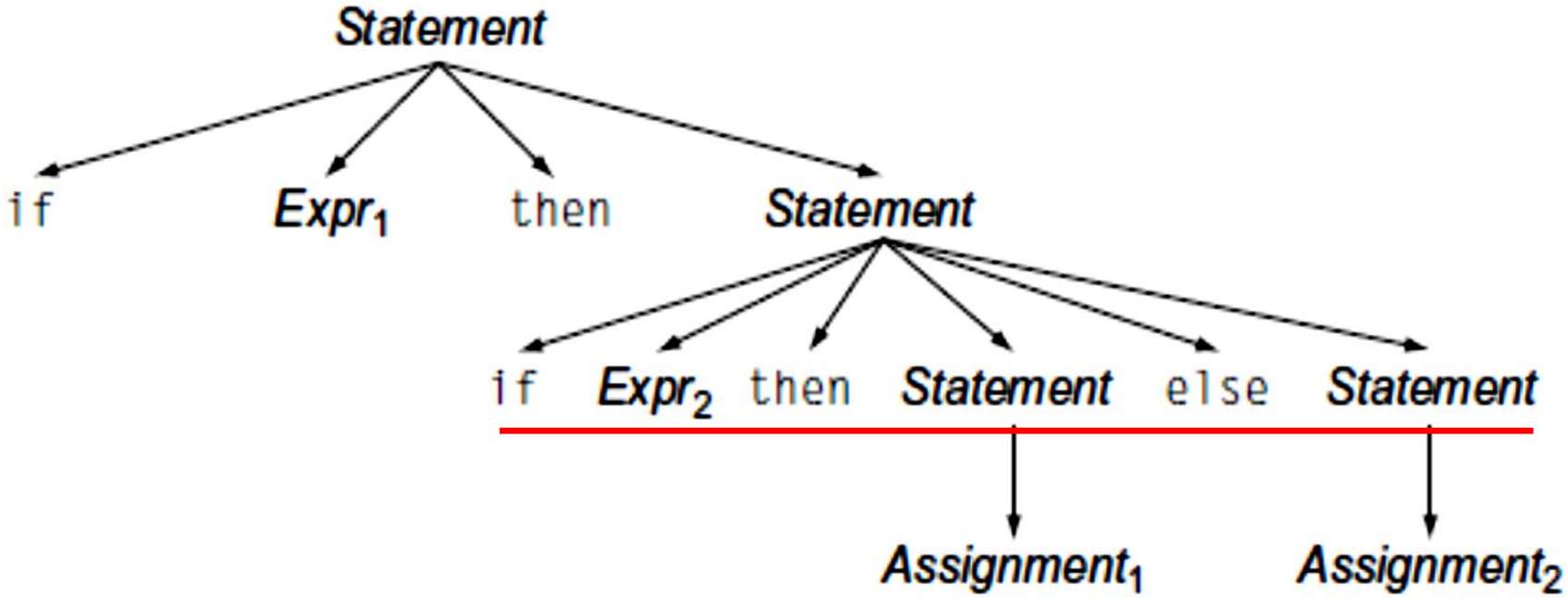


Синтаксическое дерево разбора

Неоднозначные КС-грамматики

1	Statement → if Expr then Statement else Statement |
2	if Expr then Statement |
3	Assignment |
4	... other statements ...

