# Организация лексического анализа

- Назначение и необходимость фазы лексического анализа.
- Транслитератор.
- Грамматики и распознаватели для лексического анализа.
- Методы лексического анализа.

# *Назначение фазы лексического анализа*

Лексический анализ - разбиение входного потока на лексемы.

- класс лексемы, определяющий общее название для группы элементов, обладающих общими свойствами
- значение лексемы, определяющее подстроку символов входной цепочки, соответствующих распознанному классу лексемы.

В зависимости от класса, значение лексемы может быть преобразовано во внутреннее представление уже на этапе лексического анализа. Так, например, поступают с числами, преобразуя их в машинное представление. Это обеспечивает их более компактное хранение и позволяет проверить правильность диапазона чисел на ранней стадии анализа.

# *Необходимость фазы лексического анализа*

1. Замена, цепочек символов, представляющих элементарные конструкции языка, делает внутренне представление программы более удобным для дальнейшего анализа распознавателем.
2. Уменьшается длина программы за счет устранения из нее несущественных для дальнейшего анализа пробелов, комментариев, игнорируемых символов.
3. Один и тот же язык программирования может иметь различные внешние представления элементарных конструкций.
4. Лексический анализатор использует более простые, по сравнению с синтаксическим анализатором, методы разбора.
5. Блок лексического анализа естественным образом вписывается в иерархическую структуру компилятора.

# Транслитератор

Устройство, осуществляющее сопоставление класса с каждым отдельным символом, называется **транслитератором**.

**Наиболее типичными классами символов являются:**

- **буква** - класс, с которым сопоставляется множество букв, причем необязательно только одного алфавита;
- **цифра** - множество символов, относящихся к цифрам, чаще всего от 0 до 9;
- **разделители** - пробел, перевод строки, возврат каретки перевод формата;
- **игнорируемые** - могут встречаться во входном потоке, но игнорируются и поэтому просто выбрасываются из него (например, невидимый код звукового сигнала и другие аналогичные коды);
- **запрещенные** - те символы, которые не относятся к алфавиту языка, но встречаются во входной цепочке;
- **остальные** - символы, не вошедшие ни в одну из определенных категорий.

# Грамматики и распознаватели для лексического анализа

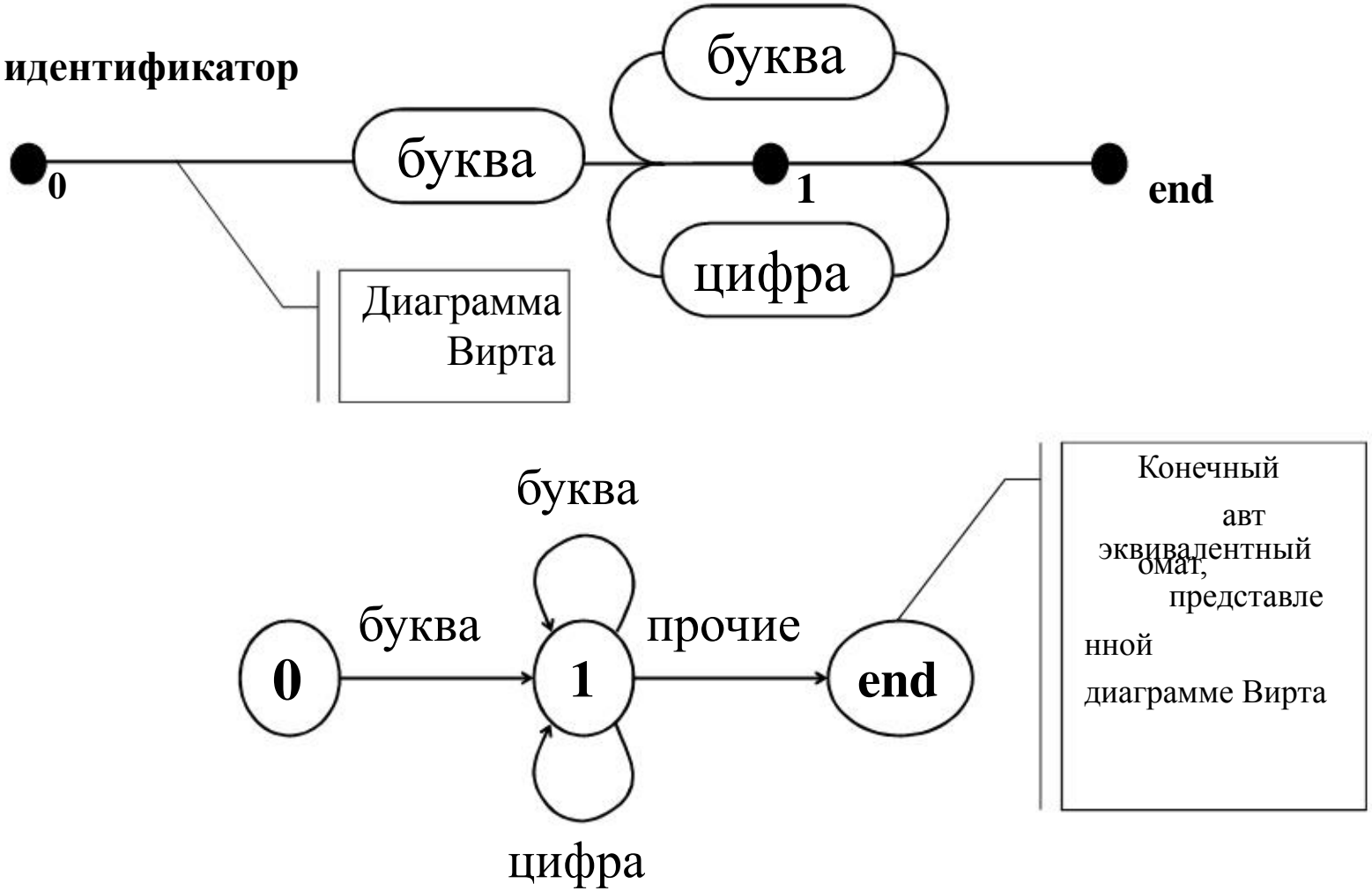
Праволинейные грамматики - конечный автомат

**Построение конечного автомата можно осуществить с использованием и некоторых грамматик с *левой рекурсией*, а также *диаграмм Вирта*.**

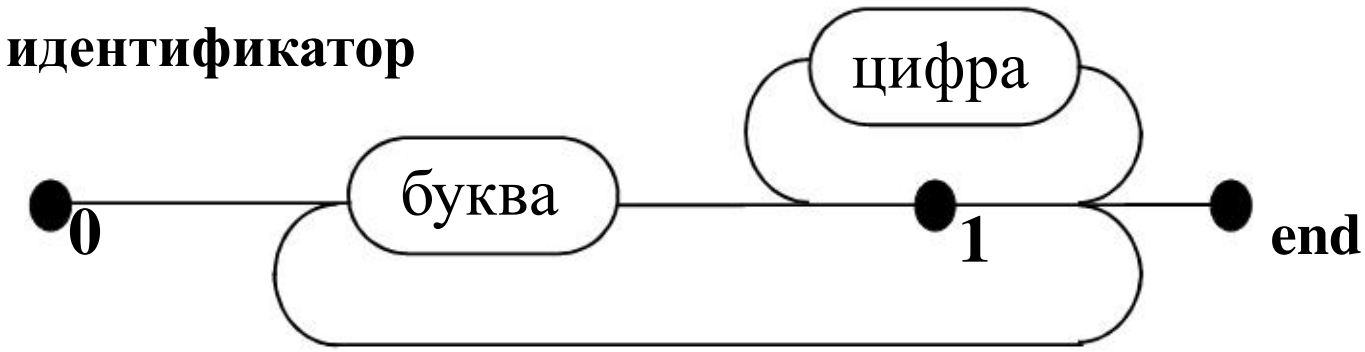
Диаграммы Вирта намного нагляднее при описании правил.
Между диаграммами Вирта, не содержащими нетерминалов, и конечными автоматами существует однозначное соответствие.

***Построение конечного автомата, по диаграмме Вирта, без нетерминалов:***

1. Вход входной дуги диаграммы преобразуется в начальное состояние конечного автомата.
2. Выход выходной дуги диаграммы образует заключительное состояние конечного автомата.
3. Выходы отдельных дуг, соединяющих символы, и точки ветвления остальных дуг диаграммы образуют множество остальных состояний конечного автомата.
4. Конечные состояния диаграммы являются допускающими состояниями конечного автомата.
5. Терминальные символы диаграммы Вирта расположенные между соответствующими дугами, преобразуются в связи между соответствующими состояниями, с входным символом, соответствующим указанному терминалу.
6. Связи, обеспечивающие переход в допускающие состояния, помечаются множеством оставшихся символов. Это множество не пересекается с множеством символов, обеспечивающих переходы из текущего в другие состояния (обозначены как “прочие”).




Преобразование диаграммы Вирта в эквивалентный конечный автомат.

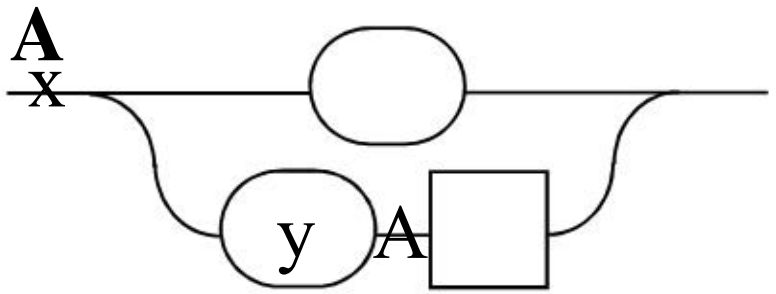



Минимизированная диаграмма Вирта, задающая идентификатор.

