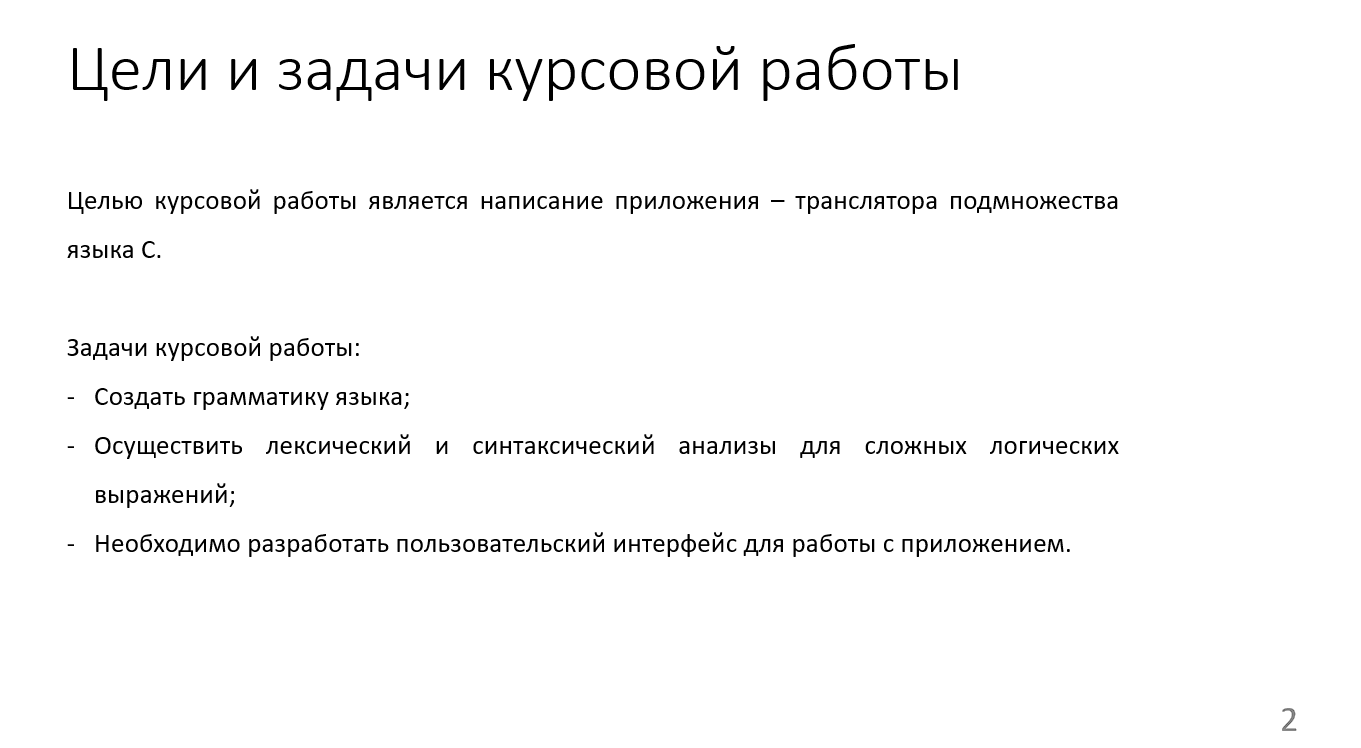
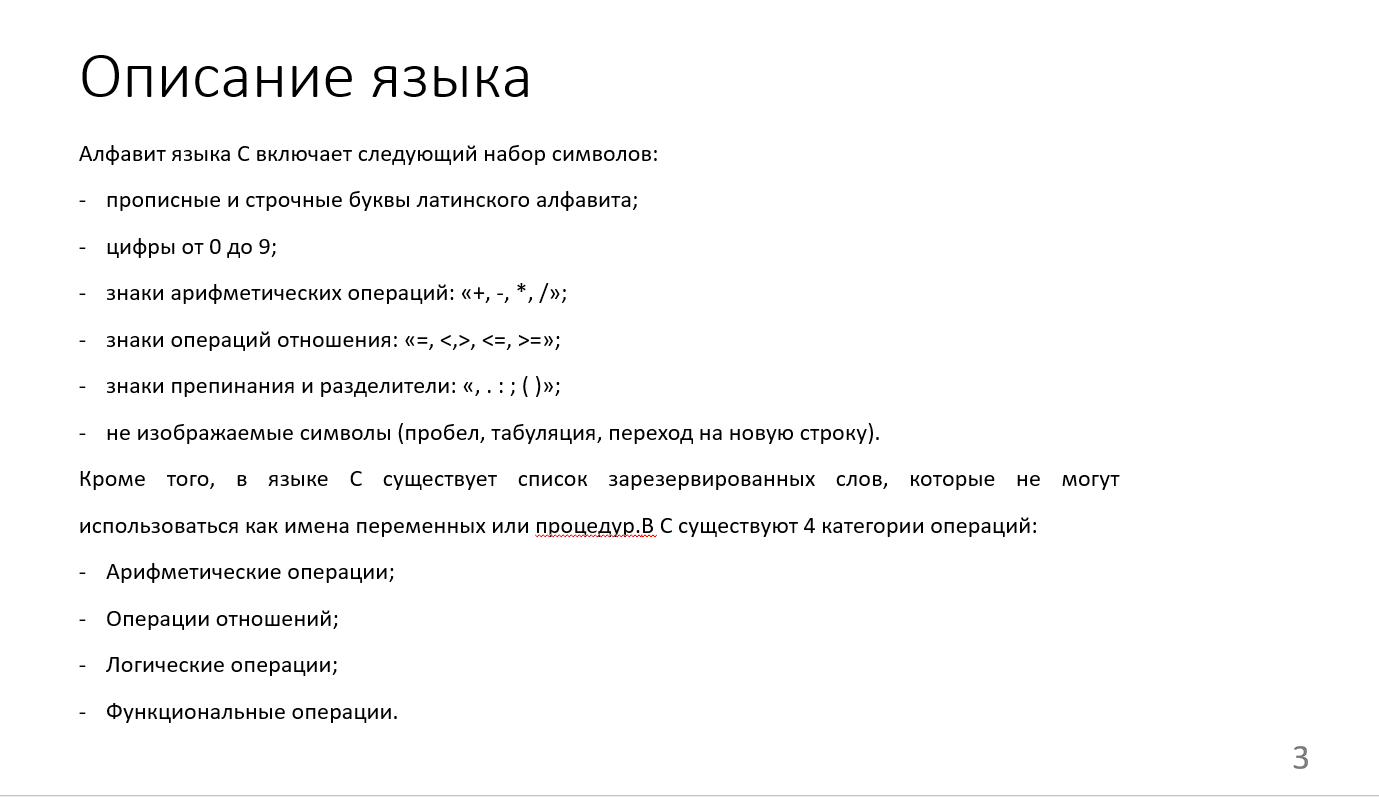
Просто заучить данный текст на слайде. Рассказывать.



