Московский авиационный институт

(национальный исследовательский институт)

Институт «Компьютерные науки и прикладная математика»

Курсовая работа

«Реализация преобразования атрибутной грамматики в дерево зависимостей вычислений»

по курсу

«Системы программирования»

IV семестр

Студент: Меркулов Ф. А.
Группа: М8О-207Б-21

Руководитель: Семёнов А. С.

Оценка:

Дата:

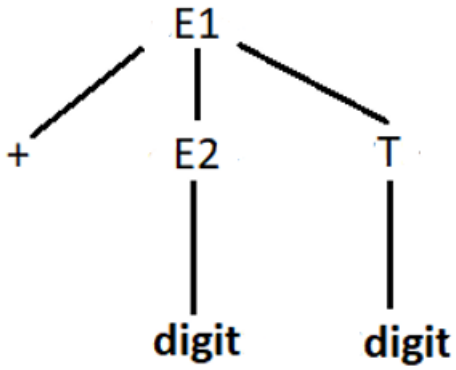
Москва, 2023

Правила атрибутной грамматики:

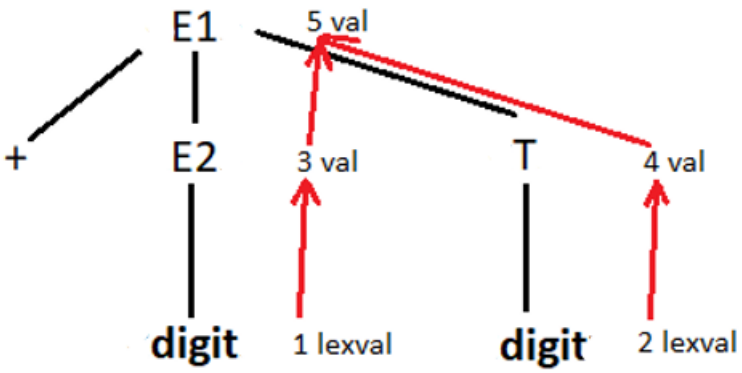
Правила продукции	Семантические правила
E1 -> + E2 T	E1.val := E2.val + T.val
E2 -> **digit**	E2.val := **digit**.lexval
T -> **digit**	T.val := **digit**.lexval

Каждый из нетерминалов *E1*, *E2* и *T* имеет синтезированный атрибут *val*; терминал ***digit*** имеет синтезируемый атрибут *lexval*.

Синтаксическое дерево примет вид:



Дерево зависимостей вычислений:



Узлы 1 и 2 представляют атрибут *lexval*, связанный с двумя листьями, с метками ***digit***.

Узел 3 представляет атрибут *val*, связанный с узлом с меткой *E2*. Ребро в узел 3 из 1 получается из семантического правила, определяющего *E2.val* через ***digit***.*lexval*. В действительности *E2.val* равно ***digit***.*lexval*, но ребро представляет зависимость, а не равенство.

Ребро в узел 4 из 2 получается из семантического правила, определяющего *T.val* через ***digit***.*lexval*. В действительности *T.val* равно ***digit***.*lexval*, но ребро представляет зависимость, а не равенство.

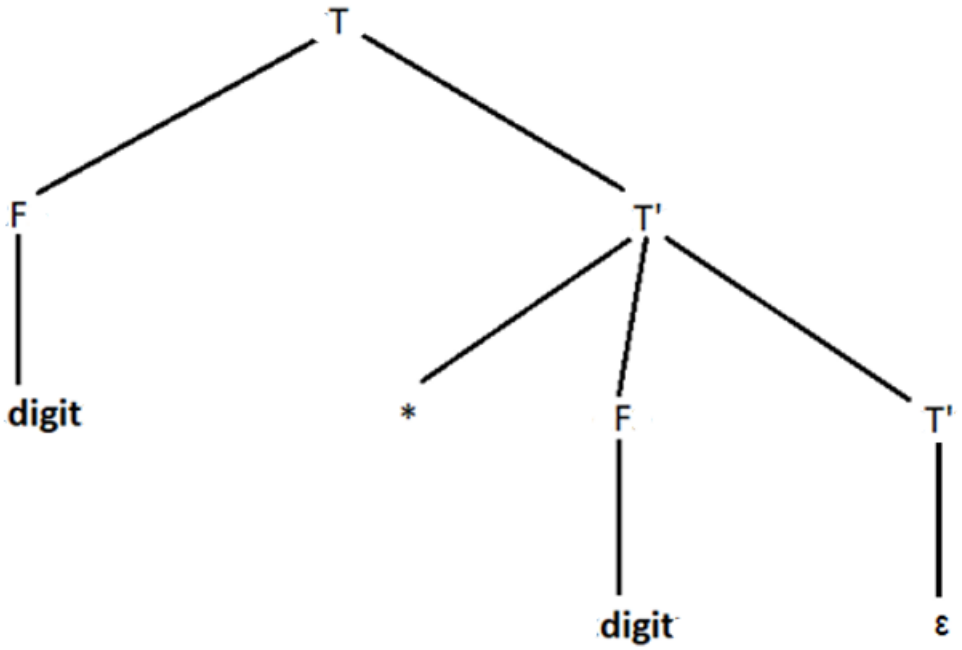
Имеются рёбра в узел 5 из узлов 3 (атрибут *E2.val*) и 4 (атрибут *T.val*), поскольку указанные значения складываются для для получения атрибута *val* в узле 5.

Правила атрибутной грамматики:

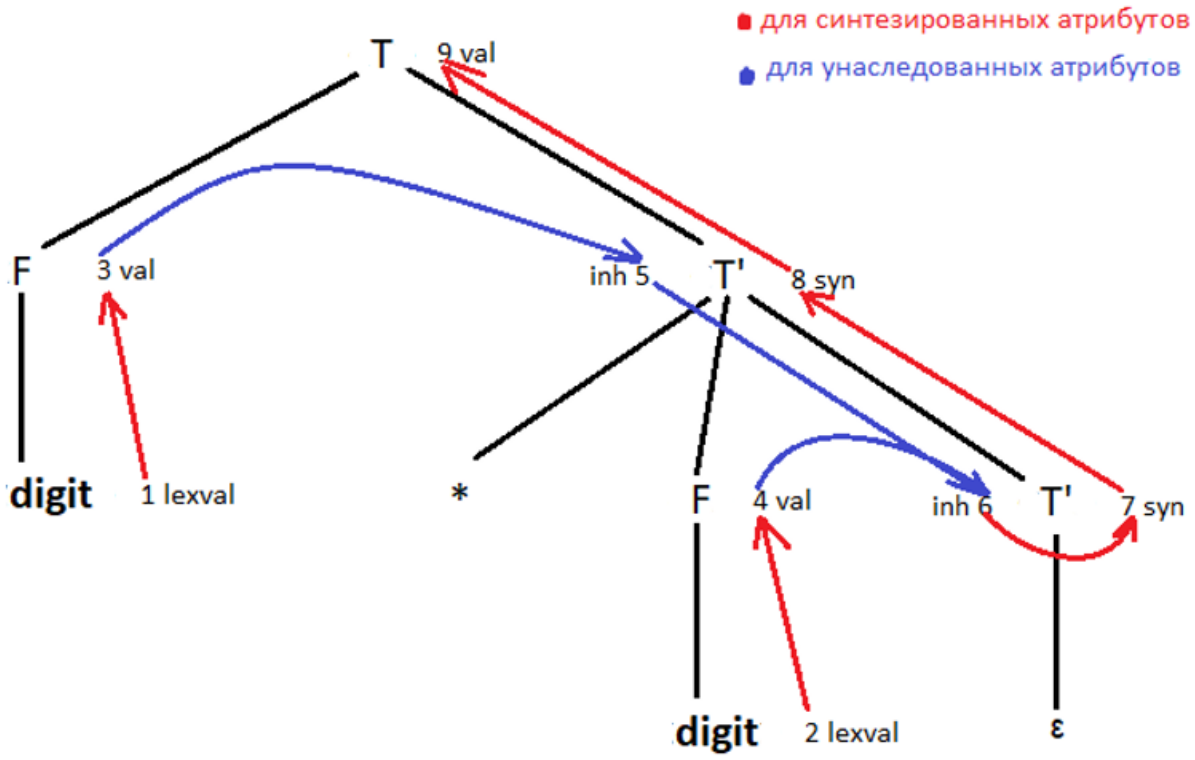
Правила продукции	Семантические правила
T -> F T’	T’.inh = F.val T.val = T’.syn
T’ -> * F T1’	T1’.inh = T’.inh * F.val T’.syn = T1’.syn
T’ -> ε	T’.syn = T’.inh
F -> **digit**	F.val = **digit**.lexval