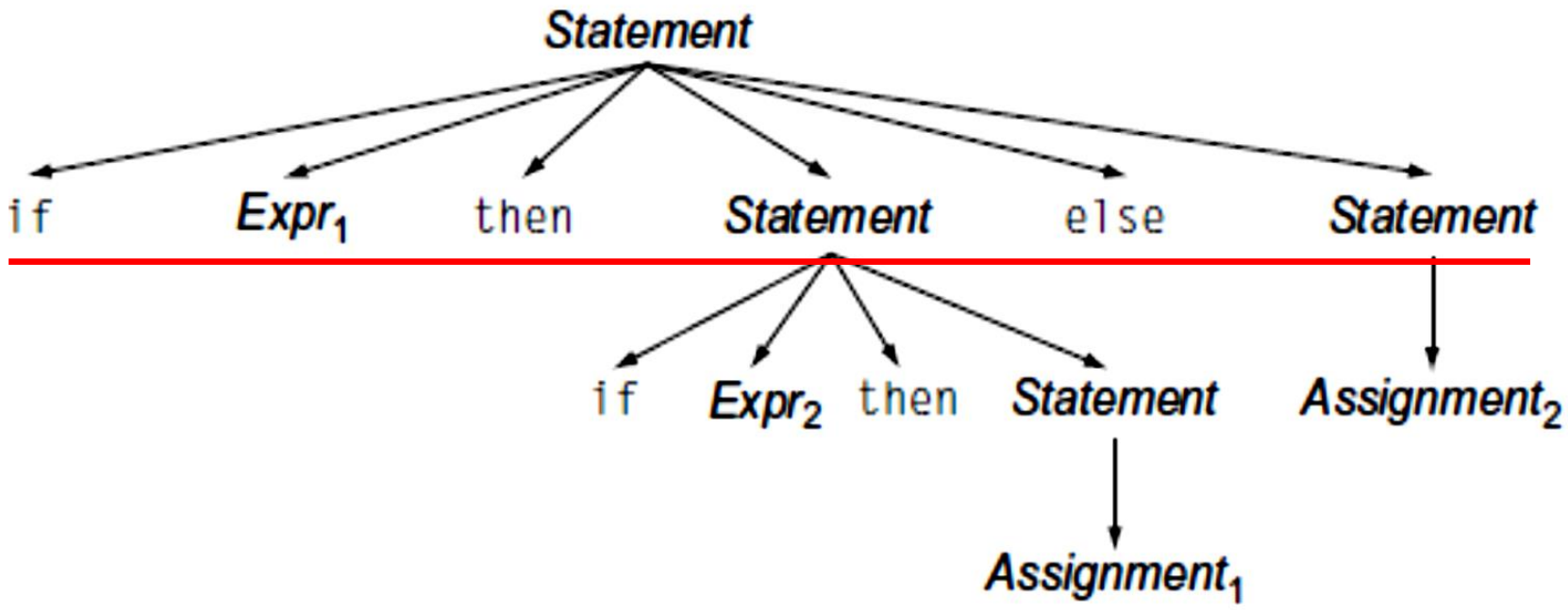
`if` *Expr*1 `then` `if` *Expr*2 `then` *Assignment*1 `else` *Assignment*2



Неоднозначные КС-грамматики

1	Statement → if Expr then Statement else Statement |
2	if Expr then Statement |
3	Assignment |
4	... other statements ...

`if` *Expr*1 `then` `if` *Expr*2 `then` *Assignment*1 `else` *Assignment*2



Неоднозначные КС-грамматики

1	Statement → if Expr then Statement |
2	if Expr then WithElse else Statement |
3	Assignment
4	WithElse → if Expr then WithElse else WithElse |
5	Assignment

Правило	Сентенциальная форма
	Statement
1	if Expr then Statement
2	if Expr then if Expr then WithElse else Statement
3	if Expr then if Expr then WithElse else Assignment
5	if Expr then if Expr then Assignment else Assignment

Задача синтаксического анализа

Построение синтаксического дерева разбора

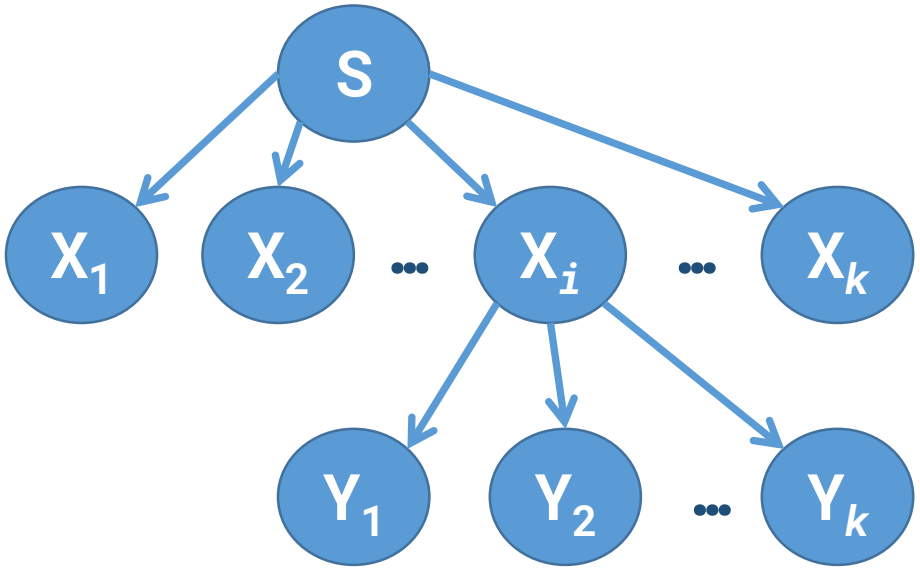
Нисходящий синтаксический анализ	Восходящий синтаксический анализ