# Связь между диаграммами Вирта и праволинейными грамматиками. Преобразование правой рекурсии в итерацию

$$A \rightarrow x$$
$$A \rightarrow yA$$

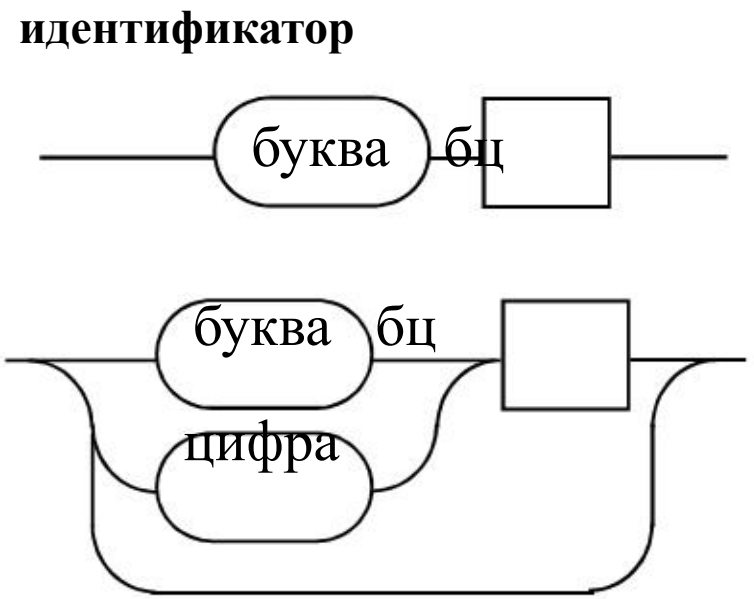
$\Rightarrow$




Непосредственное преобразование простой грамматики

**$ идентификатор = буква бц .**

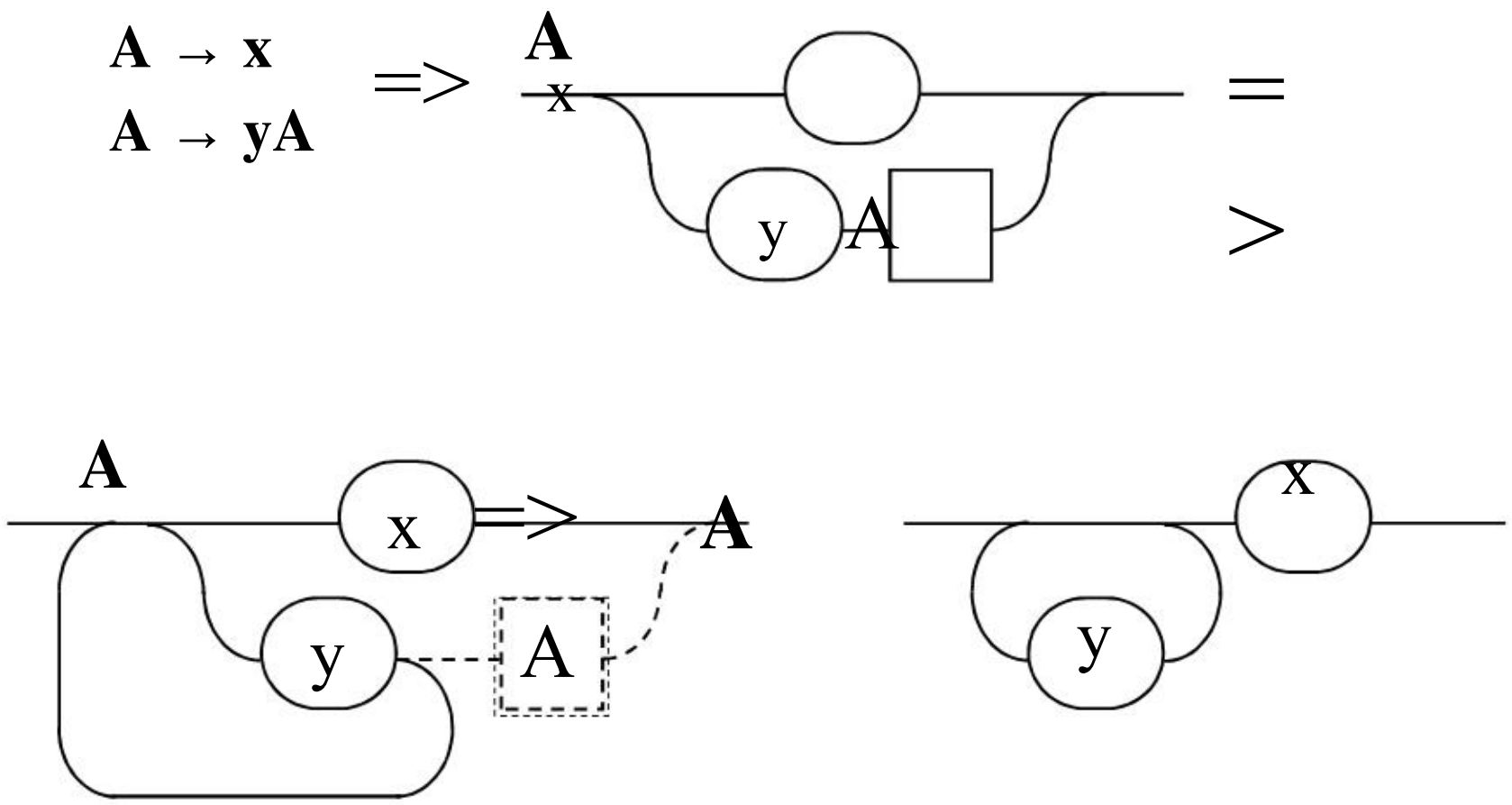
**$ бц = буква бц | цифра бц | .**




Непосредственное преобразование идентификатора

**Замена правой рекурсии на итерацию в диаграммах Вирта:**

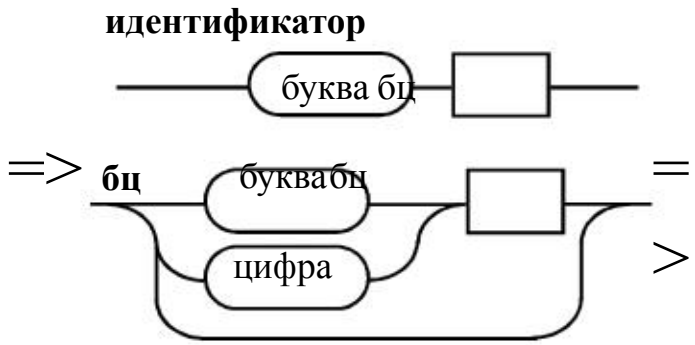
- дуга, входящая в рекурсивно определяемый нетерминал, замыкается своим выходом на самое начало правила, образуя, таким образом, цикл;
- сам нетерминал, оказавшийся в подвешенном состоянии вычеркивается, а выходившая из него дуга убирается.

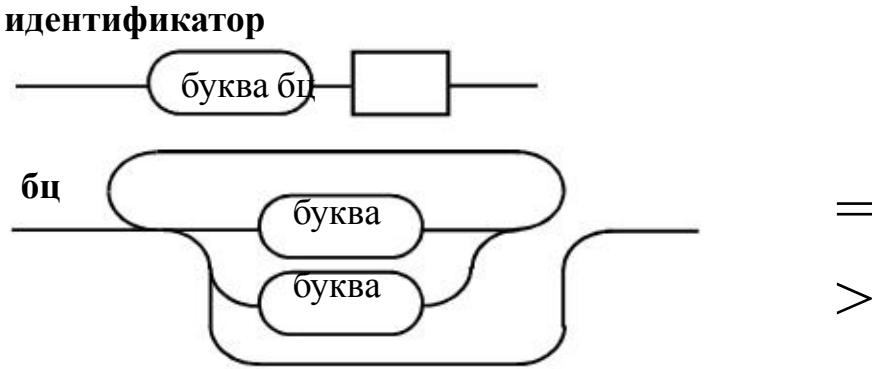


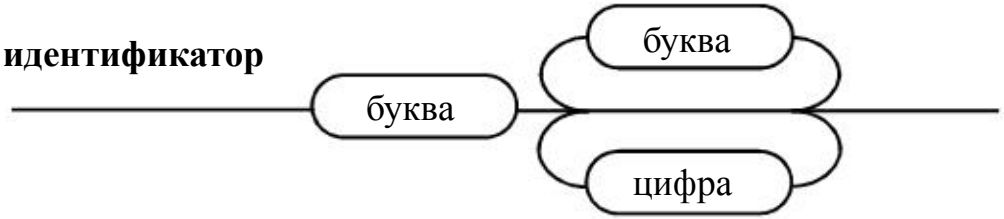

Преобразование праволинейной грамматики в итеративную диаграмму Вирта.