Каждый из нетерминалов *T* и *F* имеет синтезированный атрибут *val*; терминал ***digit*** имеет синтезированный атрибут *lexval*. Нетерминал T’ имеет два атрибута: наследуемый *inh* и синтезированный *syn.*

Синтаксическое дерево примет вид:



Дерево зависимостей вычислений:



Узлы 1 и 2 представляют атрибут *lexval*, связанный с двумя листьями с метками ***digit***.

Узлы 3 и 4 представляют атрибут *val*, связанный с двумя узлами с метками *F*. Рёбра в узел 3 из 1 и в узел 4 из 2 получаются из семантического правила, определяющего *F.val* через ***digit***.*lexval*. В действительности *F.val* равно ***digit***.*lexval*, но ребро представляет зависимость, а не равенство.

Узлы 5 и 6 представляют наследуемый атрибут *T'.inh*, связанный с каждым появлением нетерминала *T'*. Причиной наличия ребра из 3 в 5 служит правило *T'.inh = F.val*, определяющее *T'.inh* в правом дочернем узле корня с использованием значения *F.val* в левом дочернем узле. Имеются также рёбра в узел 6 из узлов 5 (атрибут *T'.inh*) и 4 (атрибут *F.val*), поскольку указанные значения перемножаются для получения атрибута *inh* в узле 6.

Узлы 7 и 8 представляют синтезируемый атрибут *syn*, связанный с вхождениями *T'*. Ребро из узла 6 в узел 7 связано с семантическим правилом *T'.syn = T'.inh*, назначенным продукции 3. Ребро из узла 7 в узел 8 связано с семантическим правилом, назначенным продукции 2.

Наконец, узел 9 представляет атрибут *T.val*. Ребро из узла 8 в узел 9 вызвано семантическим правилом *T.val = T'.syn*, связанным с продукцией 1.

Теория к алгоритму:

Атрибутная грамматика AG = (G, As, Ai, R)
G = (V, T, P, S) – КС-грамматика
As – конечное множество синтезируемых атрибутов
Ai – конечное множество наследуемых атрибутов
(As ∩ Ai = ∅)
R – конечное множество семантических правил

Атрибутная грамматика AG сопоставляет каждому символу X ∈ (V ∪ T) множество As(X) синтезируемых атрибутов и множество Ai(X) наследуемых атрибутов. Множество всех синтезируемых атрибутов всех символов из (V ∪ T) обозначается AS, наследуемых - AI. Атрибуты разных символов являются различными атрибутами. Атрибутная грамматика AG сопоставляет каждому правилу p ∈ P конечное множество R(p) семантических правил вида

Алгоритм построения дерева по слоям

Вход: Атрибутная грамматика AG = (G, As, Ai, R), - NUMBER_INFERENCE_RULES =[1,2,…] номера правил цепочки вывода

Выход: Вывод поэтапного обхода дерева

NODES = [ ] = create_node(LHS(P(1)))

index =1 //итератор в NODES

i=1 // итератор в NUMBER_INFERENCE_RULES (название ужас)

while (True)

node = NODES[index]

if (LHS(P(NUMBER_INFERENCE_RULES [i])) == node.symbol)

leftIndex = index;

foreach (X ∈ RHS(P(NUMBER_INFERENCE_RULES [i])))

NODES = NODES ∪ create_node(X)

link(index,|NODES|)

end

i ++

end

rightIndex = |NODES|;
BOUNDARIES = BOUNDARIES ∪ [leftIndex, rightIndex]

```
        index++
        if (i > |NUMBER_INFERENCE_RULES|)
          break
        end
end
```

Алгоритм преобразования атрибутной грамматики в дерево зависимостей вычислений:

Вход: Атрибутная грамматика AG = (G, As, Ai, R), NUMBER_INFERENCE_RULES =[1,2,…] номера правил цепочки вывода, проход по слоям дерева разбора в виде упорядоченного множества NODES, упорядоченное множество интервалов в котором выполняется правило продукции BOUNDARIES.
Выход: Упорядоченного множество атрибутов DEPENDENCES; упорядоченное множество индексов вершин, которым соответствуют атрибуты OWNERS

```
DEPENDENCES = [] // множество атрибутов
SIZE = |DEPENDENCES|
OWNERS = [] // упорядоченное множество индексов вершин из NODES, которым соответствуют атрибуты
index = 1 //итератор в NODES
DEPENDENCES = DEPENDENCES ∪ create_node(As(LHS(P(1)))) ∪ create_node(Ai(LHS(P(1)))) //вносим атрибуты первого использованного символа во множество атрибутов
while (|DEPENDENCES| - SIZE > 0)
    OWNERS = OWNERS  ∪ index // ставим атрибутам первого использованного символа  соответствие с вершиной
end
foreach(n ∈ NUMBER_INFERENCE_RULES)
        foreach(X ∈ RHS(P(n)))
                SIZE = |DEPENDENCES|
                DEPENDENCES = DEPENDENCES ∪ create_node(As(X)) ∪ create_node(Ai(X)) // вносим атрибуты каждого использованного символа во множество атрибутов
                while (|DEPENDENCES| – SIZE > 0) // пока мы не рассмотрим все добавленные атрибуты
```

```
                    OWNERS = OWNERS ∪ index // ставим атрибутам
использованного символа  соответствие с вершиной
                    SIZE++
                end
                index++;
    end
end
```

Вход: Атрибутная грамматика AG = (G, As, Ai, R),
NUMBER_INFERENCE_RULES =[1,2,…] номера правил цепочки вывода, упорядоченное множество интервалов в котором выполняется правило продукции BOUNDARIES, упорядоченное множество индексов вершин, которым соответствуют атрибуты OWNERS.
Выход: Дерево зависимостей имеющее вид упорядоченного множество DEPENDENCES со ссылками;

```
i = 1 // итератор в NUMBER_INFERENCE_RULES
while(i <= |NUMBER_INFERENCE_RULES|)
        left = BOUNDARIES[i][0] //индекс, который отвечает за индекс левой
границы в OWNERS
        right = BOUNDARIES[i][1] //индекс, который отвечает за индекс правой
границы в OWNERS
        leftIndex = 1 //итератор по OWNERS, который ищет левую границу
        rightIndex = 1 //итератор по OWNERS, который ищет правую границу
        while (OWNERS[rightIndex] <= right)
            if (OWNERS[leftIndex] < left)
                leftindex++
            end
            rightindex++
        end
        index = leftInfex++ //Первое вхождение left в OWNERS, index – итератор
для DEPENDENCES
        foreach(X ∈ R(P(i)))
            if (|RHS(X)| == 1)
                while(index < rightIndex)
                    if (DEPENDENCES[index] == LHS(X))
                       leftPart = index //индекс в который придёт ссылка
                    end
                    if (DEPENDENCES[index] == RHS(X))
```

```
                    rightPart = index // индекс из которого идёт ссылка
                end
                index++
            end
            link(rightPart, leftPart)
        end
        if (|RHS(X)| > 1)
            foreach(ELEM ∈ RHS(X))
                if(ELEM ∈ (As ∪ Ai))
                    while(index < rightIndex)
                        if (DEPENDENCES[index] == LHS(X))
                            leftPart = index // индекс в который придёт ссылка
                        end
                        if (DEPENDENCES[index] == RHS(X))
                            rightPart = index // индекс из которого идёт ссылка
                        end
                index++
            end
            link(rightPart, leftPart)
        end
    end
    i++
end
```