Просто заучить данный текст на слайде.



Левая факторизация - это процесс, который используется для устранения левой рекурсии из контекстно-свободной грамматики. Однако левая факторизация также может привести к неоднозначности, когда у нас есть несколько вариантов продукции, и нам нужно решить, какой использовать.

Чтобы избавиться от левой факторизации, можно выполнить следующие шаги:

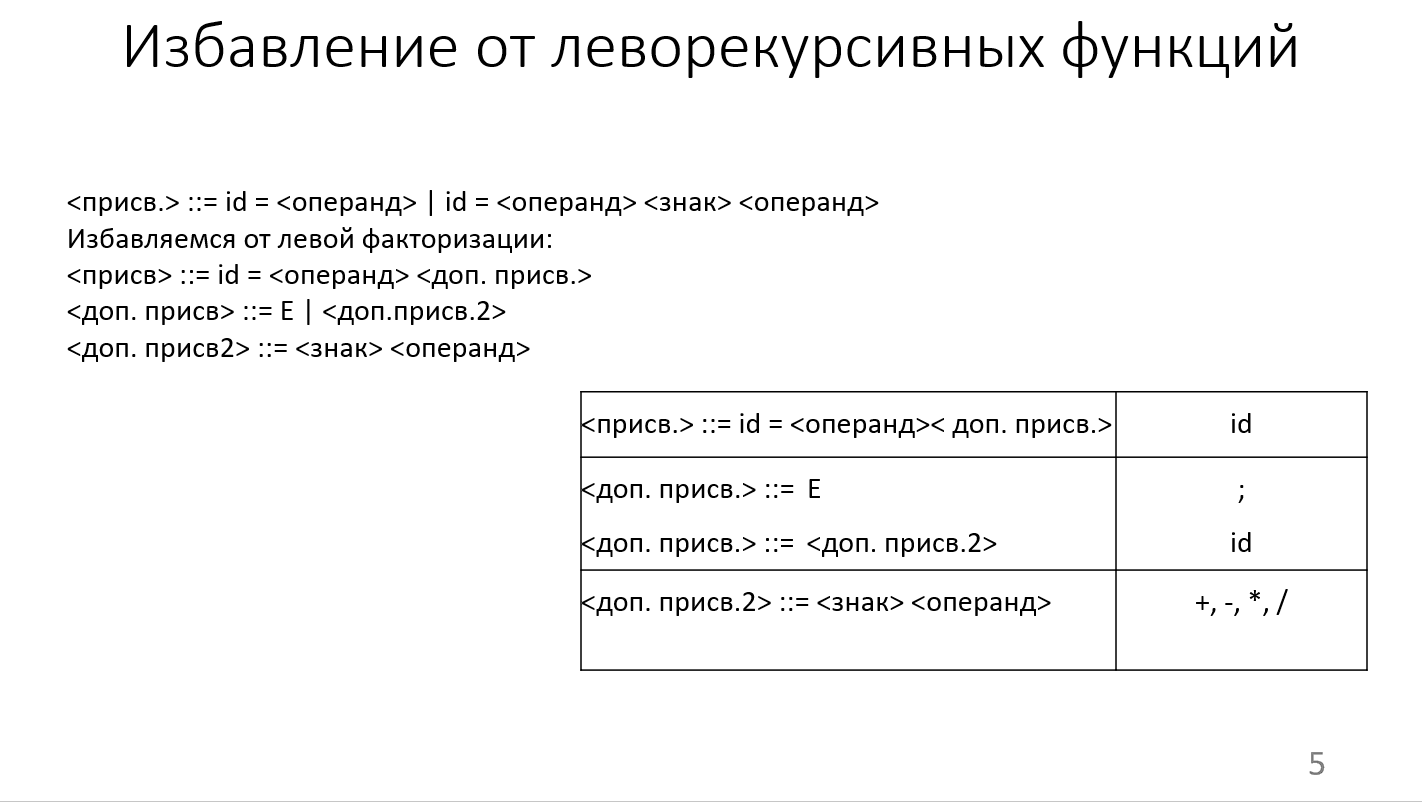
1. Разбить каждое правило на две части: одну для левой рекурсии и другую для оставшейся части продукции.

2. Создать новый нетерминал для каждой части правила, который будет представлять эту часть в грамматике.

3. Заменить леворекурсивную часть правила новым нетерминалом.

4. Факторизовать правую часть правила.

5. Повторить эту процедуру, пока все правила не будут факторизованы.