**$ идентификатор = буква бц .**

**$ бц = буква бц | цифра бц | .**



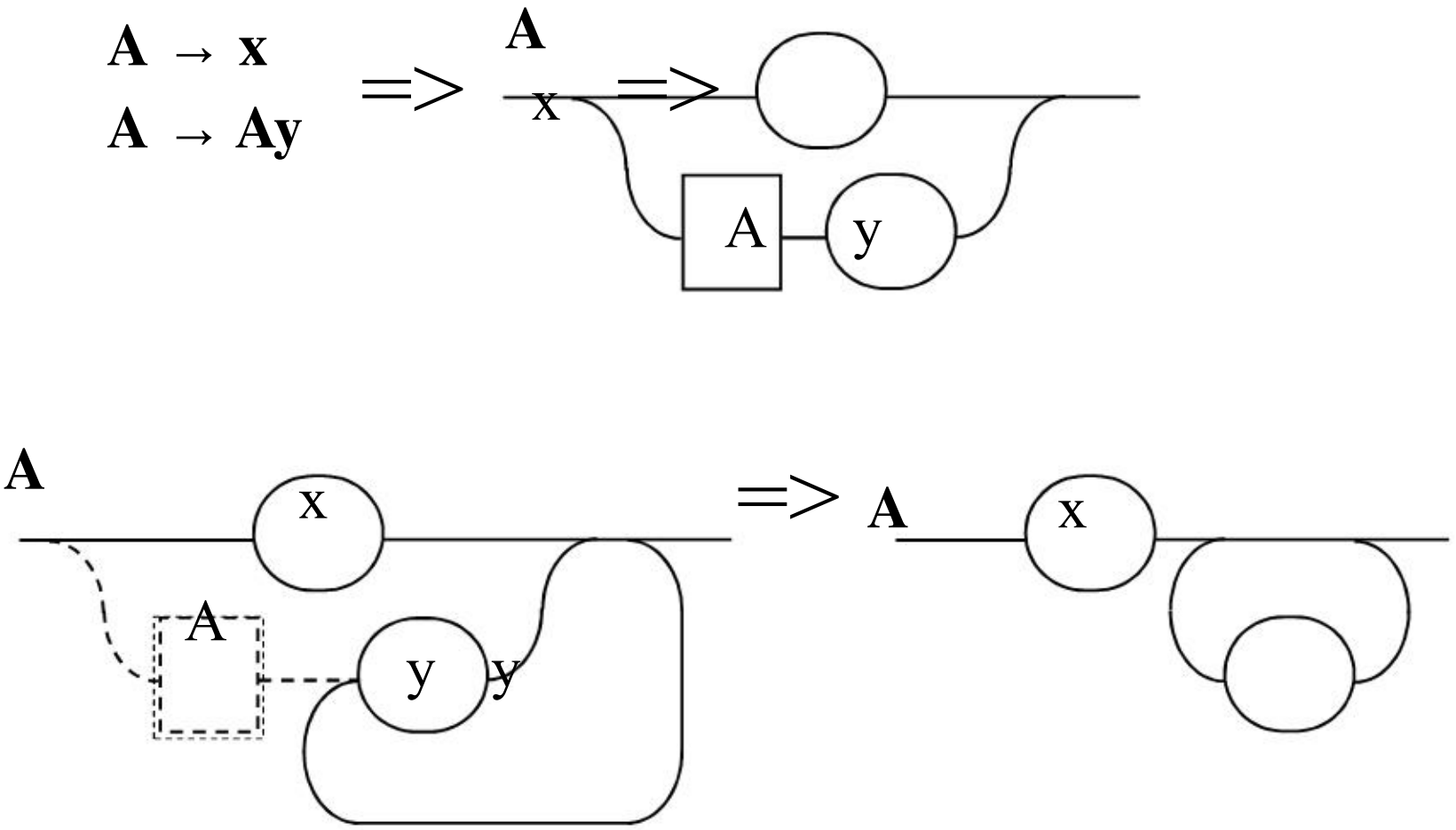




Преобразование праволинейной грамматики идентификатора в итеративную диаграмму Вирта.

# Связь между диаграммами Вирта и грамматиками с левой рекурсией. Преобразование левой рекурсии в итерацию

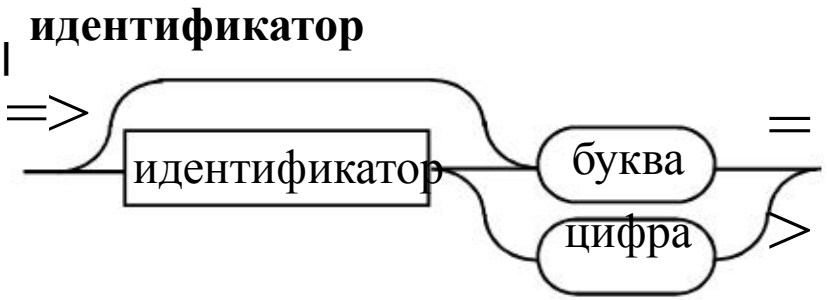
**Преобразования в итерацию для правил, содержащих левую рекурсию:**

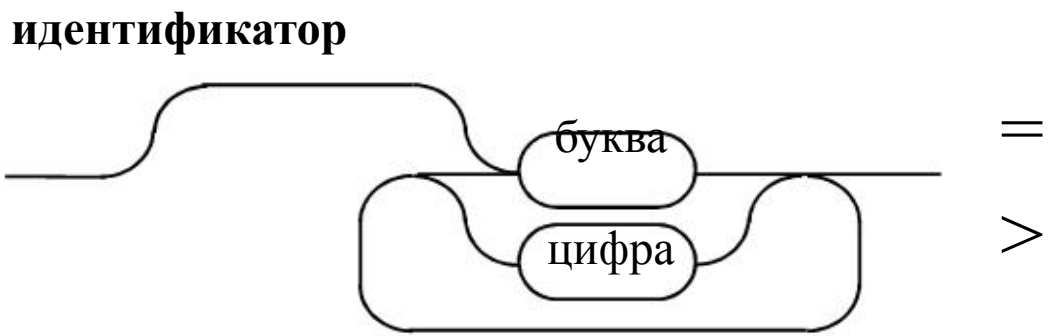
- дуга, **выходящая** из рекурсивно определяемого нетерминала, замыкается своим **входом** на самый конец правила, образуя, таким образом, цикл;
- сам нетерминал, оказавшийся без адресата, вычеркивается, а входившая в него дуга убирается.

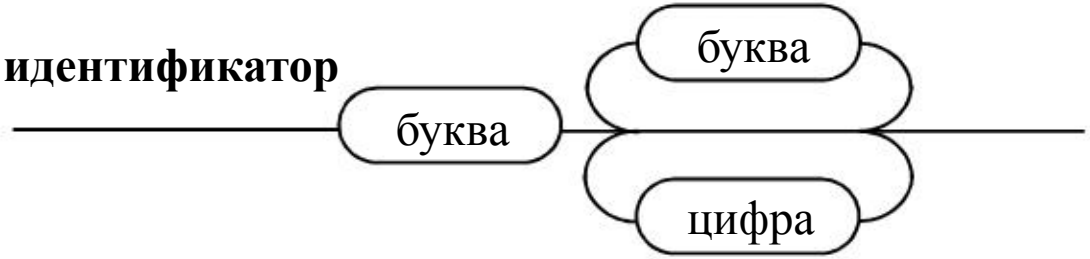



Преобразование грамматики с левой рекурсией в итеративную диаграмму Вирта

**$ идентификатор = буква |**
**идентификатор буква |**
**идентификатор цифра.**



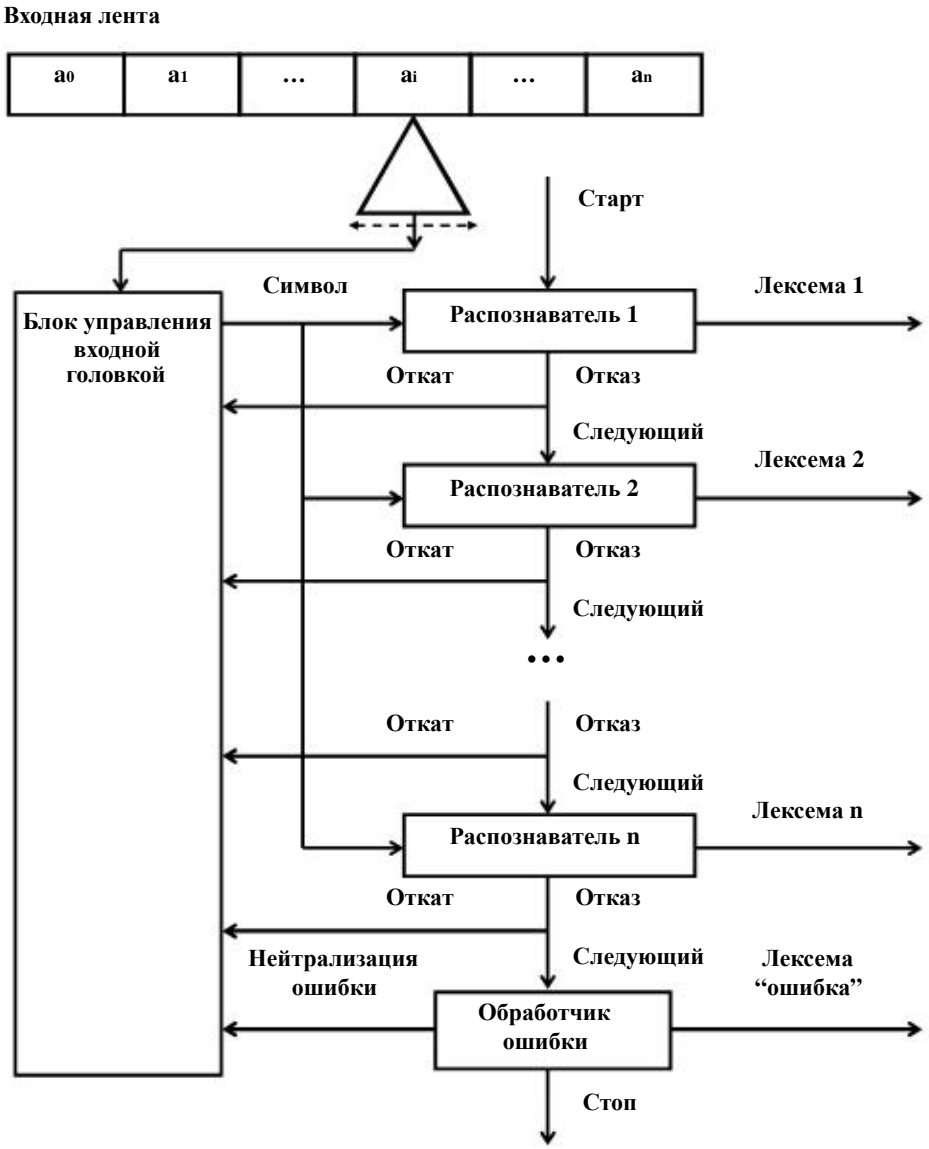




Преобразование грамматики идентификатора с левой рекурсией в итеративную диаграмму Вирта.

