

Синтаксический анализ

Задачи синтаксического анализа

- установить, имеет ли цепочка лексем структуру, заданную синтаксисом языка, т.е. решить задачу разбора по заданной грамматике,
- зафиксировать эту структуру.

Для описания синтаксиса языков программирования используются КС-грамматики, правила которых имеют вид

$$A \rightarrow \alpha, \quad \text{где } A \in N, \quad \alpha \in (T \cup N)^*.$$

Для разных подклассов КС-грамматик существуют достаточно эффективные алгоритмы разбора.

Некоторые общие алгоритмы анализа по КС-грамматикам:

- синтаксический анализ с возвратами
(время работы **экспоненциально** зависит от длины цепочки);
- алгоритм Кока-Янгера-Касами
(для разбора цепочек длины n требуется время cn^3);
- алгоритм Эрли (для разбора цепочек длины n требуется время cn^3).

Эти (и подобные им по времени работы) алгоритмы практически неприемлемы.

Алгоритмы анализа, расходующие на обработку входной цепочки линейное время, **применимы** только к некоторым **подклассам** КС-грамматик.

Применимость синтаксических анализаторов

Каждый метод синтаксического анализа основан на своей технике построения дерева вывода и предполагает свой способ построения по грамматике программы-анализатора, которая будет осуществлять разбор цепочек.

Корректный анализатор завершает свою работу для любой входной цепочки и выдает верный ответ о принадлежности цепочки языку.

Анализатор *некорректен*, если:

- не распознает хотя бы одну цепочку, принадлежащую языку;
- распознает хотя бы одну цепочку, языку не принадлежащую;
- закликивается на какой-либо цепочке.

Метод анализа *применим* к данной грамматике, если анализатор, построенный в соответствии с этим методом, **корректен** и строит все возможные выводы цепочек в данной грамматике.

Метод рекурсивного спуска (РС-метод)

Пусть дана грамматика $G_{pc} = (\{ a,b,c, \perp \}, \{ S,A,B \}, P, S)$, где

$$\begin{aligned} P: \quad S &\rightarrow AB\perp & L(G) &= \{ c^n abc^m a\perp \mid n,m \geq 0 \} \\ A &\rightarrow a \mid cA \\ B &\rightarrow bA \end{aligned}$$

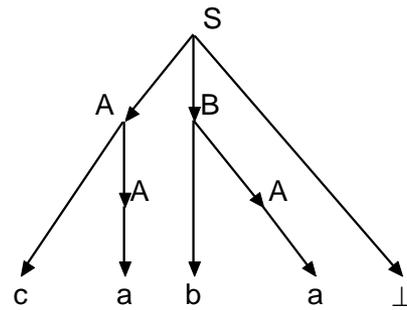
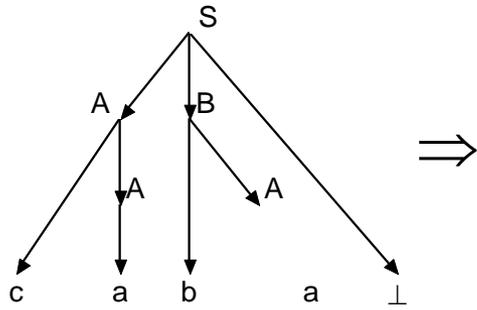
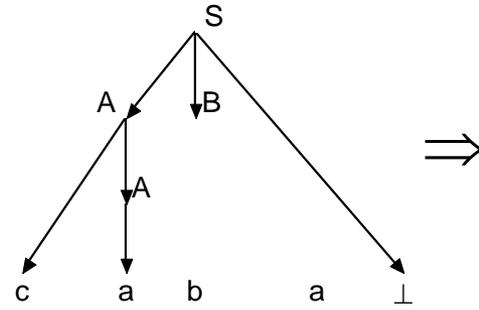
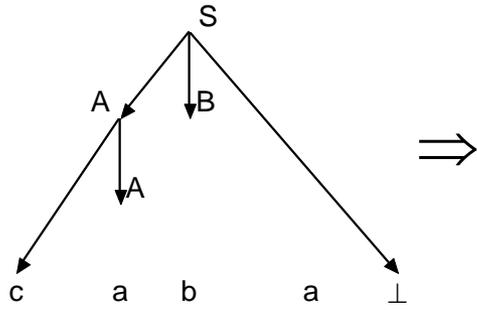
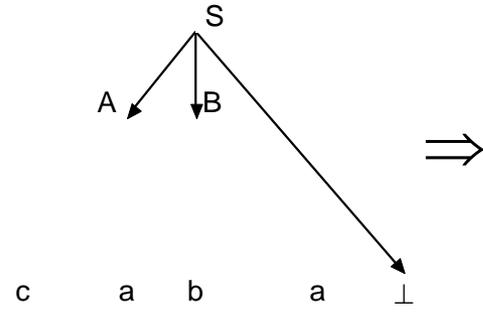
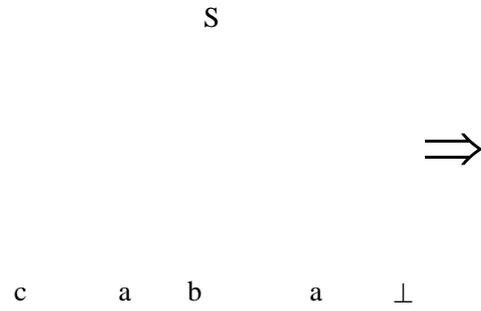
и надо определить, принадлежит ли цепочка **caba** языку $L(G)$.