Алгоритм топологической сортировки:

1. Выберите узел, не имеющий входных рёбер.
2. Удалите узел и все исходящие рёбра.
3. Повторить

Если алгоритм завершается с оставшимися узлами, то в этих узлах происходит направленный цикл и, следовательно, не подходящего порядка (дерево зависимостей не может быть построено (оно уже не будет деревом))

Если алгоритм успешно удаляет все узлы, то порядок удаления является подходящим порядком вычисления, и не было никаких направленных циклов.

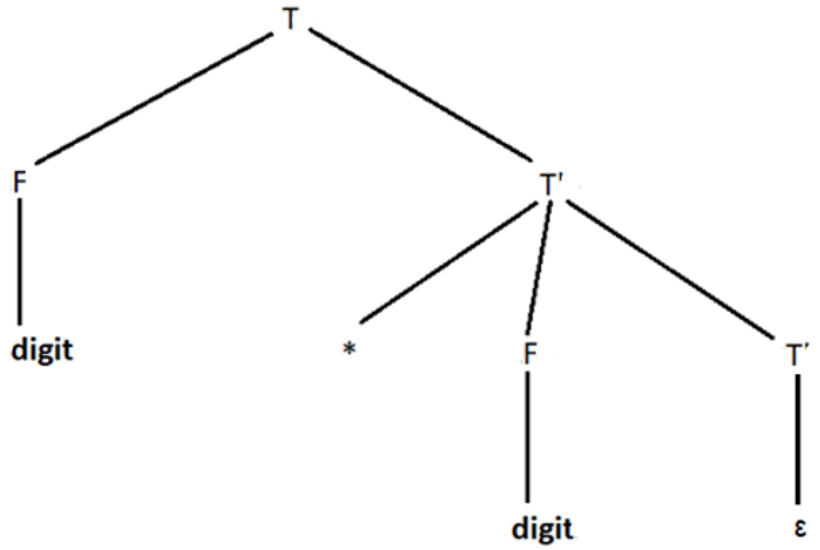
Алгоритм топологической сортировки недетерминированный и зависит от обхода дерева разбора, мы будем обходить дерева слева направо, сверху вниз.

Пошаговый алгоритм использования топологической сортировки для определения порядка вычислений:

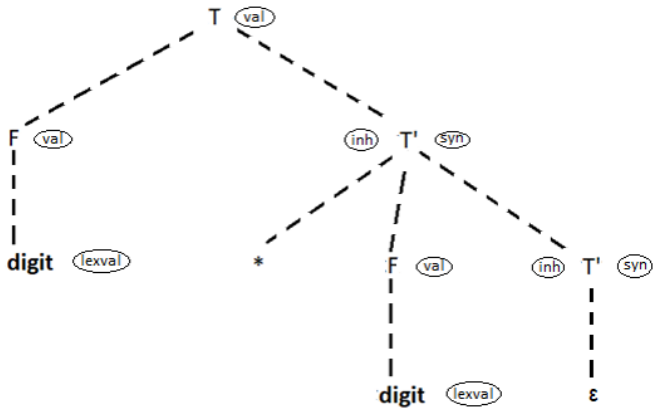
Правила атрибутной грамматики:

Правила продукции	Семантические правила
T -> F T’	T’.inh = F.val T.val = T’.syn
T’ -> * F T1’	T1’.inh = T’.inh * F.val T’.syn = T1’.syn
T’ -> ε	T’.syn = T’.inh
F -> **digit**	F.val = **digit**.lexval

1) По данной АТ-схеме строится синтаксическое дерево разбора вида:

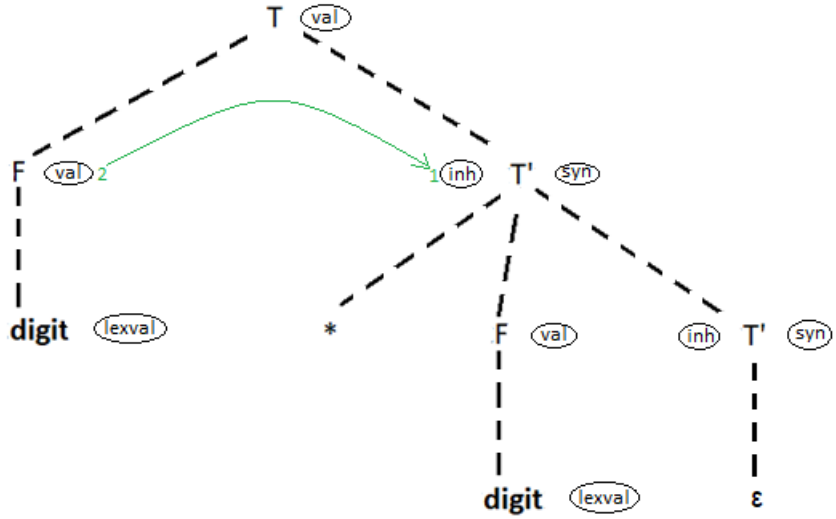


2) Проходим по всем семантическим правилам и определяем всевозможные атрибуты, для всевозможных символов из правил продукции:



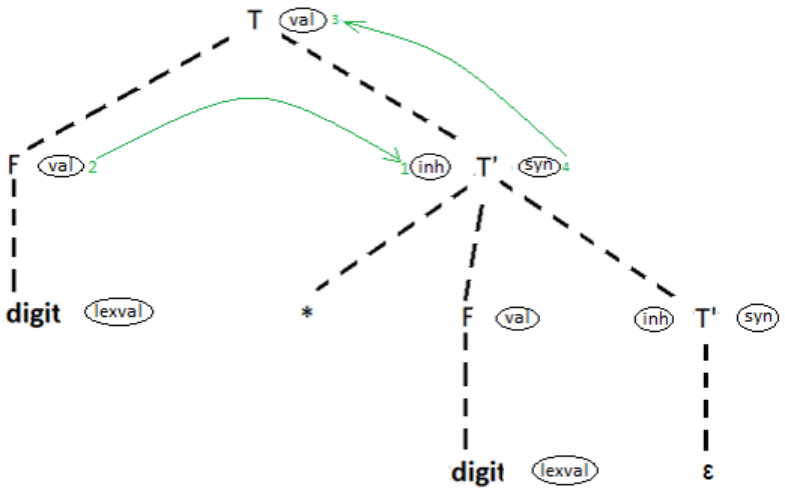
Тем самым мы получаем узлы, между которыми в дальнейшем будут связи зависимостей.
Проходим по всем семантическим правилам и в соответствии с содержимым выстраиваем связи, нумеруя вершины по возрастанию, ради того, чтобы в дальнейшем определить порядок.

а) По первому правилу продукции T -> F T’ и его первому семантическому правилу T’.inh = F.val устанавливаем связь между соответствующими T’.inh и F.val, при этом связь исходит из F.val и идёт в T’.inh, а также им сопоставляются индексы 1 (для T’.inh) и 2 (для F.val), так как это первые 2 встреченных атрибута при обходе.