# Методы лексического анализа

**Непрямой лексический анализ**, или **лексический анализ с возвратами**, заключается в последовательной проверке версий о классах лексем. Если проверка текущей версии не подтверждается, то происходит откат назад по цепочке символов и осуществляется проверка следующей версии.

**Прямой лексический анализ** позволяет определить значение лексемы без откатов назад по цепочке символов.




Обобщенная структура непрямого лексического анализатора

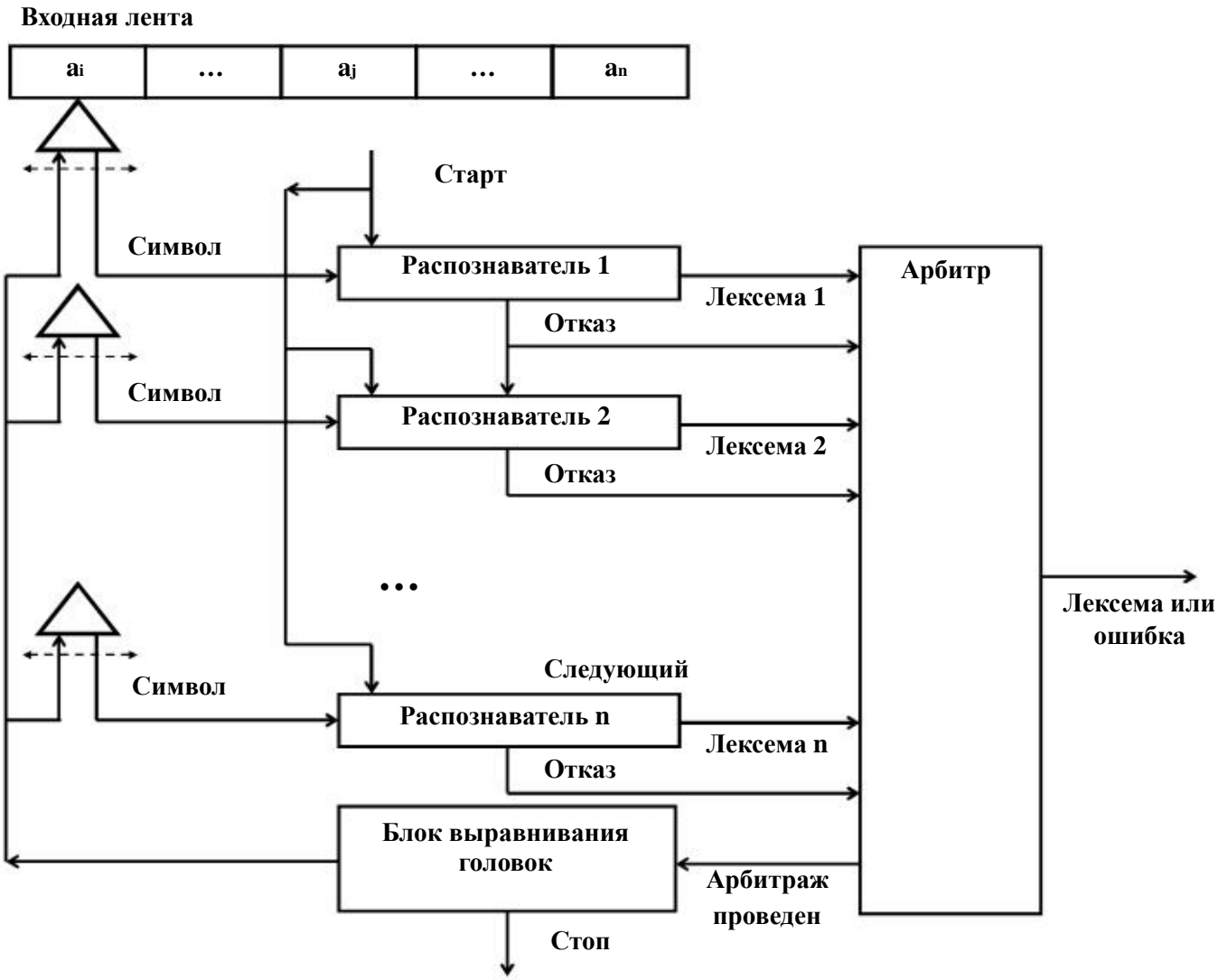
**К достоинствам** непрямого лексического анализатора следует отнести:

- прозрачность общей регулярной структуры, которая легко может изменяться и наращиваться;
- простота каждого отдельно автомата, распознающего одну достаточно элементарную конструкцию;
- практическая применимость подхода в самых разнообразных языках, независимо от сложности выделения в нем тех или иных лексем.

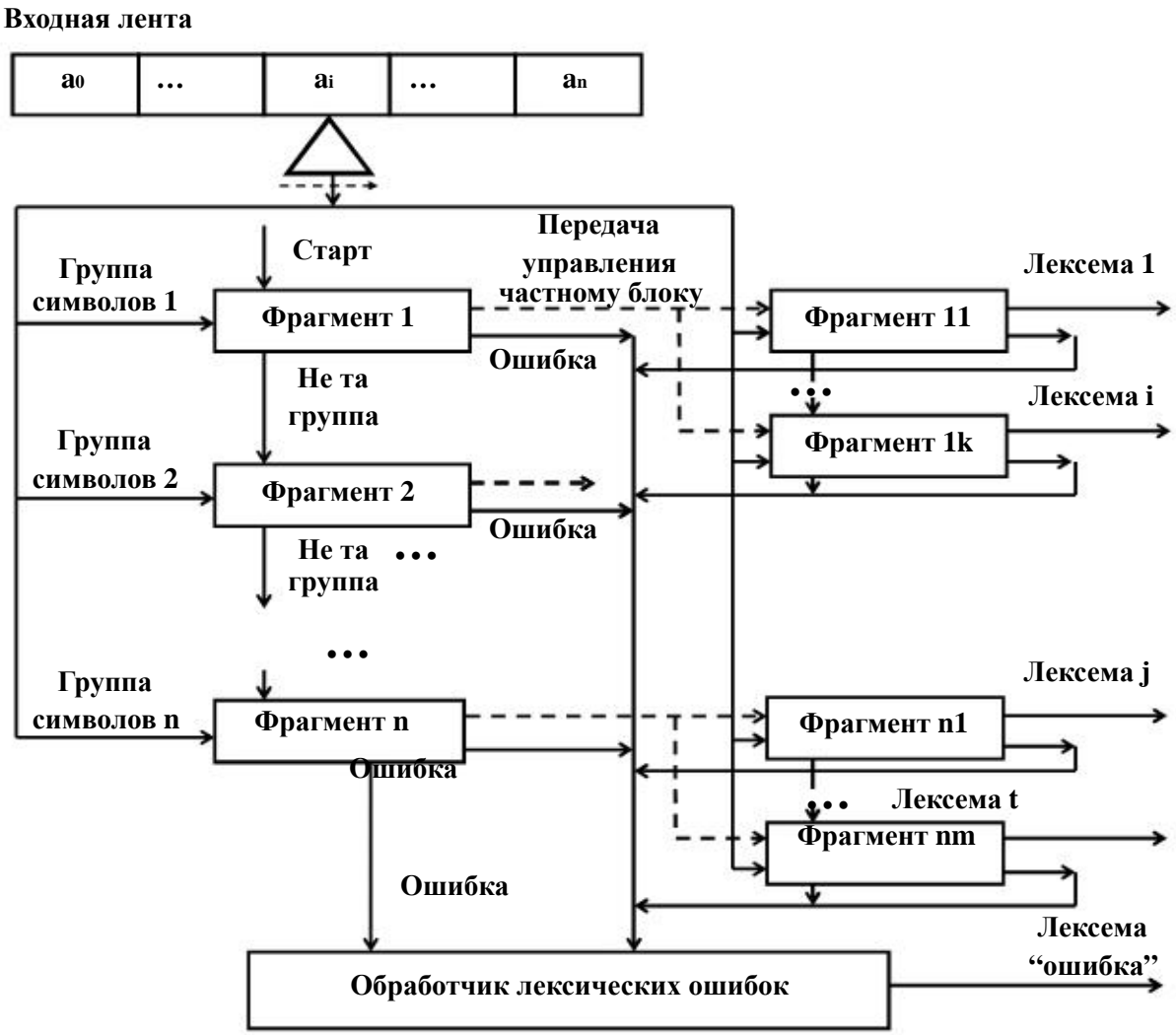
Пример: ***DOI=3,10,3***

***DO I = 3, 10, 3 или DOI=3 ?***

**К недостаткам** непрямого лексического анализатора следует отнести низкую скорость распознавания правильных лексем. Это связано с постоянными возвратами входной головки к исходному состоянию, если версия о лексеме не подтверждается.