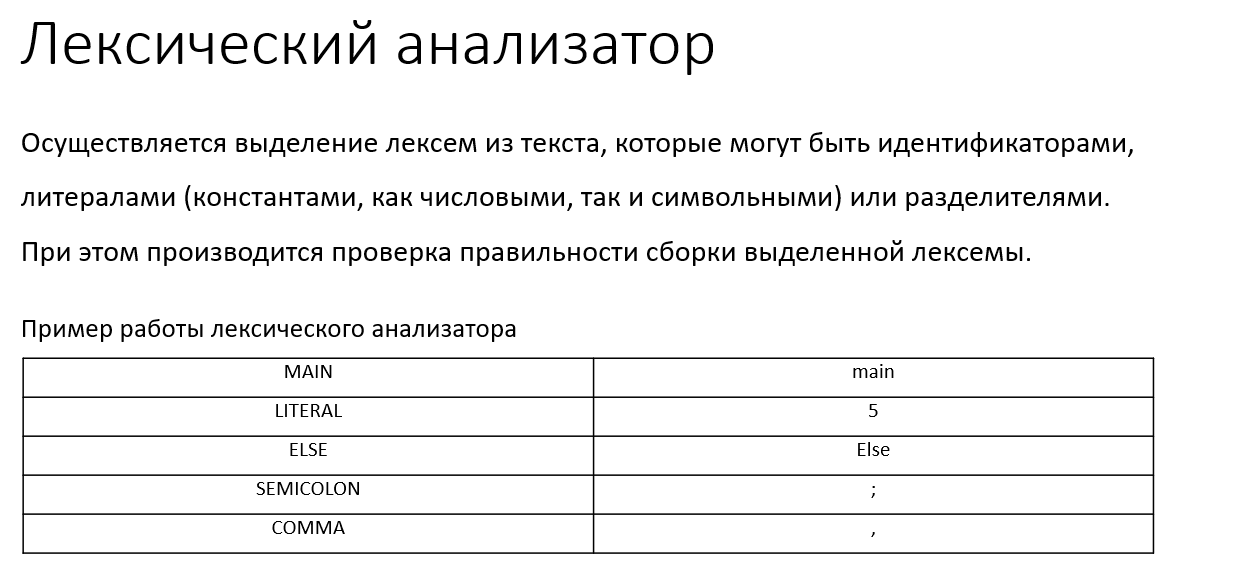


Лексический анализатор - это компонент компилятора или интерпретатора, который преобразует входную последовательность символов (обычно текст программы) в последовательность лексем (токенов), которые представляют собой слова и символы, имеющие смысл на уровне языка программирования. Лексический анализатор также извлекает значение лексем (константы) и определяет их тип (например, число или строку).

Работа лексического анализатора начинается с чтения входной последовательности символов и их разбиения на лексемы. Лексический анализатор использует правила (регулярные выражения или конечные автоматы), которые определяют, какие символы составляют определенные лексемы, а также определяют типы лексем.

Примеры лексем, которые могут быть выделены лексическим анализатором, включают в себя ключевые слова, идентификаторы (названия переменных), константы (например, числа, строки, символы), операторы и знаки препинания.

Лексический анализатор является первым шагом в компиляции и интерпретации программы. Результатом работы лексического анализатора является последовательность лексем, которая передается следующему компоненту компилятора или интерпретатора для синтаксического анализа.



Нисходящий анализатор (LL-анализатор) является одним из методов синтаксического анализа для преобразования входной последовательности лексем в дерево разбора. Он работает сверху вниз (от корня дерева вниз) и использует контекстно-свободную грамматику для определения правил синтаксического разбора.

Основные принципы нисходящего анализатора включают:

1. Начало с символа начального нетерминала грамматики. Начальный нетерминал представляет собой корневой элемент дерева разбора нашей грамматики, и начиная с него мы можем построить все остальные элементы дерева.