Нисходящий синтаксический анализ

При нисходящем анализе входная цепочка символов просматривается слева направо и порождается левосторонний вывод

Метод рекурсивного спуска

$$S \rightarrow X_1 X_2 \ldots X_{i-1} X_i X_{i+1} \ldots X_k$$
$$X_i \rightarrow Y_1 Y_2 \ \ldots \ Y_l$$



Псевдокод разбора для нетерминала

```
void S()
{
   Выбираем S-правило S → X1X2…Xk;
   for (i от 1 до k)
  {
     if (Xi – нетерминал)
       Вызов процедуры Xi();
     else if (Xi равно текущему входному символу)
       Переходим к следующему входному символу;
     else /* Обнаружена ошибка */;
   }
}
```

Условия применимости метода рекурсивного спуска

Два варианта правил в КС-грамматике:

1. $A \rightarrow \gamma, \quad \gamma \in (\mathbf{VT} \cup \mathbf{VN})^*$ и это единственное правило для A
2. $A \rightarrow a_1\beta_1 \mid a_2\beta_2 \mid \ \ldots \ \mid a_n\beta_n, a_i \in \mathbf{VT}, \beta_i \in (\mathbf{VT} \cup \mathbf{VN})^*, a_i \neq a_j$

Рекурсия

Символ $A \in$ **VN** называется рекурсивным, если для него существует цепочка вывода вида

$A \Rightarrow^{+} \alpha A \beta$, где $\alpha, \beta \in$ (**VT** $\cup$ **VN**)*

Если $\alpha = \varepsilon$ и $\beta \neq \varepsilon$, то рекурсия называется *левой*.

Если $\alpha \neq \varepsilon$ и $\beta = \varepsilon$, то рекурсия называется *правой*.

Если $\alpha = \varepsilon$ и $\beta = \varepsilon$, то рекурсия представляет собой *цикл*.

Левая рекурсия

Правило	Сентенциальная форма	Входная цепочка символов
	Expr	↑name + name × name
1	Expr + Term	↑name + name × name
1	Expr + Term + Term	↑name + name × name
1	…	↑name + name × name

0	Goal → Expr
1	Expr → Expr + Term |
2	Expr - Term |
3	Term
4	Term → Term × Factor |
5	Term ÷ Factor |
6	Factor
7	Factor → (Expr) |
8	num |
9	name

Алгоритм устранения левой рекурсии

1. **VN** = $\{A_1, A_2, \ldots, A_n\}$, $i = 1$
2. Правила для A_i.

   Если в правилах нет левой рекурсии, то они записываются в **P'** без изменений, а A_i добавляется в **VN'**.

   Иначе правила для A_i записываются в виде
   $A_i \to A_i\alpha_1 \mid A_i\alpha_2 \mid \ldots \mid A_i\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_p$, где ни одна из цепочек β_j не начинается с A_k таких, что $k \leq i$.

   Вместо этих правил в **P'** записываются правила вида:

$$A_i \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_p \mid \beta_1 A'_i \mid \beta_2 A'_i \mid \ldots \mid \beta_p A'_i$$

$$A'_i \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_m \mid \alpha_1 A'_i \mid \alpha_2 A'_i \mid \ldots \mid \alpha_m A'_i$$

A_i и A'_i включаются в **VN**.

3. Если $i = n$, то **G'** построена, перейти к 6, иначе $i = i + 1$, $j = 1$, перейти к 4

Алгоритм устранения левой рекурсии

4. Для A_i в **P'** заменить все правила вида $A_i \to A_j\alpha$, где
   $\alpha \in$ (**VT** $\cup$ **VN**)*, на правила вида
   $A_i \to \beta_1\alpha \mid \beta_2\alpha \mid \ldots \mid \beta_m\alpha$, причем
   $A_j \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$ – все правила для A_j

5. Если $j = i - 1$, перейти к 2, иначе $j = j + 1$ и перейти к 4

6. Целевым символом грамматики **G'** становится A_k, соответствующий символу S исходной грамматики