Структура лексического анализатора с параллельной работой распознавателей




Обобщенная структура прямого лексического анализатора

# Лексический анализатор демонстрационного языка программирования

Содержание

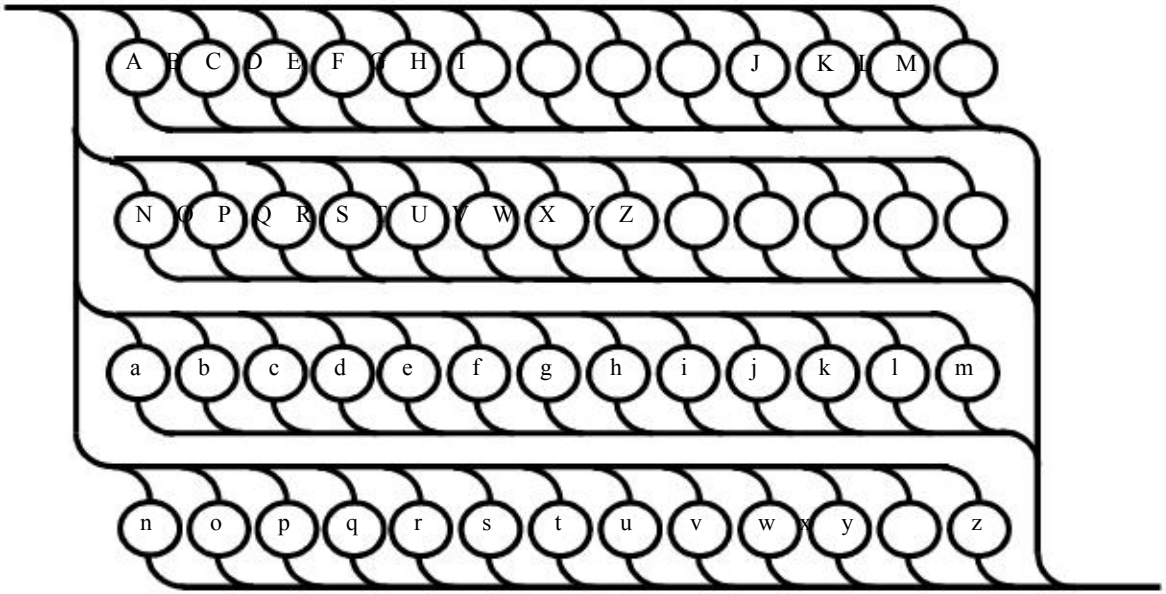
Транслитератор DPL.
Непрямой лексический анализатор DPL.

Прямой лексический анализатор DPL.

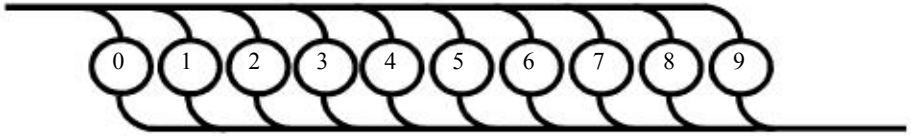
# Общая организация транслитератора

1. **Класс букв**: прописные и строчные буквы латинского алфавита
2. **Класс цифр**: объединяет арабские цифры от 0 до 9.
3. **Класс пропусков**: состоит из пробела, перевода строки, табуляции, перевода формата (разделяющего текст на отдельные страницы).
4. **Класс игнорируемых символов**: включает все символы, которые, как предполагается, не отображаются в окне текстового редактора.
5. **Класс прочих символов**: включает все оставшиеся символы.

