13. Нисходящий анализ методом рекурсивного спуска.

Нисходящий анализ методом рекурсивного спуска — это общий метод синтаксического разбора, при котором для каждого нетерминала грамматики A создается отдельная процедура, отвечающая за распознавание фрагмента входной цепочки, выводимого из этого нетерминала.
Процесс начинается с вызова процедуры для аксиомы грамматики S.
Внутри каждой такой функции происходит выбор подходящего правила вывода (вида $A \to \alpha$) и его последовательная обработка: если в правой части встречается терминал, он сравнивается с текущим символом входа, а если нетерминал — рекурсивно вызывается соответствующая ему процедура.

Грамматика $G = (\Sigma, \Gamma, P, S)$, w — входная цепочка, `lookahead` — глобальный указатель на текущий символ.

```
1 void A() {
2 old = lookahead
3 цикл по правилам (A → X1 ... Xn) :
          w[lookahead] ∈ SELECT(A → X1 ... Xn)
4     цикл по i = 1 .. n
5       если (Xi ∈ Γ)            вызов Xi()
6       иначе если (Xi = w[lookahead]) lookahead++
7       иначе
8         lookahead = old
9         перейти на 3
10       если i = n
11         возврат
12 *откат*}
```

В случае неуспешной обработки очередного правила вывода варианты откатов рассматриваются в следующем порядке:

1. если ещё не исчерпаны все правила из множества SELECT (цикл в строке 3), происходит переход к следующему правилу;
2. если процедура проверила все правила для нетерминала A и успешного вывода фрагмента входной строки не произошло, происходит откат к ближайшему левому брату B нетерминала A и выбор другого правила для B при его наличии;
3. если откат дошёл до самого левого сына нетерминала C и успешного вывода фрагмента входной строки не произошло, происходит откат в процедуру для нетерминала C.

Основные принципы и особенности реализации этого метода включают:

- **Механизм выбора правила:** Для выбора правила вывода используется текущий символ входной цепочки и **множества выбора** ($select$), аналогично работе МП-автомата. Если грамматика является **LL(1)-грамматикой**, то выбор правила всегда детерминирован, и алгоритм работает **без откатов** (backtracking). В общем случае, если выбор неоднозначен, при неудаче одного варианта происходит «откат» для выбора другой альтернативы.
- **Обработка символов:** При обработке правила $A \to X_1 \ldots X_n$ символы правой части анализируются по порядку. Если X_i — терминал, он сравнивается с текущим символом (`lookahead`); при совпадении указатель входа сдвигается, при несовпадении — фиксируется ошибка или происходит откат. Если X_i — нетерминал, вызывается его функция, которая может принимать наследуемые атрибуты как аргументы и возвращать синтезированные атрибуты как результат.
- **Ограничение по левой рекурсии:** Метод рекурсивного спуска **неприменим к леворекурсивным грамматикам**, так как наличие правил вида $A \to A\alpha$ приводит к бесконечной рекурсии (процедура будет вызывать сама себя бесконечно, не продвигаясь по входной цепочке). Перед реализацией метода такие грамматики должны быть преобразованы в эквивалентные нелеворекурсивные.
- **Программная структура:** Транслятор, работающий по этому методу, обычно использует глобальную переменную `lookahead` для хранения текущего символа входного потока. Каждая функция нетерминала анализирует этот символ через специальные условия (например, `case` или `if-then-else`) для определения дальнейшего пути разбора.

Метод рекурсивного спуска начинает работу с вызова процедуры для аксиомы S грамматики с позицией `lookahead` $= 0$. Успешное завершение работы — это возврат управления в процедуру S, когда `lookahead` указывает на конец входной строки. Для корректных откатов на протяжении работы алгоритма должна каким-либо образом сохраняться информация о том, какие правила для каждого внутреннего узла были обработаны.
Позднее в курсе будет рассмотрен построение нисходящего транслятора, основанного на методе рекурсивного спуска, осуществляющего семантический анализ параллельно с синтаксическим.

-