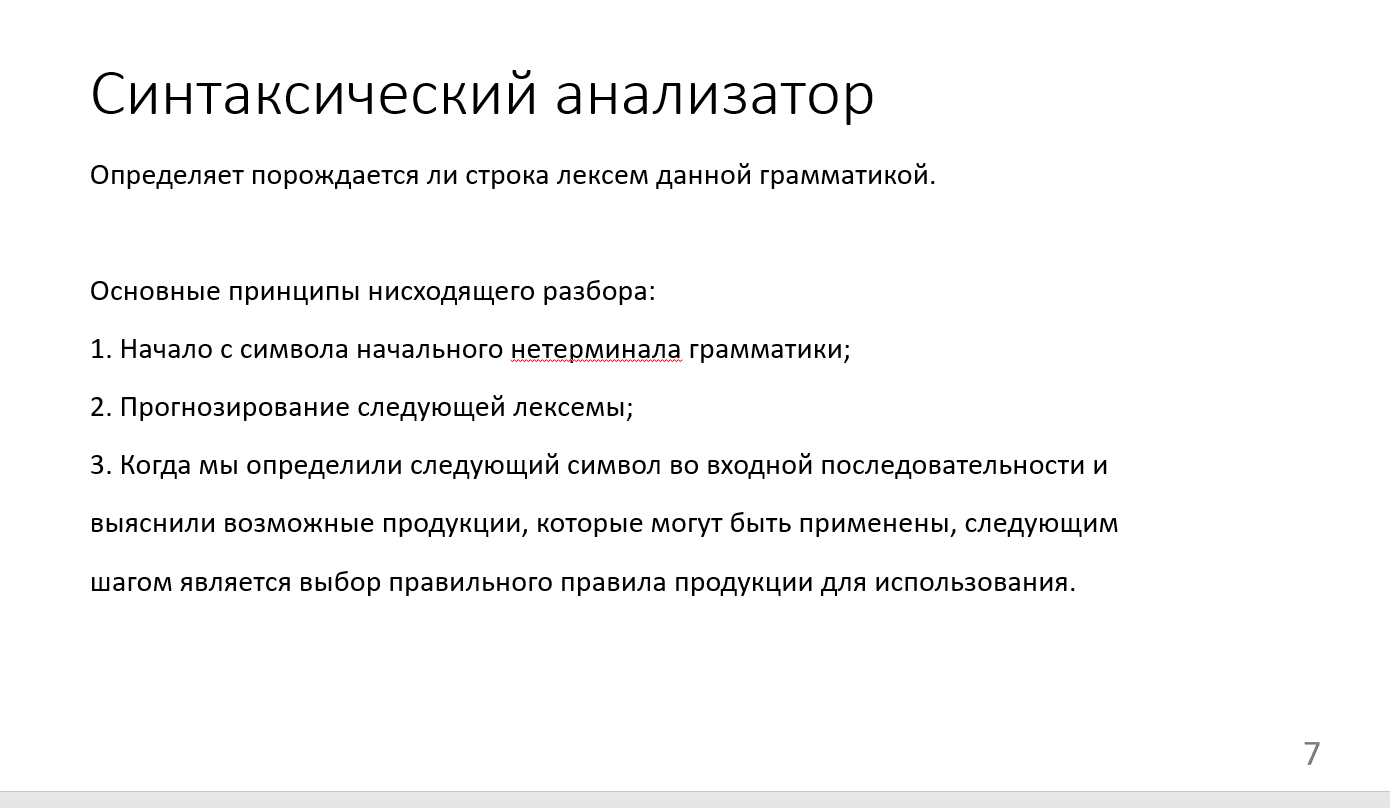
2. Прогнозирование следующей лексемы. Нисходящий анализатор использует стек для предсказания следующей лексемы из входной последовательности. Это делается путем проверки текущего символа в стеке с возможными продукциями, содержащимися в грамматике. Если продукция НЕ содержит терминальных символов (символов, которые присутствуют во входном потоке, таких как числовые значения, операторы, скобки и т. Д.), То мы должны знать, какие терминальные символы присутствуют во входном потоке.

3. Выбор правила продукции. Когда мы определили следующий символ во входной последовательности и выяснили возможные продукции, которые могут быть применены, следующим шагом является выбор правильного правила продукции для использования. Это делается на основе текущего символа в стеке и следующего символа во входной последовательности.

4. Выполнение продукции. Когда мы выбрали нужное правило продукции, следующим шагом является выполнение этого правила, чтобы построить поддерево - увеличивая глубину нашего дерева разбора.