**tltLetter**



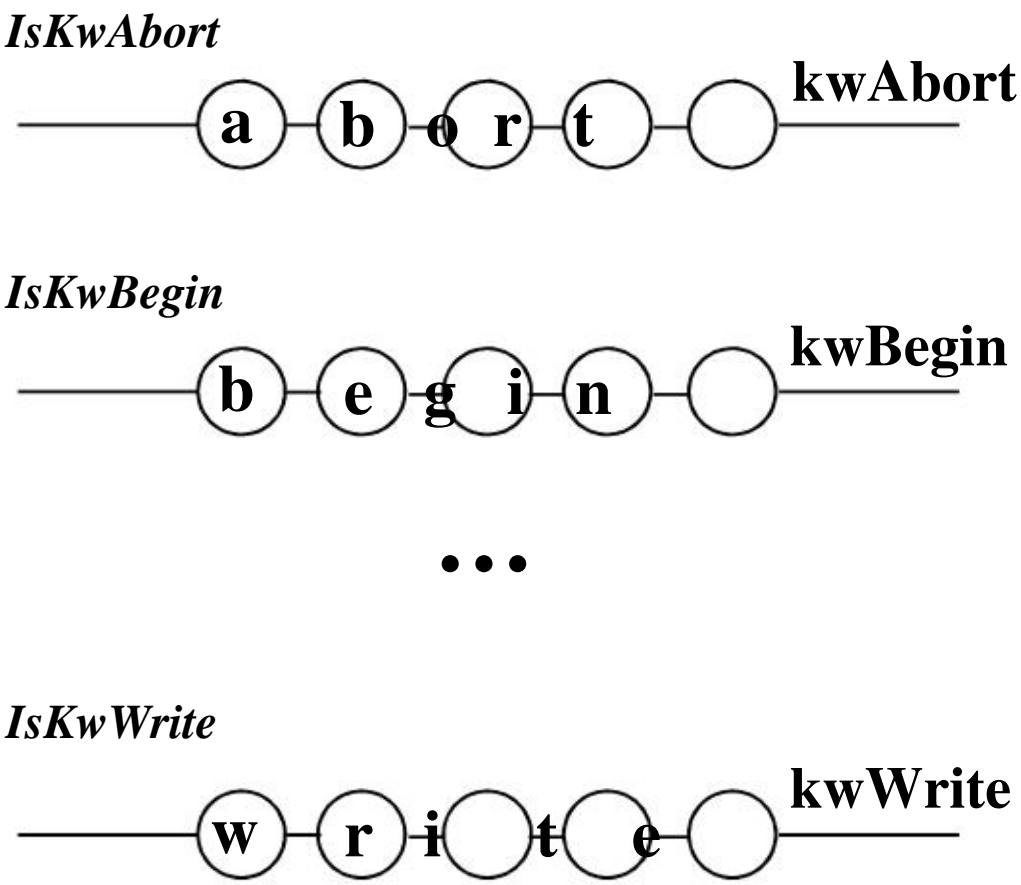

**tltDigit**



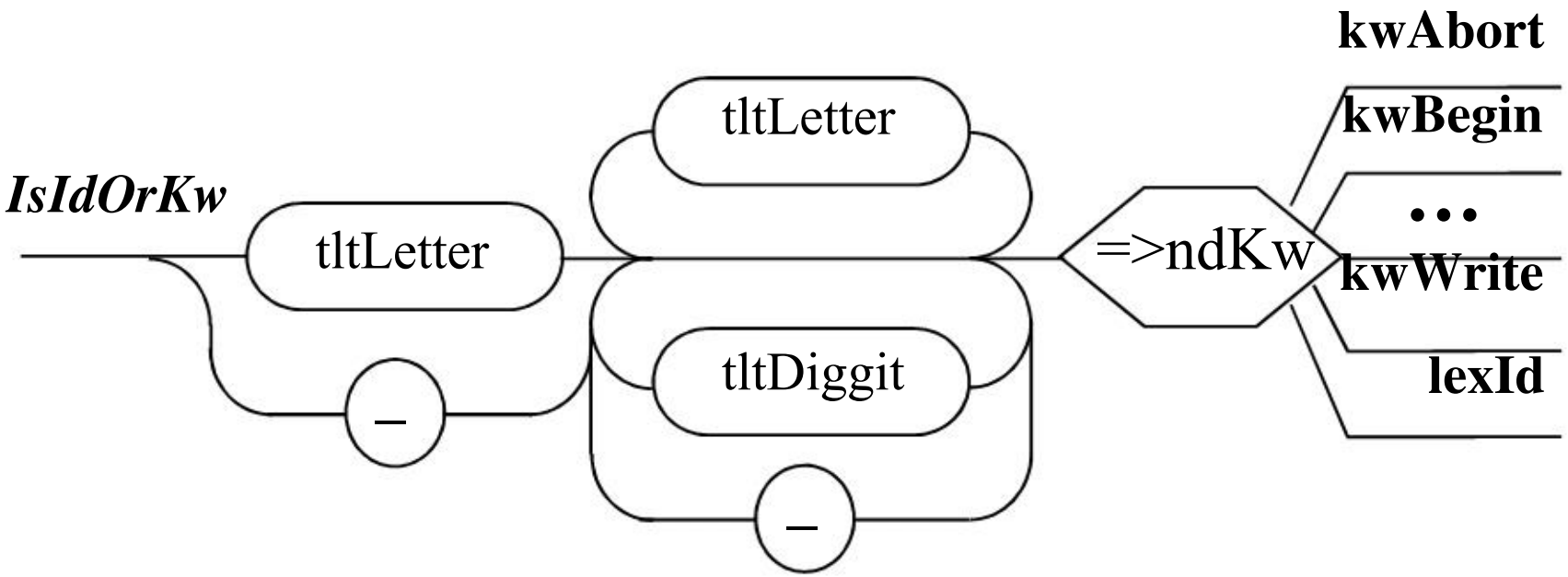

**tltSkip**


Диаграммы Вирта, задающие некоторые из классов символов, порождаемых

# Разработка непрямого лексического анализатора

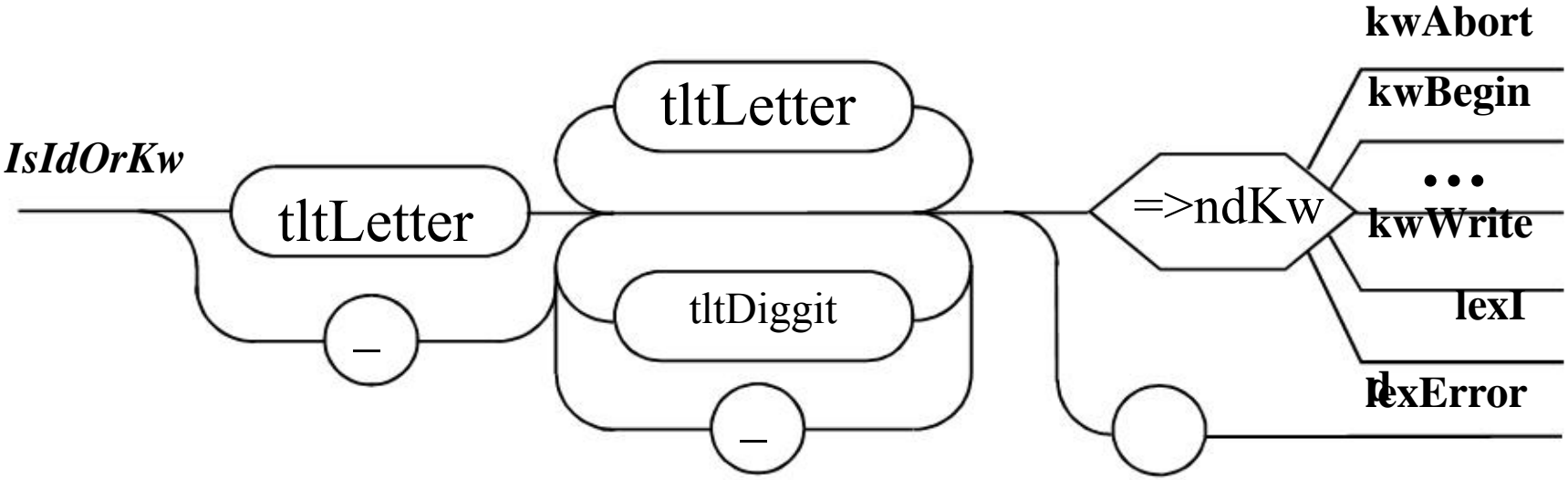



а) Непосредственный анализ ключевых слов.



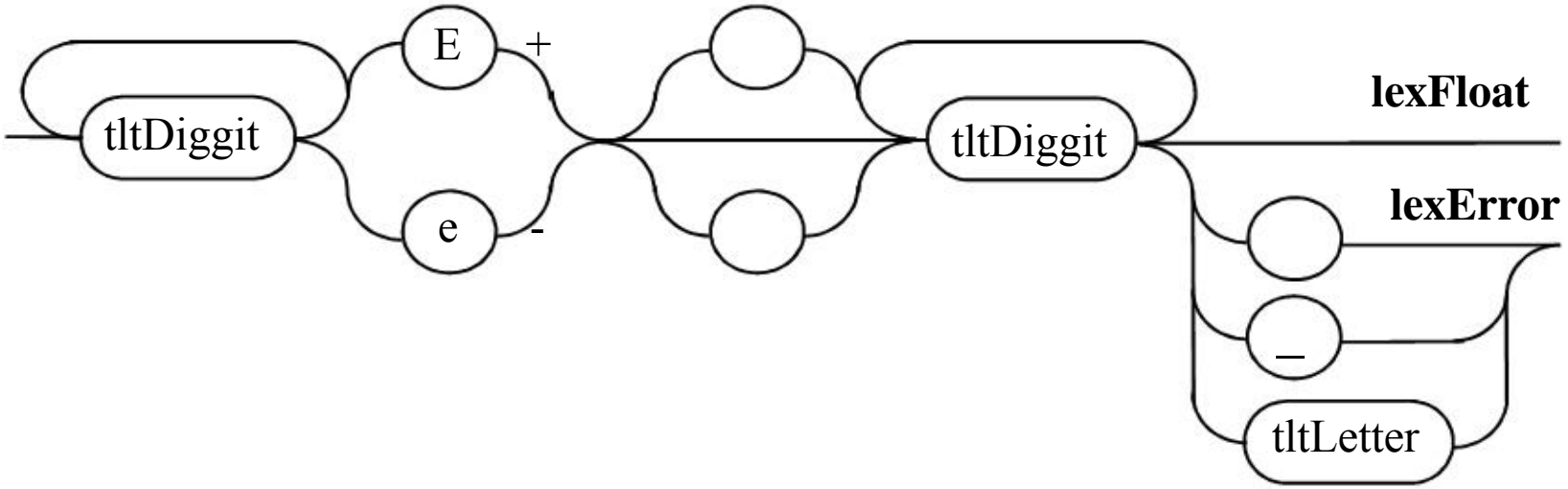

б) Семантическое выделение ключевых слов из анализа идентификатора


*Примечание 1*. Лексема, порождаемая при достижении конца обрабатываемого текста.

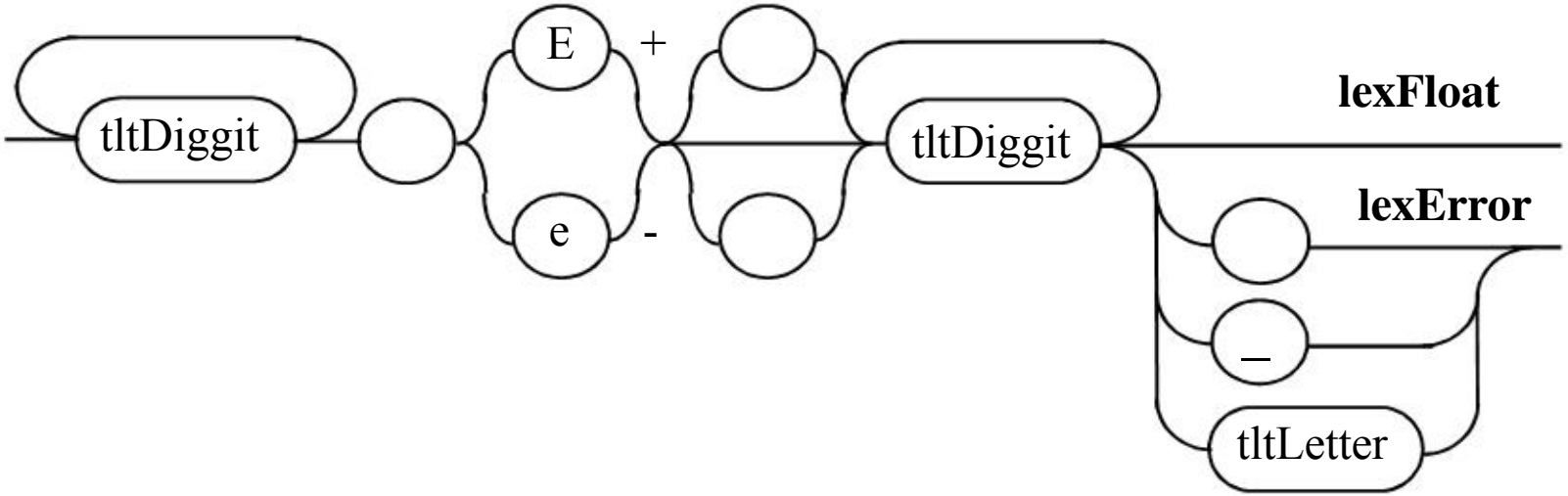



*Примечание 2.* Идентификатор и ключевые слова описываются правилом с семантической вставкой. Осуществляется также анализ на недопустимость возможного слияния идентификатора с действительным числом, начинающимся с точки.