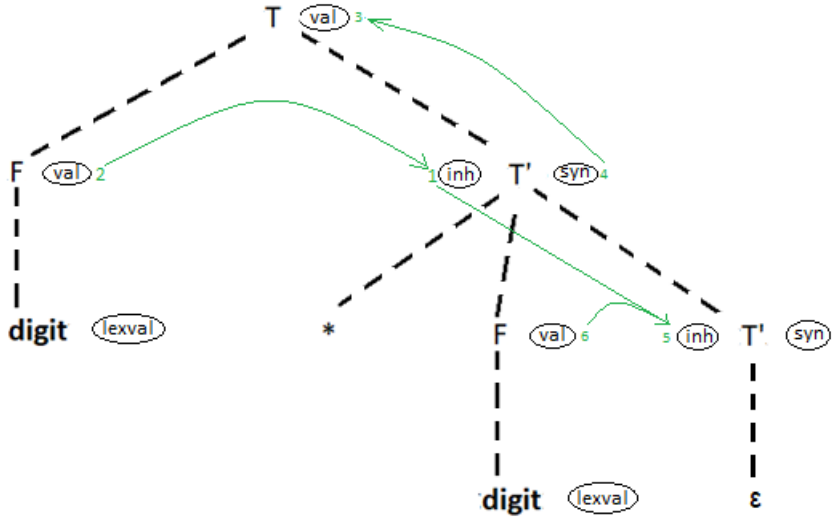


б) По первому правилу продукции T -> F T’ и его второму семантическому правилу T.val = T’.syn устанавливаем связь между соответствующими T.val и T’.syn, при этом связь исходит из T’.syn и идёт в T.val, а также им сопоставляются индексы 3 (для T.val) и 4 (для

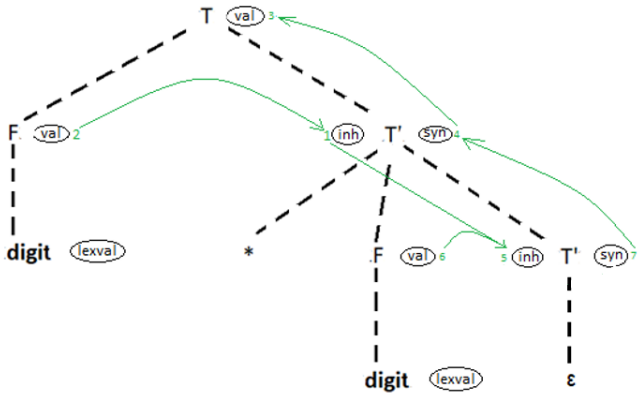
T’.syn), так как это 3 и 4 встреченные атрибуты, при обходе.



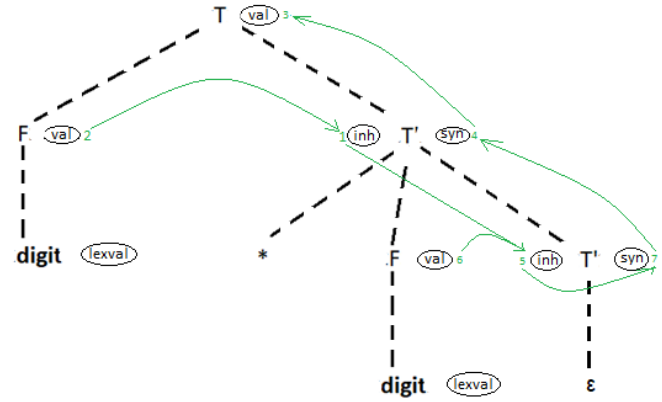
в) По второму правилу продукции T’ -> * F T1’ и его первому семантическому правилу T1’.inh = T’.inh * F.val устанавливаем связь между соответствующими T1’.inh, T’.inh и F.val, при этом связи исходят из T’.inh, F.val и идут в T1’.inh, а так++же им сопоставляются индексы 5 (для T1’.inh) и 6 (для F.val), так как это 5 и 6 встреченные атрибуты, при обходе, а T’.inh уже встречался, поэтому ему не сопоставляется индекс.



г) По второму правилу продукции T’ -> * F T1’ и его первому семантическому правилу T’.syn = T1’.syn устанавливаем связь между соответствующими T’.syn и T1’.syn, при этом связи исходят из T1’.syn и идут в T’.syn, а также индекс 7 сопоставляется с T1’.syn, так как это 7 встреченные атрибуты, при обходе, а T’.syn уже встречался, поэтому ему не сопоставляется индекс.

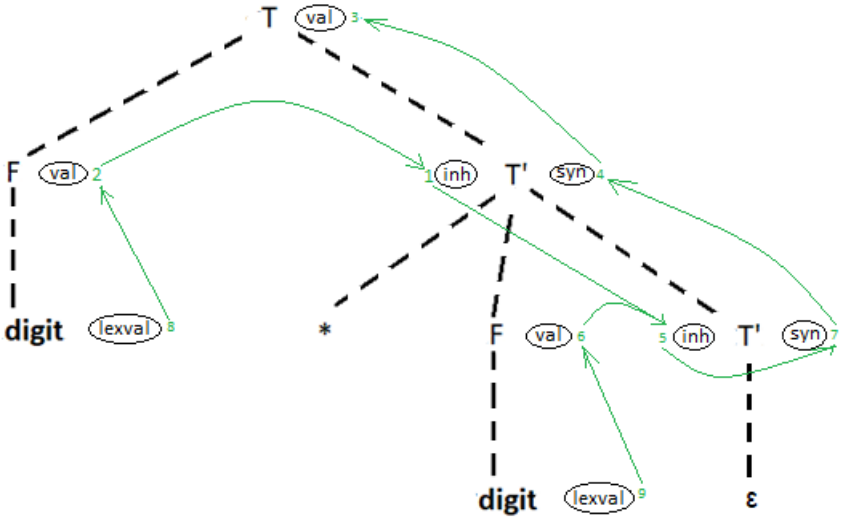


д) По третьему правилу продукции T’ -> ε и его единственному семантическому правилу T’.syn = T’.inh устанавливаем связь между соответствующими T’.syn и T’.inh, при этом связь исходит из T’.syn и идёт в T’.syn, а также индекс не ставятся, так как T’.syn и T’.inh уже встречались при обходе.



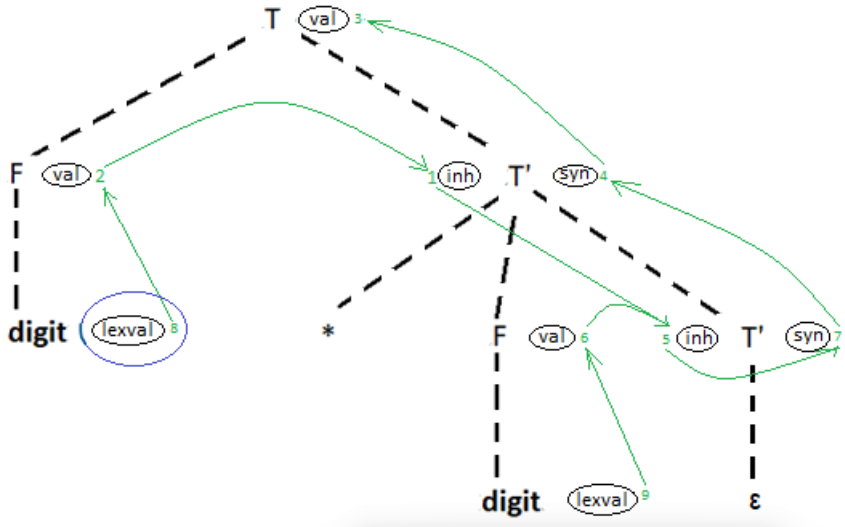
е) По четвёртому (последнему) правилу продукции F -> **digit** и его единственному семантическому правилу F.val = **digit**.lexval устанавливаем связь между соответствующими F.val и digit.lexval, при этом связь исходит из соответствующих digit.lexval и идёт в соответствующие F.val, а также для digit.lexval ставятся индексы 8 и 9, так как это 8 и 9 атрибуты встреченные при обходе, а F.val не получают

новые индексы, так как они уже встречались при обходе.