5. Обработка ошибок. Иногда входная последовательность может не соответствовать грамматике, что является ошибкой. Нисходящие анализаторы могут обнаруживать и выводить сообщения об ошибках, в основном за счет использования дополнительных решений семантического разбора и использования таблиц идентификаторов и ключевых слов.

В целом, нисходящий анализатор основывается на правилах контекстно-свободной грамматики для разбора входной последовательности. Он может использоваться для разбора самых разных языков программирования и веб-страниц.



Количество "L" в термине LL(k) обозначает, что нисходящий анализатор использует левостороннюю (left-to-right) стратегию. Количество "L" означает, что анализатор сканирует входную последовательность слева направо, используя стратегию предсказания следующих символов на основе левого контекста.

"K" означает, что нисходящий анализатор использует k-предшествующих символов (lookaheads), чтобы принимать решения о продолжении разбора.

Таким образом, чтобы определить LL(k) нисходящий анализатор, нужно выполнить следующие действия:

1. Определить, какие правила контекстно-свободной грамматики используются в нисходящем анализаторе.

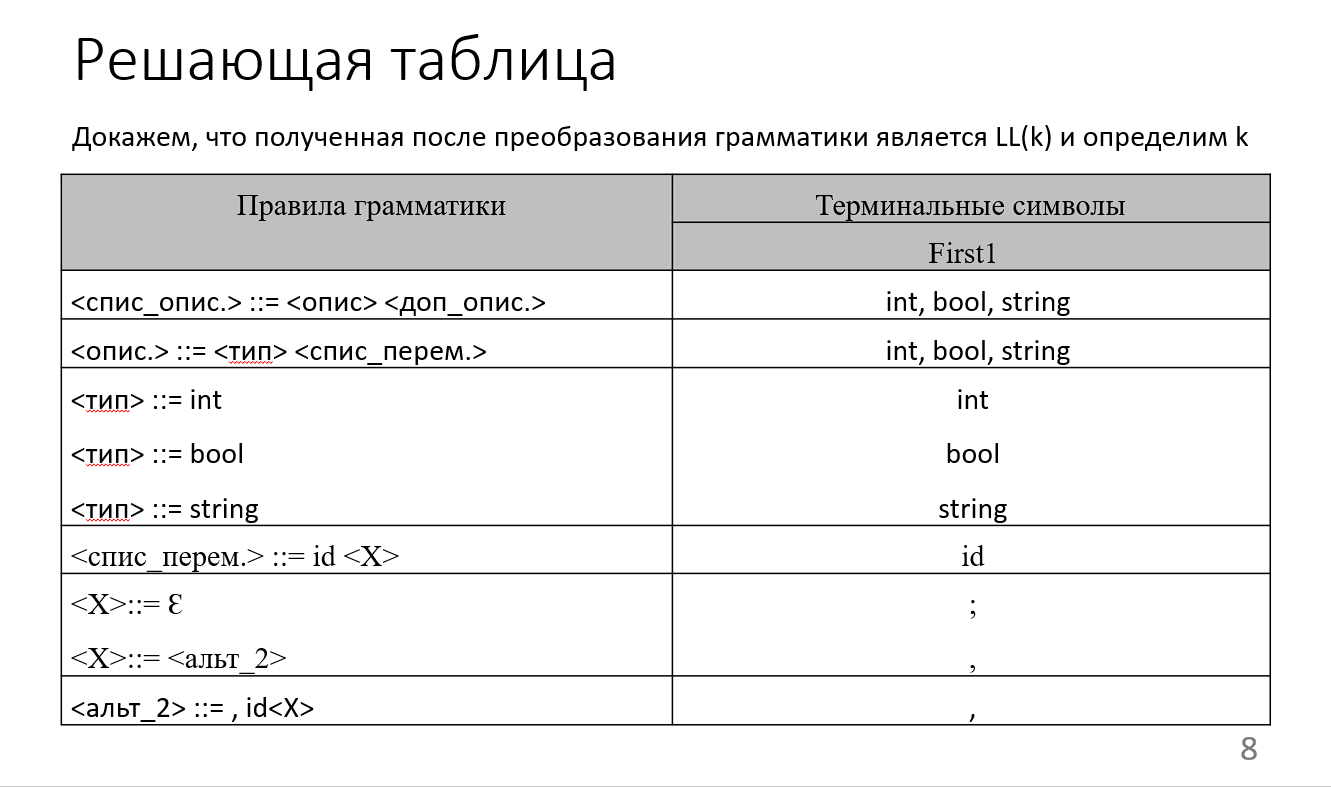
2. Определить, какие символы входной последовательности являются "lookaheads" для каждого правила, которое использует нисходящий анализатор.