Построим вывод этой цепочки:

$$S \rightarrow AB\perp \rightarrow cAB\perp \rightarrow caB\perp \rightarrow cabA\perp \rightarrow caba\perp$$

Следовательно, цепочка принадлежит языку $L(G)$.

Последовательность применений правил вывода эквивалентна построению дерева разбора методом "сверху вниз", а метод рекурсивного спуска фактически реализует этот способ разбора.



Метод рекурсивного спуска (РС-метод)

- Для **каждого нетерминала** грамматики создается своя процедура, носящая **его имя**; задача этой процедуры - начиная с указанного места исходной цепочки, найти подцепочку, которая выводится из этого нетерминала.
- Если такую подцепочку считать не удастся, то процедура завершает свою работу, сигнализируя об ошибке, что означает, что цепочка не принадлежит языку, и останавливая разбор.
- Если подцепочку удалось найти, то работа процедуры считается нормально завершённой и осуществляется возврат в точку вызова.
- Тело каждой такой процедуры пишется непосредственно по правилам вывода из соответствующего нетерминала.
- Текущая анализируемая лексема входной цепочки должны быть уже прочитана и доступна в процедуре, именно по ней осуществляется выбор нужной альтернативы.
- Для каждой альтернативы из правой части правила вывода осуществляется поиск подцепочки, выводимой из этой альтернативы. При этом:
 - терминалы проверяются в самой процедуре, и в случае удачной проверки считывается очередная лексема;
 - нетерминалам соответствуют вызовы процедур, носящих их имена.

Метод рекурсивного спуска

Работа системы процедур, построенных в соответствии с РС-методом, начинается с главной функции *main* (), которая:

- считывает первый символ исходной цепочки (заданной во входном потоке *stdin*),
- затем вызывает процедуру *S* (), которая проверяет, выводится ли входная цепочка из начального символа *S* (в общем случае это делается с участием других процедур, которые, в свою очередь, рекурсивно могут вызывать и саму *S* () для анализа фрагментов исходной цепочки).

Считаем, что в конце любой анализируемой цепочки всегда присутствует символ \perp (признак конца цепочки). На практике этим признаком может быть ситуация «конец файла» или маркер «конец строки».

В задачу *main* () входит также распознавание символа \perp .

Можно считать, что *main* () соответствует добавленному в грамматику правилу $M \rightarrow S\perp$, где *M* — новый начальный символ.

Реализация РС-метода для грамматики G_{pc}.

```
int c;
void A ();
void B ();
void gc () {
    cin >> c;
}

void S () {
    A ();
    B ();
    if (c != '⊥') throw c;
}

int main () {
    try { gc ();
        S ();
        cout << "SUCCESS !!!" << endl;
        return 0;
    }
    catch ( int c ) {
        cout << "ERROR on lexeme" << c << endl;
        return 1;
    }
}
```

```
void A () {
    if (c == 'a') gc ();
    else
        if (c == 'c') { gc (); A (); }
        else throw c;
}

void B () {
    if (c == 'b') { gc (); A (); }
    else throw c;
}
```

Достаточное условие применимости РС-метода

Метод рекурсивного спуска применим к КС-грамматике, если каждая ее группа правил вывода из A имеет один из следующих видов:

1. либо $A \rightarrow \alpha$, где $\alpha \in (T \cup N)^*$ и это единственное правило вывода для этого нетерминала;
2. либо $A \rightarrow a_1\alpha_1 \mid a_2\alpha_2 \mid \dots \mid a_n\alpha_n$,
где $a_i \in T$ для всех $i = 1, 2, \dots, n$; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (T \cup N)^*$,
3. либо $A \rightarrow a_1\alpha_1 \mid a_2\alpha_2 \mid \dots \mid a_n\alpha_n \mid \varepsilon$,
где $a_i \in T$ для всех $i = 1, 2, \dots, n$; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (T \cup N)^*$,
и $\mathbf{first(A)} \cap \mathbf{follow(A)} = \emptyset$.

Множество $\mathbf{first(A)}$ - это множество терминальных символов, которыми начинаются цепочки, выводимые из A в грамматике $G = (T, N, P, S)$:

$$\mathbf{first(A)} = \{ a \in T \mid A \Rightarrow a\alpha', \text{ где } A \in N, \alpha' \in (T \cup N)^* \}.$$

Множество $\mathbf{follow(A)}$ - это множество терминальных символов, которые следуют за цепочками, выводимыми из A в грамматике $G = (T, N, P, S)$,

$$\mathbf{follow(A)} = \{ a \in T \mid S \Rightarrow \alpha A \beta, \beta \Rightarrow a \gamma, A \in N, \alpha, \beta, \gamma \in (T \cup N)^* \}.$$

Если все правила вывода заданной КС-грамматики G удовлетворяют указанному условию, то РС-метод к ней применим, при этом грамматика G называется грамматикой **канонического вида** для РС-метода.

О применимости РС-метода

Сформулированное выше условие не является необходимым.

Метод рекурсивного спуска представляет собой одну из возможных реализаций нисходящего анализа с прогнозируемым выбором альтернатив.

Прогнозируемый выбор означает, что **по грамматике** можно заранее **предсказать**, какую альтернативу нужно будет выбрать на очередном шаге вывода в соответствии с текущим символом (т.е. первым символом из еще не прочитанной части входной цепочки).

РС-метод **неприменим**, если такой выбор **неоднозначен**.

Например, для грамматики

$$\begin{aligned} G: \quad & S \rightarrow A \mid B \\ & A \rightarrow aA \mid d \\ & B \rightarrow aB \mid b \end{aligned}$$

РС-метод неприменим, поскольку, если первый прочитанный символ есть 'а', то выбор альтернативы правила вывода из S неоднозначен.

РС-метод неприменим к неоднозначным грамматикам, так как на каком-либо шаге анализа выбор альтернативы вывода обязательно будет неоднозначным.

Критерий применимости РС-метода

Пусть G — КС-грамматика.

РС-метод применим к G , тогда и только тогда, когда для любой пары альтернатив $X \rightarrow \alpha \mid \beta$ выполняются следующие условия:

1. $\text{first}(\alpha) \cap \text{first}(\beta) = \emptyset$;
2. справедливо не более чем одно из двух соотношений:
 $\alpha \Rightarrow \varepsilon, \beta \Rightarrow \varepsilon$;
3. если $\beta \Rightarrow \varepsilon$, то $\text{first}(\alpha) \cap \text{follow}(X) = \emptyset$.

Множество **first** (α) — это множество терминальных символов, которыми начинаются цепочки, выводимые из цепочки α в грамматике $G = (T, N, P, S)$, т. е.

$$\text{first}(\alpha) = \{ a \in T \mid \alpha \Rightarrow a\alpha', \text{ где } \alpha \in (T \cup N)^+, \alpha' \in (T \cup N)^* \}.$$

Множество **follow**(A) — это множество терминальных символов, которые могут появляться в сентенциальных формах грамматики $G = (T, N, P, S)$ непосредственно справа от A (или от цепочек, выводимых из A), т.е.

$$\text{follow}(A) = \{ a \in T \mid S \Rightarrow \alpha A \beta, \beta \Rightarrow a \gamma, A \in N, \alpha, \beta, \gamma \in (T \cup N)^* \}.$$

Итерационные правила в КС-грамматиках

Наличие в грамматике итерационных правил вида

$$A \rightarrow \alpha \{ \beta \} \gamma, \quad \text{где } A \in N, \beta \in (T \cup N)^+, \alpha, \gamma \in (T \cup N)^*$$

скрывает правила с ε -альтернативой.