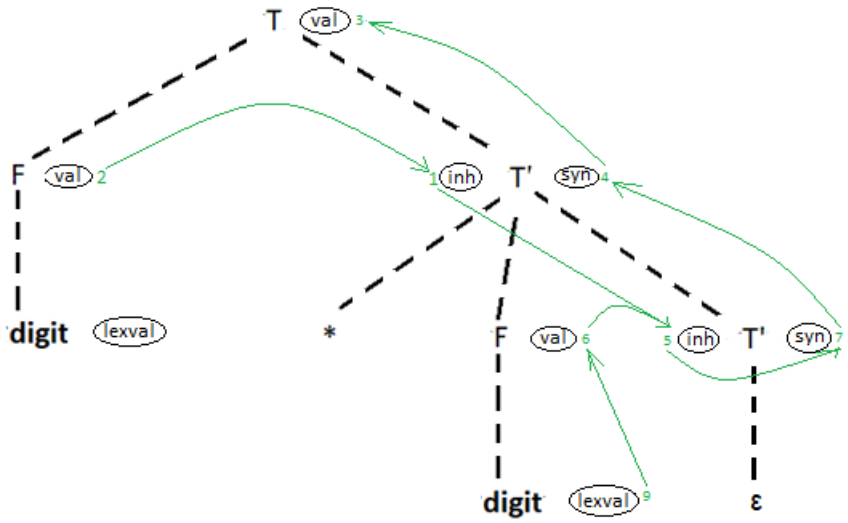
3) После выстроенных связей пытаемся определить порядок в котором надо обходить это дерево зависимостей, например с помощью топологической сортировки:

а) находим узел в который не приходит ни одна связь:



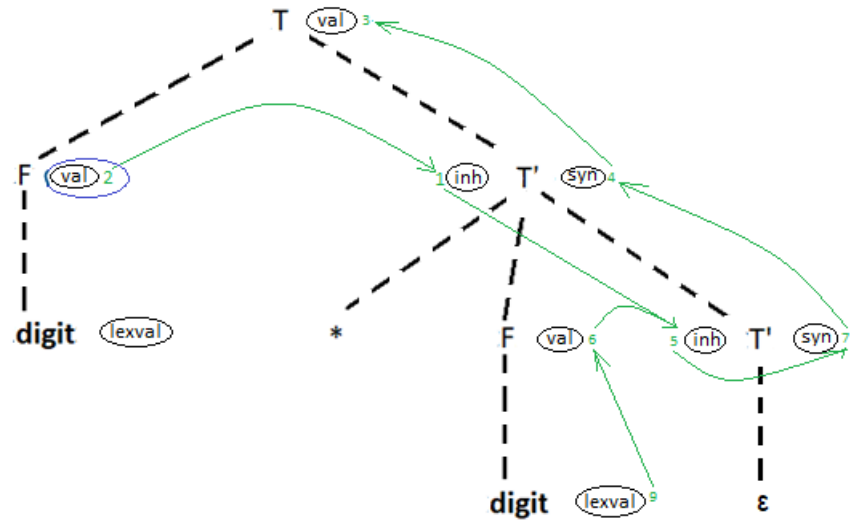
Обходя сверху вниз, слева-направо первым найдём вершину с индексом 8

б) Запоминаем индекс 8, например в конечное множество индексов узлов U, которое изначально пусто (U=∅):
U=∅ ∪{8} ={8}
и убираем этот узел из рассмотрения:

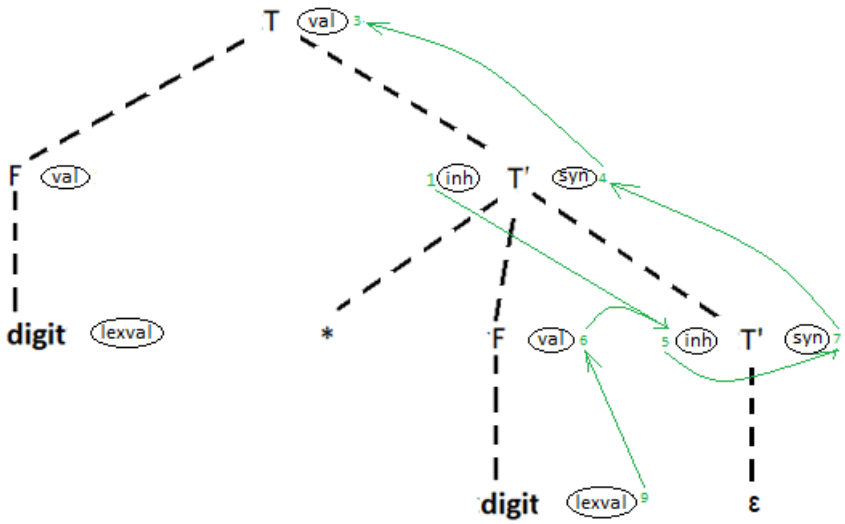


в) повторяем шаги а), б) до тех пор пока из рассмотрения не уйдут абсолютно все узлы:

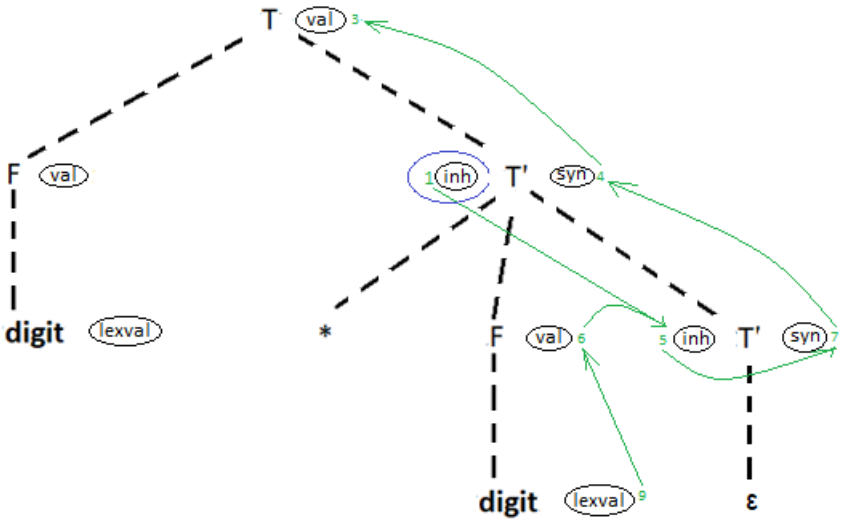
а)