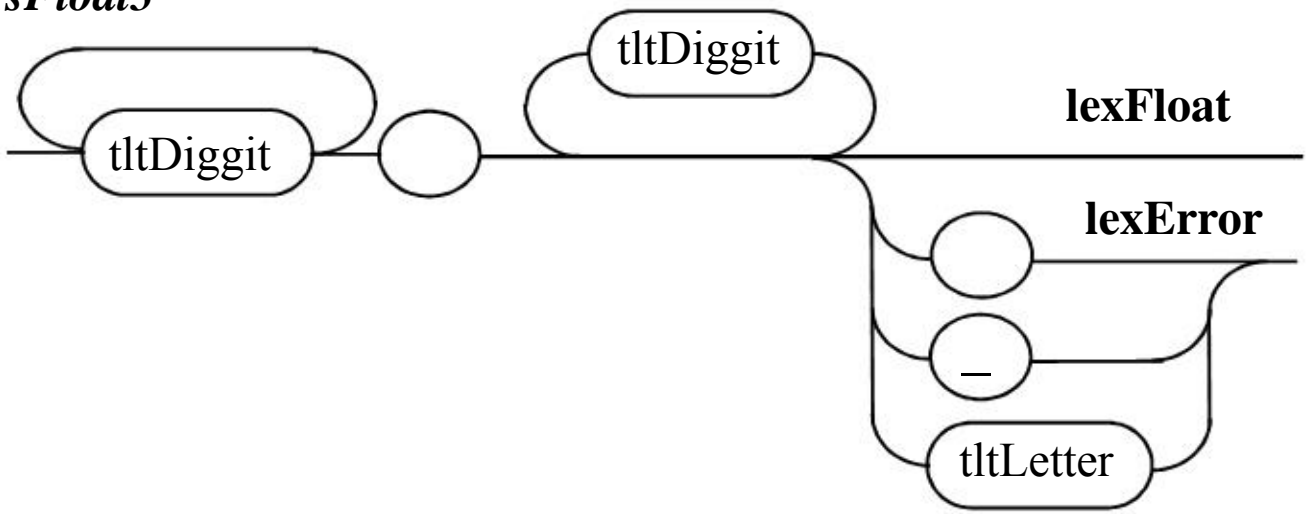
***IsFloat1***



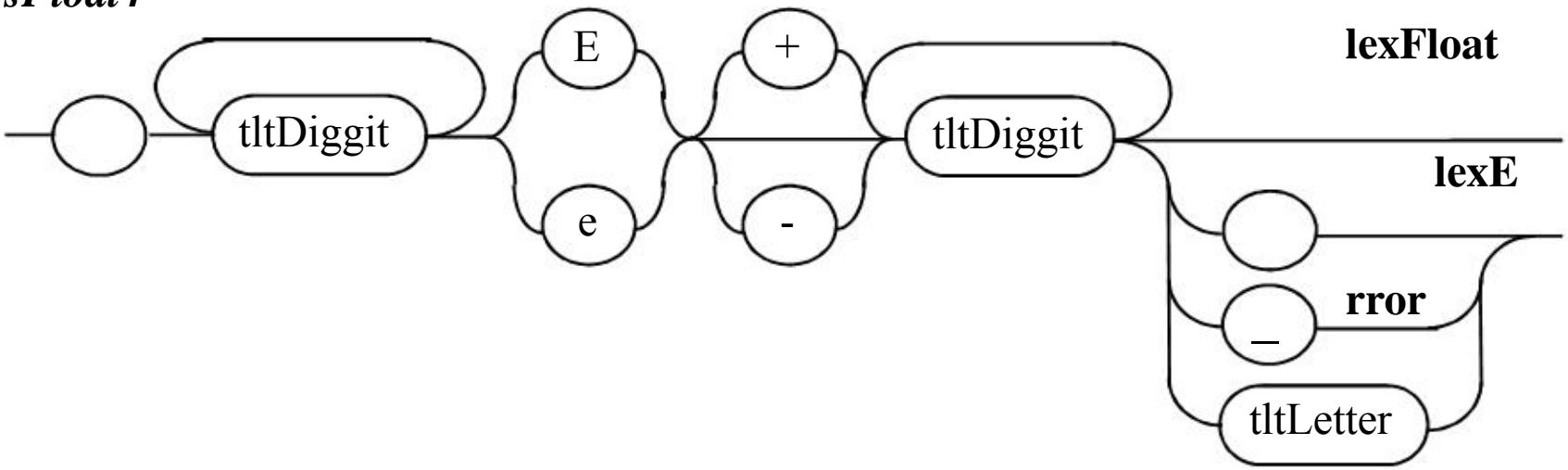

***IsFloat2***



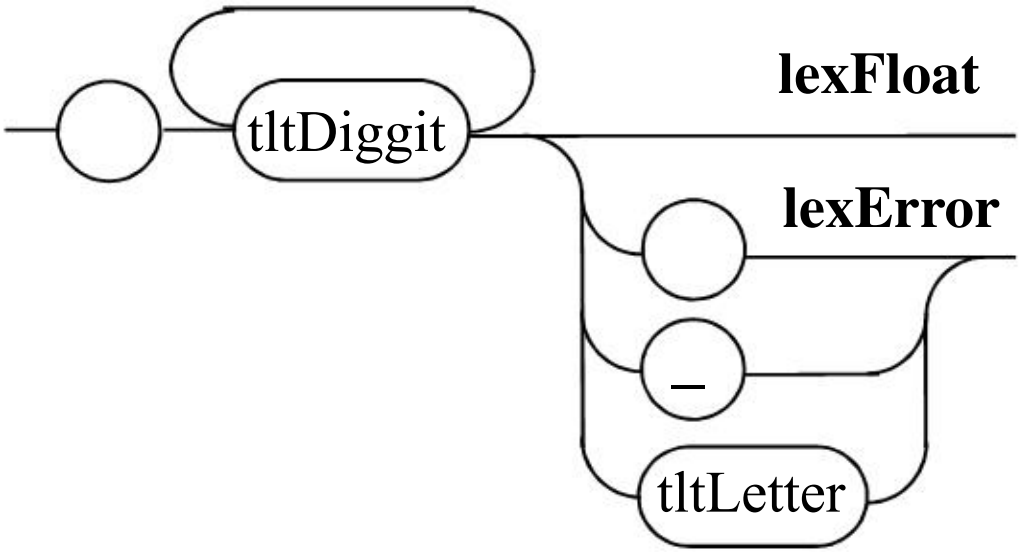
***IsFloat3***



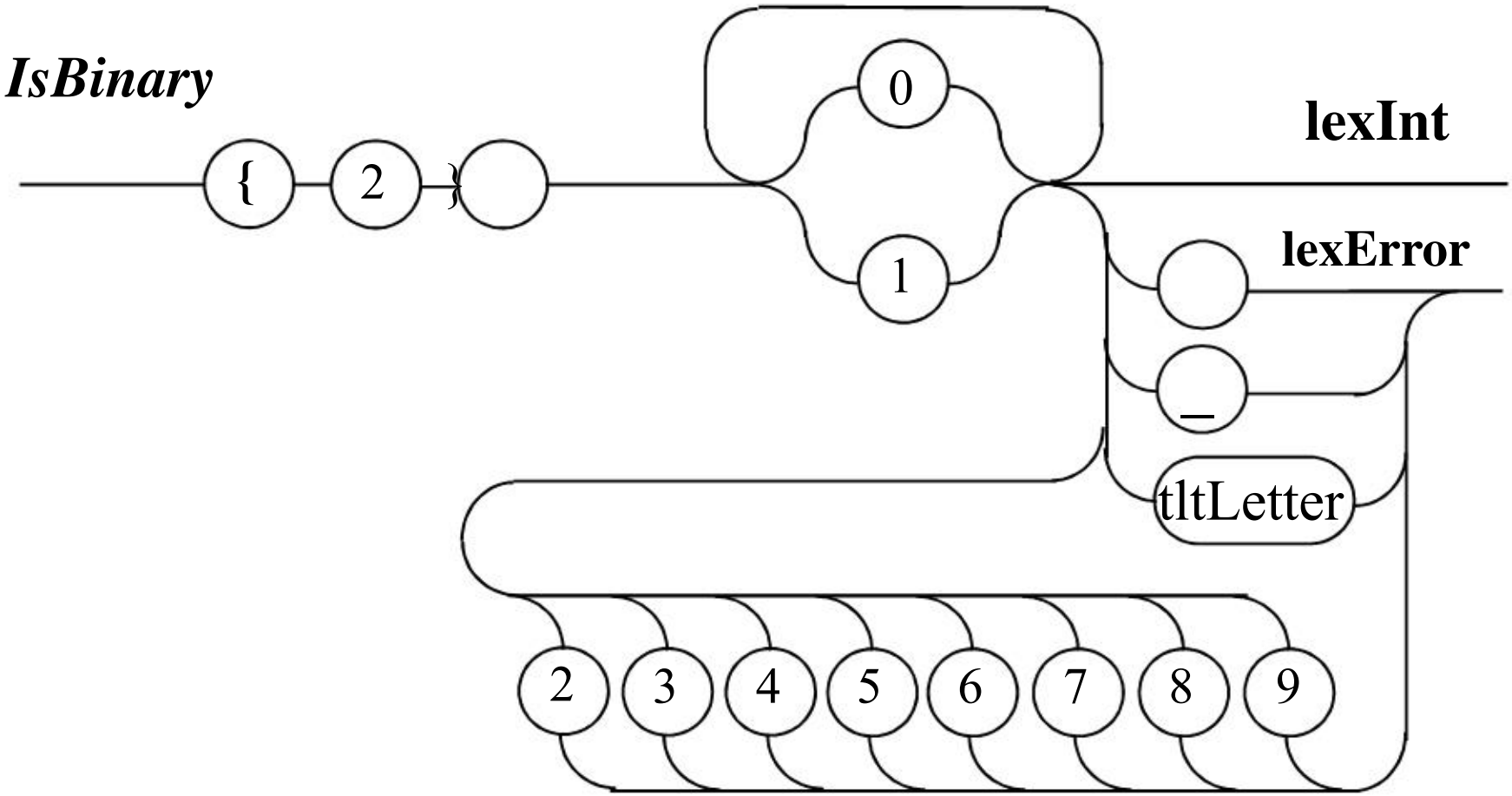

***IsFloat4***

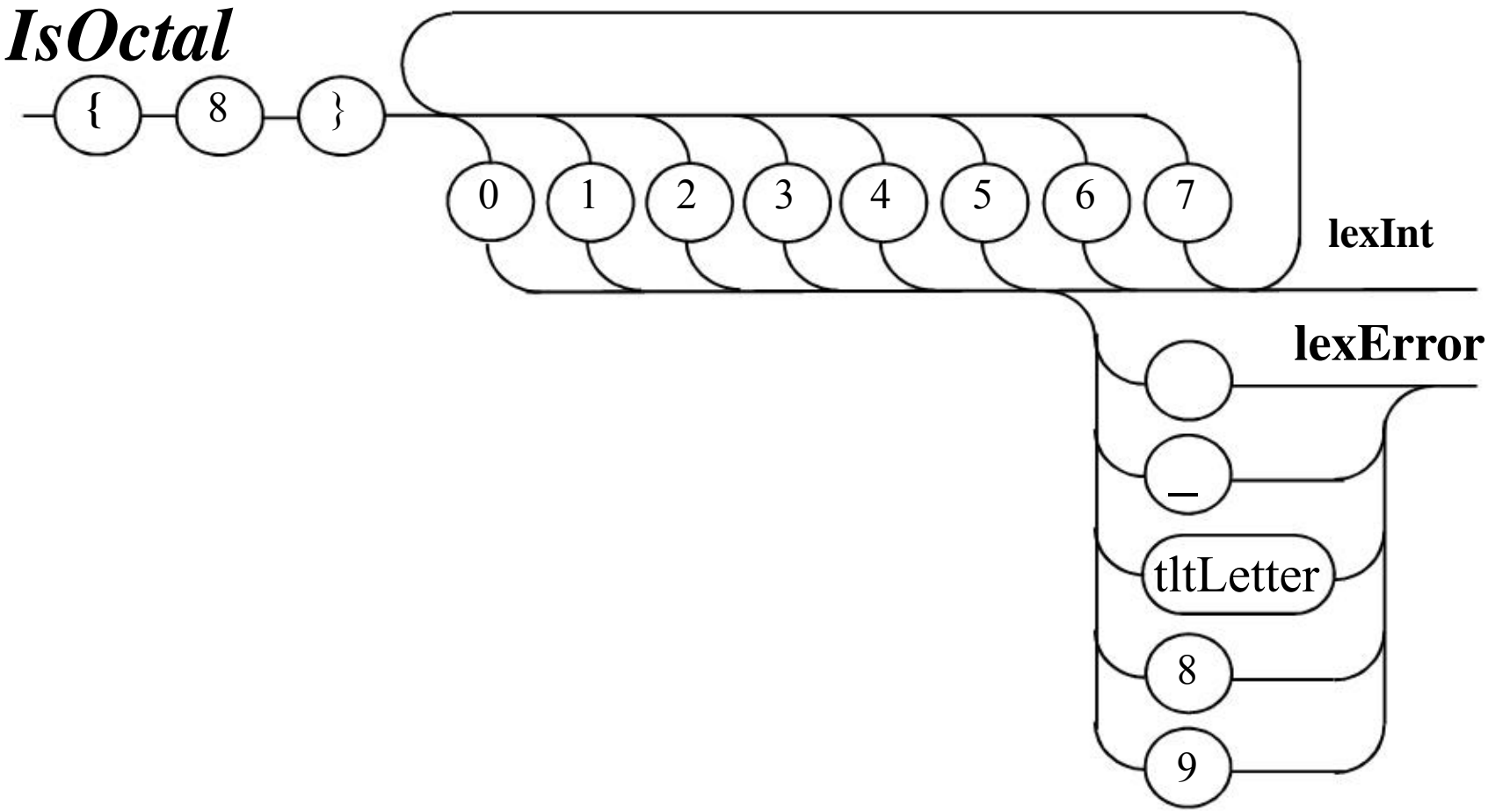


***IsFloat5***

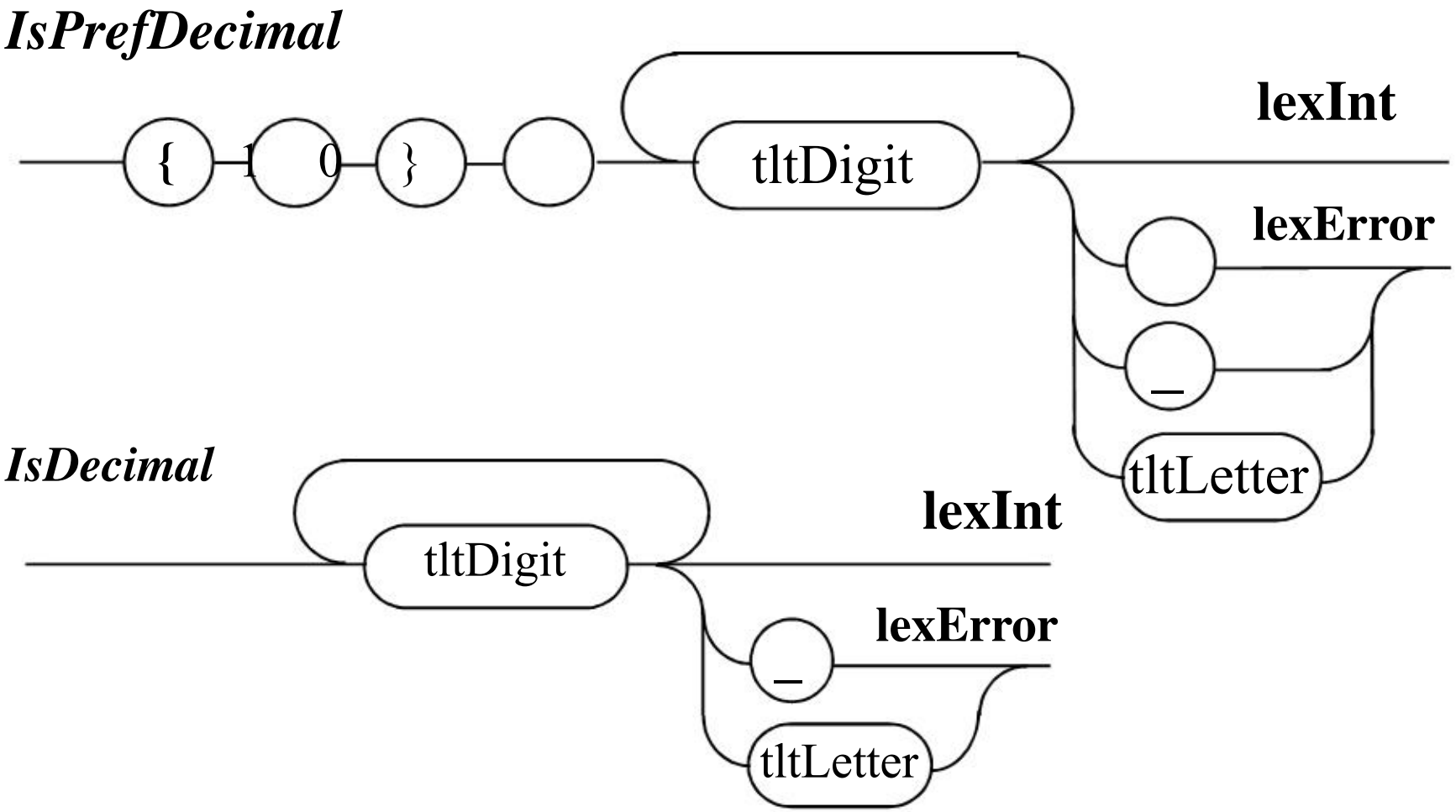



*Примечание 3* . Пять вариантов правил для распознавания действительного