Чтобы проверить грамматику с итерационными правилами на применимость РС-метода, надо заменить итерационные правила обычными правилами следующим образом: каждое правило вида

$$A \rightarrow \alpha \{ \beta \} \gamma \quad \text{заменить на два правила} \quad \begin{array}{l} A \rightarrow \alpha W \gamma \\ W \rightarrow \beta W \mid \varepsilon, \end{array}$$

(где W – новый нетерминал, ранее не принадлежавший множеству N), а затем проверить критерий применимости РС-метода.

Процедура по РС-методу для итерационных правил пишется по следующей схеме:

```
void A () {  
    < анализ цепочки  $\alpha$  >;  
    while (<текущий_символ_входной_цепочки == первый_символ_  $\beta$  >)  
        < анализ цепочки  $\beta$  >;  
    < анализ цепочки  $\gamma$  >;  
}
```

Преобразования грамматик

Проблема возможности построения грамматики, к которой применим метод рекурсивного спуска, эквивалентной грамматике, не удовлетворяющей критерию применимости РС-метода, является **алгоритмически неразрешимой проблемой**.

1) Если в грамматике есть нетерминалы, правила вывода которых **леворекурсивны**, т.е. имеют вид

$$A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \dots \mid \beta_m, \quad // A \rightarrow A\alpha \mid \beta$$

где $\alpha_i \in (T \cup N)^+$, $\beta_j \in (T \cup N)^*$, $i = 1, 2, \dots, n$; $j = 1, 2, \dots, m$,

то непосредственно применять РС-метод нельзя.

Левую рекурсию всегда можно заменить правой:

$$\begin{aligned} A &\rightarrow \beta_1 A' \mid \dots \mid \beta_m A' & // A &\rightarrow \beta A' \\ A' &\rightarrow \alpha_1 A' \mid \dots \mid \alpha_n A' \mid \varepsilon & // A' &\rightarrow \alpha A' \mid \varepsilon \end{aligned}$$

Будет получена грамматика, эквивалентная данной, т.к. из нетерминала A по-прежнему выводятся цепочки вида

$$\beta_j \{\alpha_i\}, \text{ где } i = 1, 2, \dots, n; j = 1, 2, \dots, m.$$

Преобразования грамматик

2) Если в грамматике есть нетерминал, у которого **несколько** правил вывода, и среди них есть правила, **начинающиеся нетерминальными символами**, т.е. имеют вид:

$$\begin{array}{l} A \rightarrow V_1\alpha_1 \mid \dots \mid V_n\alpha_n \mid a_1\beta_1 \mid \dots \mid a_m\beta_m \\ \mathbf{C\beta} \end{array} \qquad // \mathbf{A \rightarrow B\alpha \mid}$$
$$\begin{array}{l} V_1 \rightarrow \gamma_{11} \mid \dots \mid \gamma_{1k} \\ \dots \\ V_n \rightarrow \gamma_{n1} \mid \dots \mid \gamma_{np}, \end{array} \qquad \begin{array}{l} // \mathbf{B \rightarrow \gamma 1 \mid \gamma 2} \\ // \mathbf{C \rightarrow \delta 1 \mid \delta 2} \end{array}$$

где $V_i \in N$; $a_j \in T$; $\alpha_i, \beta_j, \gamma_{ij} \in (T \cup N)^*$,

то можно заменить вхождения нетерминалов V_i их правилами вывода в надежде, что правила вывода из нетерминала A станут удовлетворять требованиям метода рекурсивного спуска:

$$\begin{array}{l} A \rightarrow \gamma_{11}\alpha_1 \mid \dots \mid \gamma_{1k}\alpha_1 \mid \dots \mid \gamma_{n1}\alpha_n \mid \dots \mid \gamma_{np}\alpha_n \mid a_1\beta_1 \mid \dots \mid a_m\beta_m \\ // \mathbf{A \rightarrow \gamma 1\alpha \mid \gamma 12\alpha \mid \delta 1\beta \mid \delta 2\beta} \end{array}$$