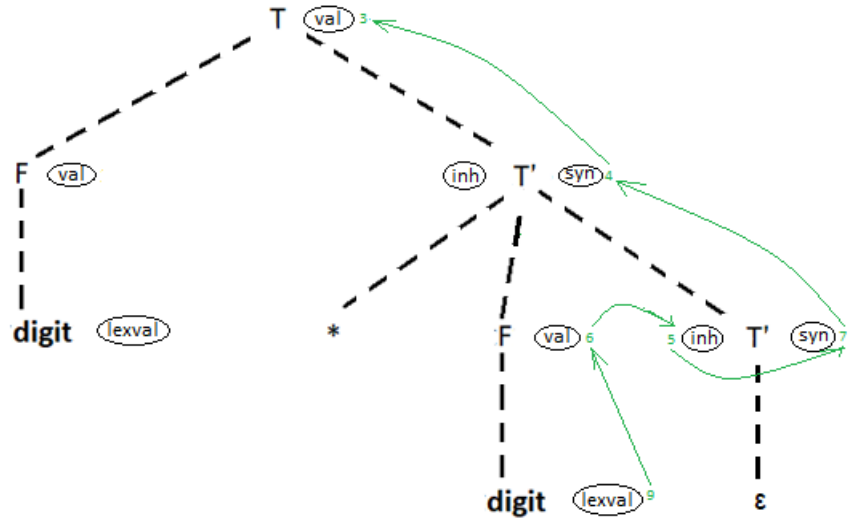
б) U = {8}∪{2} = {8,2}



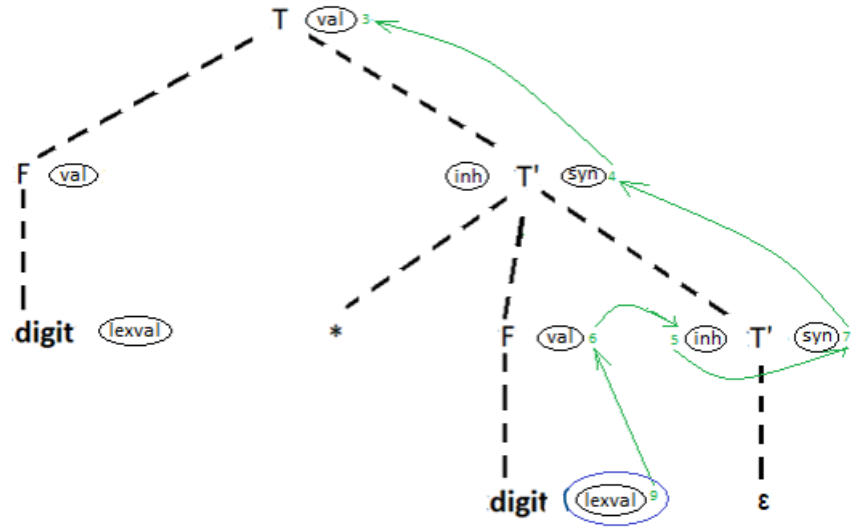
а)


T
val
3
F
val
1
inh
T'
syn
4
digit
lexval
*
F
val
6
5
inh
T'
syn
7
digit
lexval
9
ε

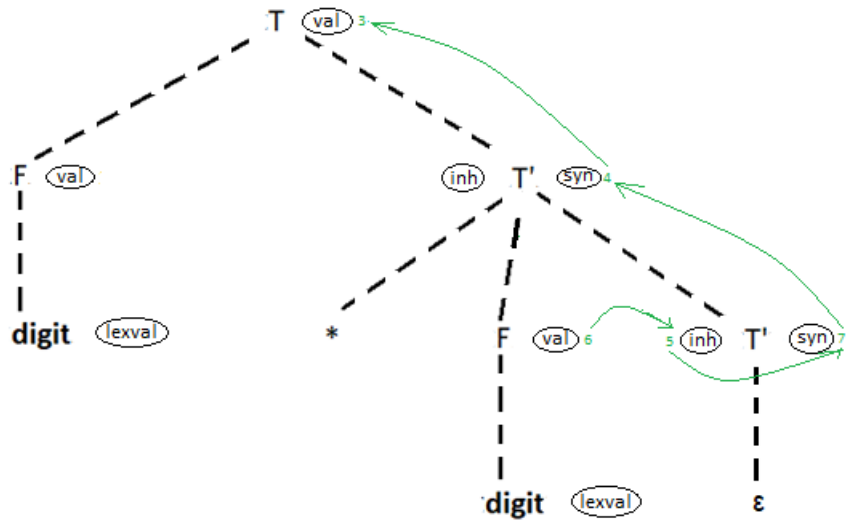
б) U = {8,2}∪{1} = {8,2,1}


T
val
3
F
val
inh
T'
syn
4
digit
lexval
*
F
val
6
5
inh
T'
syn
7
digit
lexval
9
ε

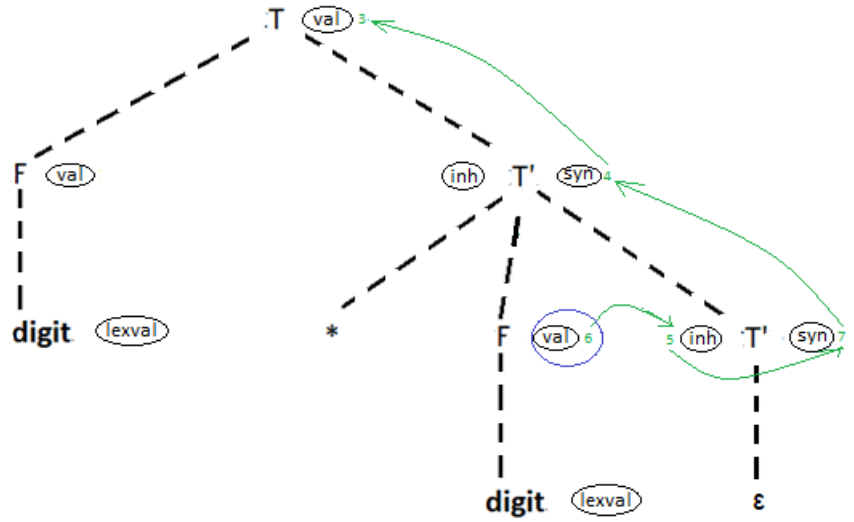
а)


T
val
3
F
val
inh
T'
syn
4
digit
lexval
*
F
val
6
5
inh
T'
syn
7
digit
lexval
9
ε

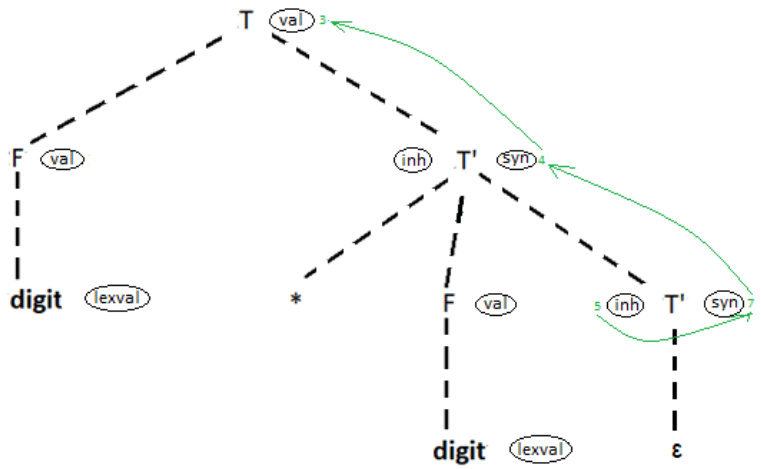
б) U = {8,2,1}∪{9} = {8,2,1,9}



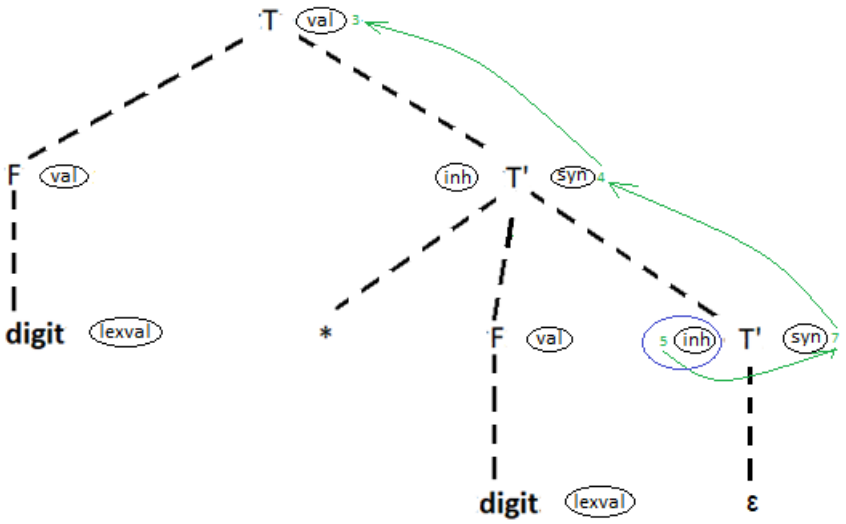
а)