Пример

G({+, −, /, *, a, b}, {S, T, E}, **P**, S)

P: $S \to S + T \mid S - T \mid T$

$T \to T * E \mid T / E \mid E$

$E \to (S) \mid a \mid b$

VN' = $\{A_1, A'_1, A_2, A'_2, A_3\}$

1. VN' = $\{A_1, A_2, A_3\}$, i = 1

$A_1 \to A_1 + A_2 \mid A_1 - A_2 \mid A_2$

$A_2 \to A_2 * A_3 \mid A_2 / A_3 \mid A_3$

$A_3 \to (A_1) \mid a \mid b$

2. $A_1 \to A_1 + A_2 \mid A_1 - A_2 \mid A_2$

$A_1 \to A_1\alpha_1 \mid A_1\alpha_2 \mid \beta_1$

$\alpha_1 = +A_2,\ \alpha_2 = -A_2,\ \beta_1 = A_2$

$A_1 \to A_2 \mid A_2A'_1$

$A'_1 \to + A_2 \mid - A_2 \mid + A_2A'_1 \mid - A_2A'_1$

3. $A_2 \to A_2 * A_3 \mid A_2 / A_3 \mid A_3$

$A_2 \to A_2\alpha_1 \mid A_2\alpha_2 \mid \beta_1$

$\alpha_1 = *A_3,\ \alpha_2 = /A_3,\ \beta_1 = A_3$

$A_2 \to A_3 \mid A_3A'_2$

$A'_2 \to * A_3 \mid / A_3 \mid * A_3A'_2 \mid / A_3A'_2$

Пример (продолжение)

4. $A_3 \to (A_1) \mid a \mid b$ – нет левой рекурсии

G($\{+, -, /, *, a, b\}$, $\{A_1, A'_1, A_2, A'_2, A_3\}$, **P**, A_1)

P: $A_1 \to A_2 \mid A_2A'_1$

$A'_1 \to + A_2 \mid - A_2 \mid + A_2A'_1 \mid - A_2A'_1$

$A_2 \to A_3 \mid A_3A'_2$

$A'_2 \to * A_3 \mid / A_3 \mid * A_3A'_2 \mid / A_3A'_2$

$A_3 \to (A_1) \mid a \mid b$

Устранение левой рекурсии

До устранения	После устранения
Expr → Expr + Term | Expr - Term | Term	Expr → Term Expr’ Expr’ → + Term Expr’ | - Term Expr’ | ε
Term → Term × Factor | Term ÷ Factor | Factor	Term → Factor Term’ Term’ → × Factor Term’ | ÷ Factor Term’ | ε

0	Goal → Expr	6	Term’ → × Factor Term’ |
1	Expr → Term Expr’	7	÷ Factor Term’ |
2	Expr’ → + Term Expr’ |	8	ε
3	- Term Expr’ |	9	Factor → (Expr) |
4	ε	10	num |
5	Term → Factor Term’	11	name

Правило	Сентенциальная форма	Входная цепочка символов
	Expr	name + name × name
1	Term Expr’	name + name × name
5	Factor Term’ Expr’	name + name × name
11	name Term’ Expr’	name + name × name
→	name Term’ Expr’	name + name × name
8	name Expr’	name + name × name
2	name + Term Expr’	name + name × name
→	name + Term Expr’	name + name × name
5	name + Factor Term’ Expr’	name + name × name
11	name + name Term’ Expr’	name + name × name
→	name + name Term’ Expr’	name + name × name
6	name + name × Factor Term’ Expr’	name + name × name
→	name + name × Factor Term’ Expr’	name + name × name
11	name + name × name Term’ Expr’	name + name × name
→	name + name × name Term’ Expr’	name + name × name
8	name + name × name Expr’	name + name × name
4	name + name × name	name + name × name

Левая факторизация

Это преобразование позволяет исключить из КС-грамматики правила, имеющие в своих правых частях одинаковые префиксы

Stmt → **if** Expr **then** Stmt **else** Stmt | **if** Expr **then** Stmt

$$A \to \alpha\beta_1 \mid \alpha\beta_2$$

$$\Downarrow$$

$$A \to \alpha A'$$

$$A' \to \beta_1 \mid \beta_2$$

Алгоритм левой факторизации

Вход: грамматика G

Выход: эквивалентная левофакторизованная грамматика

Метод: для каждого нетерминала A находим самый длинный префикс α, общий для двух или большего числа альтернатив. Заменим все правила $A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \gamma$, где γ представляет все альтернативы, не начинающиеся с α, правилами

$$A \to \alpha A' \mid \gamma$$

$$A' \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

Выполнять преобразование до тех пор, пока в грамматике остаются правила с одинаковыми префиксами