Преобразования грамматик

3) Если в грамматике есть нетерминал, у которого несколько правил вывода начинаются **одинаковыми терминальными символами**, т.е. имеют вид

$$A \rightarrow a\alpha_1 \mid a\alpha_2 \mid \dots \mid a\alpha_n \mid \beta_1 \mid \dots \mid \beta_m, \quad // \mathbf{A \rightarrow a\alpha_1 \mid a\alpha_2 \mid \beta}$$

$$\text{где } a \in T; \alpha_i, \beta_j \in (T \cup N)^*,$$

то непосредственно применять РС-метод нельзя. Можно преобразовать правила вывода данного нетерминала, объединив правила с общими началами в одно правило:

$$\begin{array}{ll} A \rightarrow aA' \mid \beta_1 \mid \dots \mid \beta_m & // \mathbf{A \rightarrow aA' \mid \beta} \\ A' \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n & // \mathbf{A' \rightarrow \alpha_1 \mid \alpha_2} \end{array}$$

Будет получена грамматика, эквивалентная данной.

Преобразования грамматик

4) Если в правилах вывода грамматики есть пустая альтернатива, т.е. есть правила вида

$$A \rightarrow a_1\alpha_1 \mid \dots \mid a_n\alpha_n \mid \varepsilon,$$

то метод рекурсивного спуска может оказаться неприменимым.

Например, для грамматики $G = (\{a, b\}, \{S, A\}, P, S)$, где

$$\begin{aligned} P: \quad S &\rightarrow bAa && // S \rightarrow bAc \\ A &\rightarrow aA \mid \varepsilon \end{aligned}$$

РС-анализатор, реализованный по обычной схеме, будет таким:

```
void S ( ) {
    if (c == 'b') {
        gc( ); A( );
        if (c != 'a') throw c; }
    else throw c;
}
```

```
void A( ) {
    if (c == 'a') {
        gc(); A(); }
}
```

Тогда при анализе цепочки **baa** функция $A()$ будет вызвана два раза; она прочитает подцепочку **aa**, хотя второй символ **a** - это часть подцепочки, выводимой из S . В результате окажется, что **baa** не принадлежит языку, порождаемому грамматикой, но в действительности это не так.

Т.е. если $\mathbf{FIRST(A) \cap FOLLOW(A) \neq \emptyset}$, то метод рекурсивного спуска **неприменим** к данной грамматике.

Итак, если в грамматике есть правила с пустой альтернативой вида:

$$A \rightarrow \alpha_1 A \mid \dots \mid \alpha_n A \mid \beta_1 \mid \dots \mid \beta_m \mid \varepsilon$$
$$B \rightarrow \alpha A \beta$$

и $first(A) \cap follow(A) \neq \emptyset$ (из-за вхождения A в правила вывода для B), то можно преобразовать грамматику, заменив правило вывода из B на следующие два правила:

$$B \rightarrow \alpha A'$$
$$A' \rightarrow \alpha_1 A' \mid \dots \mid \alpha_n A' \mid \beta_1 \beta \mid \dots \mid \beta_m \beta \mid \beta$$

Полученная грамматика будет эквивалентна исходной, т. к. из B по-прежнему выводятся цепочки вида $\alpha \{\alpha_i\} \beta_j \beta$ либо $\alpha \{\alpha_i\} \beta$.

Однако правило вывода для нетерминального символа A' будет иметь альтернативы, начинающиеся одинаковыми терминальными символами (т. к. $first(A) \cap follow(A) \neq \emptyset$); следовательно, потребуются дальнейшие преобразования, и успех не гарантирован.

Пример преобразования грамматики

$$S \rightarrow fASd \mid \varepsilon$$
$$A \rightarrow \underline{Aa} \mid \underline{Ab} \mid dB \mid f$$
$$B \rightarrow bcB \mid \varepsilon$$
 \Rightarrow
$$S \rightarrow fASd \mid \varepsilon$$
$$A \rightarrow \underline{dBA'} \mid fA'$$
$$A' \rightarrow aA' \mid bA' \mid \varepsilon$$
$$B \rightarrow bcB \mid \varepsilon$$
$$\text{FIRST}(S) = \{ f \}, \text{ FOLLOW}(S) = \{ d \}; \cap = \emptyset$$
$$\text{FIRST}(A') = \{ a, b \}, \text{ FOLLOW}(A') = \{ f, d \}; \cap = \emptyset$$
$$\text{FIRST}(B) = \{ b \}, \text{ FOLLOW}(B) = \{ a, b, f, d \}; \cap = \{ b \} \neq \emptyset$$
$$S \rightarrow fASd \mid \varepsilon$$
$$A \rightarrow dB' \mid fA'$$
$$\Rightarrow B' \rightarrow bcB' \mid A' \quad \rightarrow \quad B' \rightarrow \underline{bcB'} \mid aA' \mid \underline{bA'} \mid \varepsilon$$
$$A' \rightarrow aA' \mid bA' \mid \varepsilon$$
$$B \rightarrow bcB \mid \varepsilon - \text{недостижимые правила, их можно убрать.}$$
$$S \rightarrow fASd \mid \varepsilon$$
$$A \rightarrow dB' \mid fA'$$
$$\Rightarrow B' \rightarrow \underline{bC} \mid aA' \mid \varepsilon$$
$$C \rightarrow \underline{cB'} \mid A' \quad \rightarrow \quad C \rightarrow \underline{cB'} \mid aA' \mid \underline{bA'} \mid \varepsilon$$
$$A' \rightarrow aA' \mid bA' \mid \varepsilon$$