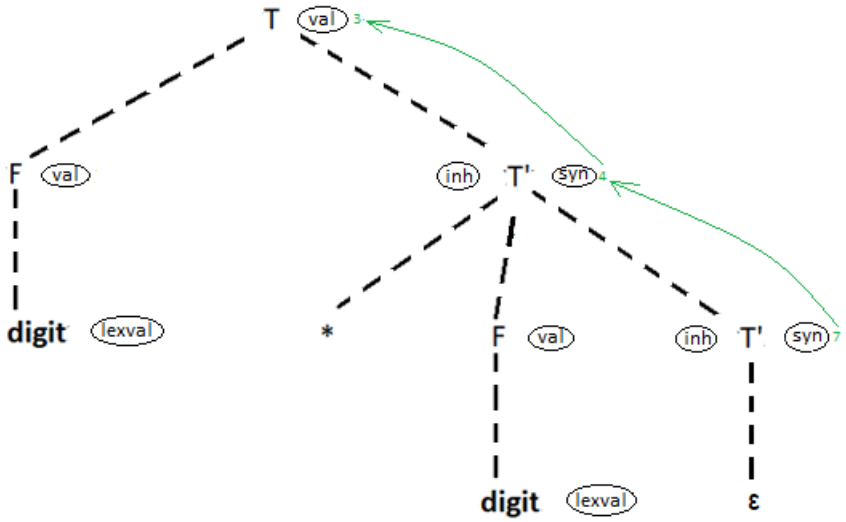
б) U = {8,2,1,9}∪{6} = {8,2,1,9,6}



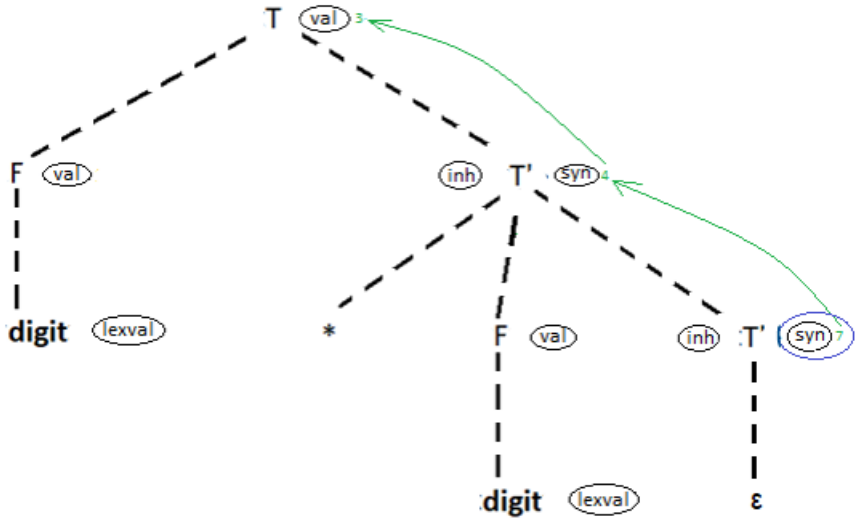
а)



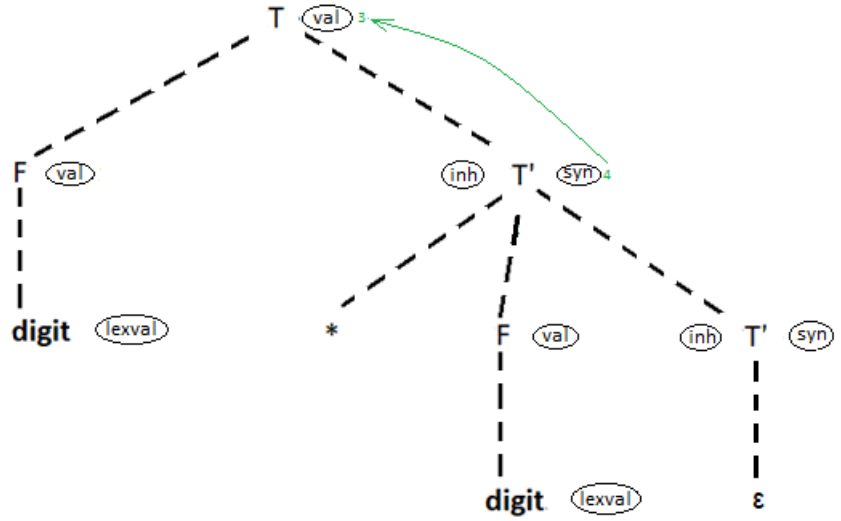
б) U = {8,2,1,9,6}∪{5} = {8,2,1,9,6,5}



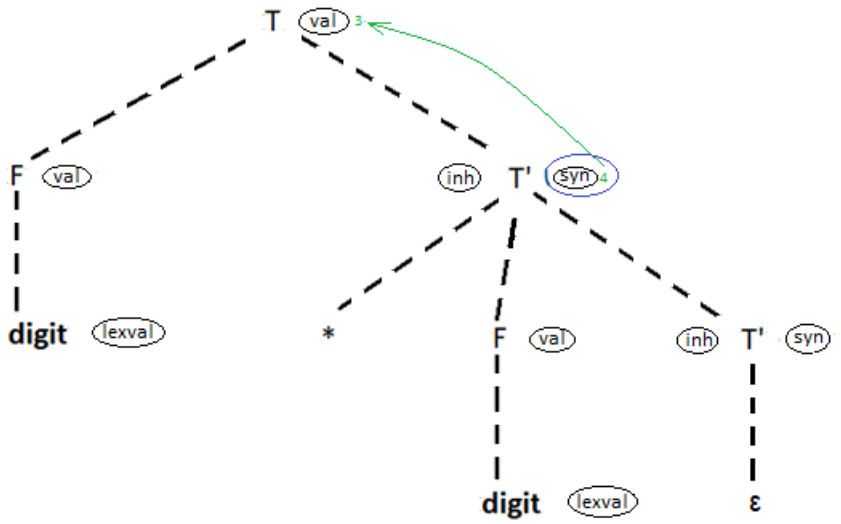
а)



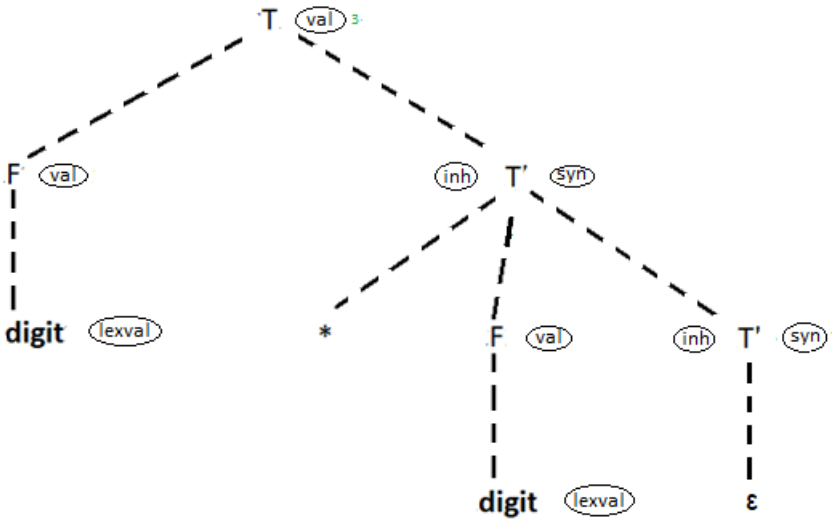
б) U = {8,2,1,9,6,5} ∪ {7} = {8,2,1,9,6,5,7}