3. Посчитать максимальное количество "lookaheads" (k), используемых нисходящим анализатором для определения следующего символа во входной последовательности.

4. Если k>1, то это будет LL(k) нисходящий анализатор.

Например, если нисходящий анализатор использует грамматику, которая имеет три правила и каждый использует один предшествующий символ, то это будет LL(1) нисходящий анализатор.

Важно понимать, что LL(k) нисходящая стратегия более сложная, чем просто LL-стратегия, потому что требуется больше вычислений, чтобы определить следующий символ во входной последовательности. Однако, LL(k) нисходящий анализатор может обрабатывать большие грамматики с большим количеством конфликтов и позволяет более точно описывать синтаксис языка программирования или языка разметки.



Множество FIRST и FOLLOW - это понятия, используемые в синтаксическом анализе как часть работы алгоритма нисходящего анализа (например, LL(1) анализатор).

Множество FIRST для нетерминального символа A представляет собой множество всех терминальных символов, которые могут быть сгенерированы первой продукцией А. Если одна продукция для А начинается с терминального символа, тогда его FIRST множество состоит только из этого символа. Если же с А могут быть сгенерированы пустые строки, то FIRST должны быть расширены до следующей продукции А. Пустое множество, ε , включается в FIRST множество наследования, если нетерминал может быть пустой. Если для этого нетерминала все продукции делают его пустым, то ε также добавляется в FIRST множество.

Множество FOLLOW для нетерминального символа A представляет собой множество всех терминальных символов, которые могут появляться исключительно после A. Другими словами, это множество терминалов, которые могут быть расположены справа от А в правых частях продукций. FOLLOW множество может быть пустым, если A никогда не находится в конце продукции (то есть A является последним символом справа) или если конечный терминал всегда следует за A. В случае, если фактический FOLLOW множество является пустым, добавлять ε в продукции A нельзя.

Множества FIRST и FOLLOW используются для совершенствования алгоритма нисходящего анализатора (например, LL(1) анализатор), чтобы принимать решения на основе первого терминала и следующих терминалов во входной последовательности. Они применяются для определения следующего символа на основе текущего символа и FOLLOW множества предшествующего символа.

Метод Бауэра-Замельзона - это метод построения леворекурсивных грамматик, который позволяет устранить левую рекурсию путем преобразования грамматики в контекстно-свободную форму Хомского. Он является эффективным инструментом для избавления от леворекурсивных правил в грамматиках.

Основной принцип метода Бауэра-Замельзона состоит из двух шагов:

1. Удаление непосредственной левой рекурсии. Для продукции вида А -> Aα | β, создается новый нетерминал A', который заменяет леворекурсивную продукцию: A -> βA', A' -> αA' | ε.

2. Преобразование грамматики в форму Хомского. Это достигается путем замены каждой продукции, содержащей три или более символа, на эквивалентную пару продукций, содержащих только два символа. Например, если у нас есть продукция вида A -> BCDEF, мы можем заменить ее на A -> BG и G -> CDEF.

Результатом применения метода Бауэра-Замельзона является грамматика, которая не содержит левой рекурсии и находится в форме Хомского. Это может обеспечить более эффективный синтаксический анализ, поскольку алгоритмы компиляции и интерпретации часто лучше работают с грамматиками, которые находятся в форме Хомского. Все правила для метода Бауэра-Замельзона являются детерминированными и не производят ошибок.