S - не менялось,

$$\text{FIRST}(B') = \{ a, b \}, \text{ FOLLOW}(B') = \{ f, d \}; \cap = \emptyset$$
$$\text{FIRST}(A') = \{ a, b \}, \text{ FOLLOW}(A') = \{ f, d \}; \cap = \emptyset$$
$$\text{FIRST}(C) = \{ a, b, c \}, \text{ FOLLOW}(C) = \{ f, d \}; \cap = \emptyset$$

Т.е. получили эквивалентную грамматику, к которой применим метод рекурсивного спуска.

Синтаксический анализатор для М-языка

```
class Parser {
    Lex curr_lex;
    type_of_lex c_type;
    int c_val;
    Scanner scan;
    Stack < int, 100 > st_int;
    Stack < type_of_lex, 100 > st_lex;
    void P(); void D1(); void D (); void B (); void S ();
    void E(); void E1(); void T(); void F();
    void dec ( type_of_lex type); void check_id ();
    void check_op (); void check_not (); void eq_type ();
    void eq_bool (); void check_id_in_read ();
    void gl ( ) {
        curr_lex = scan.get_lex ();
        c_type = curr_lex.get_type ();
        c_val = curr_lex.get_value ();
    }
public:
    Poliz prog;
    Parser (const char *program) : scan (program), prog (1000) { }
    void analyze ( );
};
```

Синтаксический анализатор для М-языка

```
void Parser::analyze () {  
    gl ();  
    P ();  
    prog.print ();  
    cout << endl << "Yes!!!" << endl;  
}
```

```
void Parser::P () {  
    if (c_type == LEX_PROGRAM)  
        gl ();  
    else  
        throw curr_lex;  
    D1();  
    if (c_type == LEX_SEMICOLON)  
        gl ();  
    else  
        throw curr_lex;  
    B ();  
    if (c_type != LEX_FIN)  
        throw curr_lex;  
}
```

Другие методы распознавания КС-языков.

Существуют другие методы анализа, анализаторы по которым применимы к более широким подклассам КС-грамматик, чем РС-анализатор, но обладающие теми же свойствами: входная цепочка считывается **один раз слева направо**, процесс разбора **детерминирован**, и в результате на обработку цепочки длины n расходуется время cn .

Например:

Анализатор для LL(k)-грамматик осуществляет левосторонний вывод (анализ сверху-вниз), принимая во внимание k входных символов, расположенных справа, от текущей позиции.

Анализатор для LR(k)-грамматик осуществляет правосторонний вывод (анализ снизу-вверх), принимая во внимание k входных символов, расположенных справа, от текущей позиции.

Анализатор для грамматик предшествования осуществляет правосторонний вывод (анализ снизу-вверх), учитывая только некоторые отношения между парами смежных символов выводимой цепочки.

Другие методы распознавания КС-языков.

Любая грамматика, анализируемая РС-методом, является LL(1)-грамматикой - обратное неверно.

Любая LL-грамматика является LR-грамматикой - обратное неверно.

Левосторонний (нисходящий) синтаксический анализ предпочтителен с точки зрения процесса трансляции, поскольку на его основе легче организовать процесс порождения цепочек результирующего языка.

Восходящий синтаксический анализ привлекательнее тем, что часто для языков программирования легче построить правоанализируемую грамматику, а на ее основе - правосторонний распознаватель.

Конкретный выбор анализатора зависит от конкретного компилятора, от сложности грамматики входного языка программирования и от того, как будут использованы результаты работы анализатора.