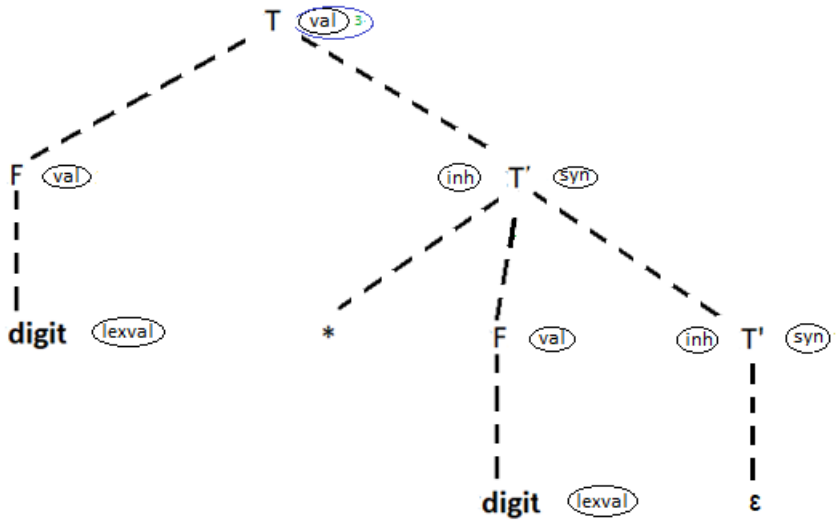
а)



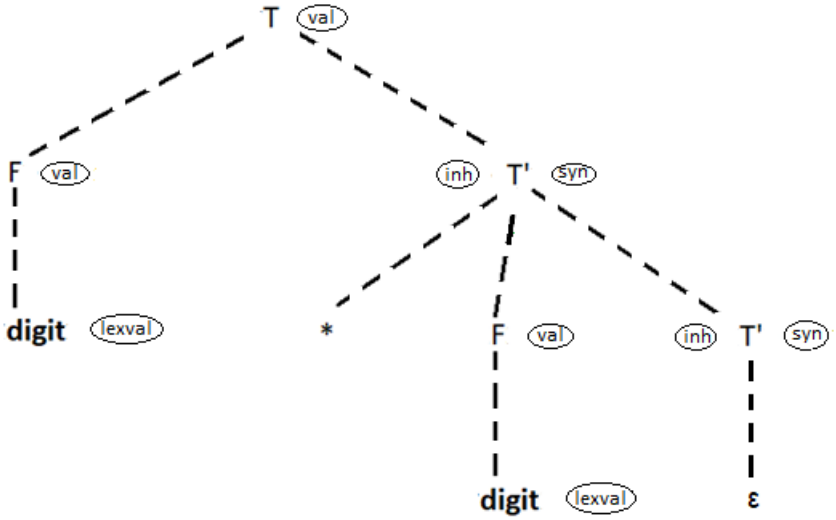
б) U = {8,2,1,9,6,5,7}∪{4} = {8,2,1,9,6,5,7,4}



а)



б) U = {8,2,1,9,6,5,7,4}∪{3} = {8,2,1,9,6,5,7,4,3}



4) Рассмотрев все узлы и убрав их из рассмотрения, мы получаем конечное множество индексов дерева U, которое хранит подходящий порядок вычисления. Для того чтобы более удобно и понятно хранить порядок, можно заменить индексацию узлов, чтобы в соответствующем узле стоял порядковый номер этого узла в конечном множестве U и так как чем раньше узел встречается в U, тем раньше он должен выполняться, то заменив индексацию таким способом мы по индексам узлов однозначно сможем сказать каким по порядку должен выполняться соответствующий узел:
U = {8,2,1,9,6,5,7,4,3}, то есть будут замены индексов:
8 станет 1 (первый индекс в множестве U)
2 останется 2 (второй индекс в множестве U)
1 станет 3 (третий индекс в U)

9 станет 4 (четвёртый индекс в U)

6 станет 5 (пятый в U)

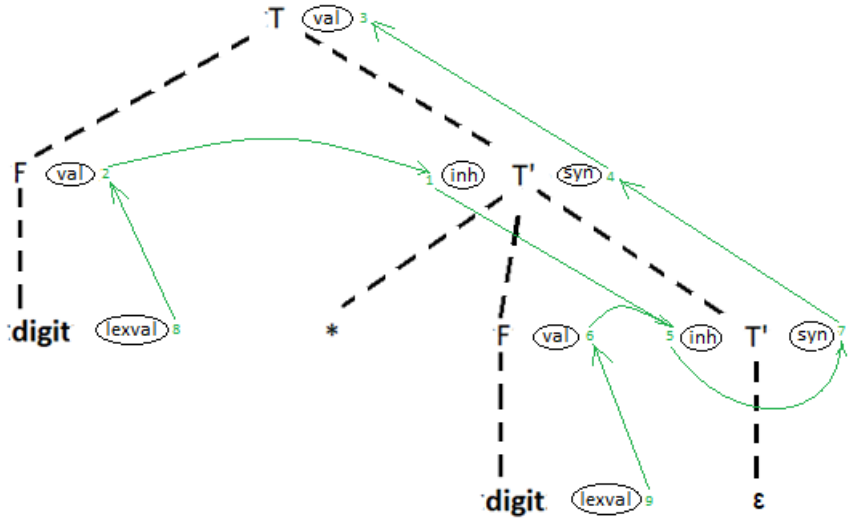
5 станет 6 (шестой в U)

7 останется 7 (седьмой в U)

4 станет 8 (восьмой в U)

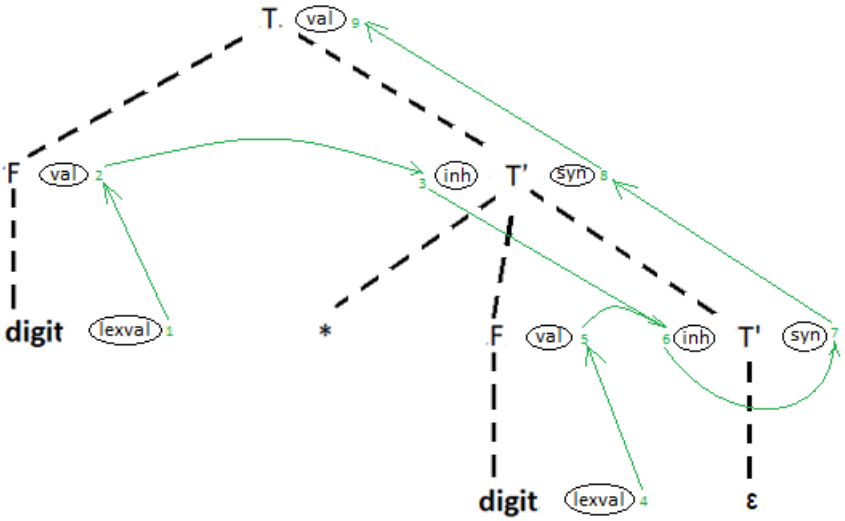
3 станет 9 (девятый в U)

И дерево зависимостей из такого:



5)

Станет таким:



По которому однозначно определяется порядок зависимостей.