Пример

Stmt → **if** Expr **then** Stmt | **if** Expr **then** Stmt **else** Stmt | **a**

Expr → **b**

Stmt → **if** Expr **then** Stmt Stmt' | **a**

Stmt' → **else** Stmt | ε

Expr → **b**

Пример реализации метода рекурсивного спуска

$S \rightarrow aA \mid bB$

$A \rightarrow a \mid bA \mid cC$

$B \rightarrow b \mid aB \mid cC$

$C \rightarrow AaBb$

Грамматика удовлетворяет условию применимости метода рекурсивного спуска

```
int main()
{
   ...
   if(S()) return 0;
   else return -1;
}
int S()
{
   c = gc();
   if(c == 'a'){
       if(!A()) return 0;
   }else if(c == 'b'){
       if(!B()) return 0;
   }else{
       return 0;
   }
   return 1;
}
```

Пример реализации метода рекурсивного спуска

$S \to aA \mid bB$

$A \to a \mid bA \mid cC$

$B \to b \mid aB \mid cC$

$C \to AaBb$

Грамматика удовлетворяет условию применимости метода рекурсивного спуска

```
int A()
{
    c = gc();
    if(c == 'b')
    {
      if(!A()) return 0;
    }else if(c == 'c')
    {
      if(!C()) return 0;
    }else if(c == 'a')
    {
      return 1;
    }else{
      return 0;
    }
    return 1;
}
```

Пример реализации метода рекурсивного спуска

$S \to aA \mid bB$

$A \to a \mid bA \mid cC$

$B \to b \mid aB \mid cC$

$C \to AaBb$

Грамматика удовлетворяет условию применимости метода рекурсивного спуска

```
int B()
{
    c = gc();
    if(c == 'a')
    {
      if(!B()) return 0;
    }else if(c == 'c')
    {
      if(!C()) return 0;
    }else if(c == 'b')
    {
      return 1;
    }else{
      return 0;
    }
    return 1;
}
```

Пример реализации метода рекурсивного спуска

$S \to aA \mid bB$

$A \to a \mid bA \mid cC$

$B \to b \mid aB \mid cC$

$C \to AaBb$

Грамматика удовлетворяет условию применимости метода рекурсивного спуска

```
int C()
{
    if(!A()) return 0;
    c = gc();
    if(c == 'a')
    {
      if(!B()) return 0;
      c = gc();
      if(c != 'b')
      {
            return 0;
      }
    }else{
      return 0;
    }
    return 1;
}
```

Пример реализации метода рекурсивного спуска

$S \to S + T \mid S - T \mid T$

$T \to T * E \mid T / E \mid E$

$E \to (S) \mid a \mid b$

$S \to T\{+ T \mid - T\}$

$T \to E\{* E \mid / E\}$

$E \to (S) \mid a \mid b$

```
int S()
{
  if(!T()) return 0;
  while(gc() == '+' ||
        gc() == '-')
  {
     if(!T()) return 0;
  }
  return 1;
}
```

Задания для самостоятельной работы

$S \to P = E \mid \textbf{while}\ E\ \textbf{do}\ S$

$P \to I \mid I\ (E\ \{,\ E\})$

$E \to E + T \mid T$

$T \to T * F \mid F$

$F \to P \mid (E)$

$I \to a \mid b \mid c$

$F \to \textbf{function}\ I(I)\ S;\ I := E\ \textbf{end}$

$S \to ;\ I := ES \mid \varepsilon$

$E \to E * I \mid E + I \mid I$

Задания для самостоятельной работы

Восстановить КС-грамматику по функциям, реализующим синтаксический анализ методом рекурсивного спуска.

```
void S()
{
    A();
    if(c != ‘⊥’)
        ERROR();
}
void A()
{
    B();
    while (c == ‘a’)
    {
        c = fgetc(fp);
        B();
    }
    B();
}
```

```
void B()
{
    if(c == ‘b’)
        c = fgetc(fp);
}
int main()
{
    fp = fopen(“data”, “r”);
    c = fgetc(fp);
    S();
    printf(“O.K.!”);
    return 0;
}
```