числа приводятся только для демонстрации арбитража при непрямом лексическом анализе. На практике легко можно обойтись одним правилом. Выдача ошибки происходит, если действительное число не отделяется разделителем от идентификатора или другого действительного числа, начинающегося с десятичной точки

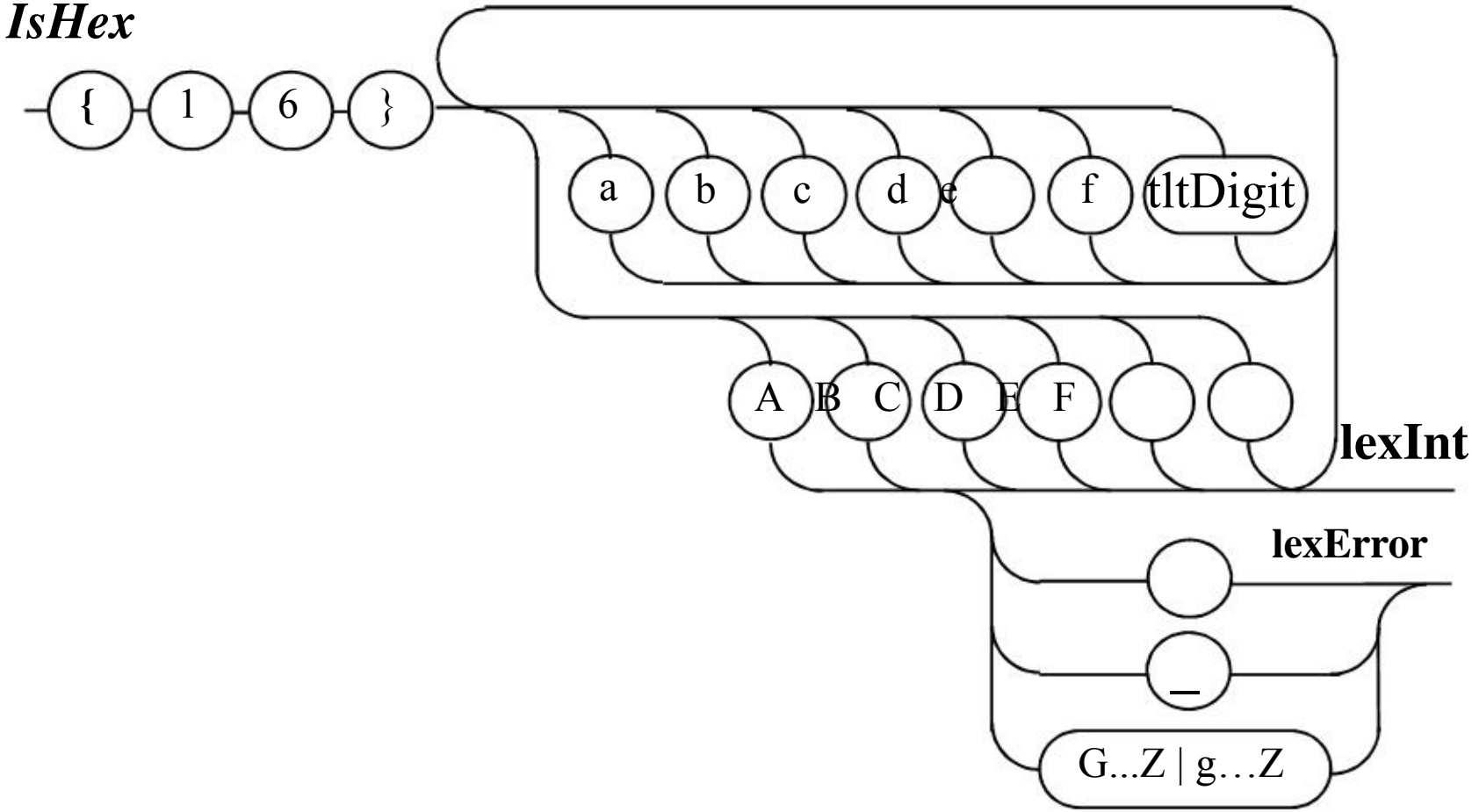

IsBinary
{
2
}
0
1
lexInt
lexError
_
tltLetter
2
3
4
5
6
7
8
9



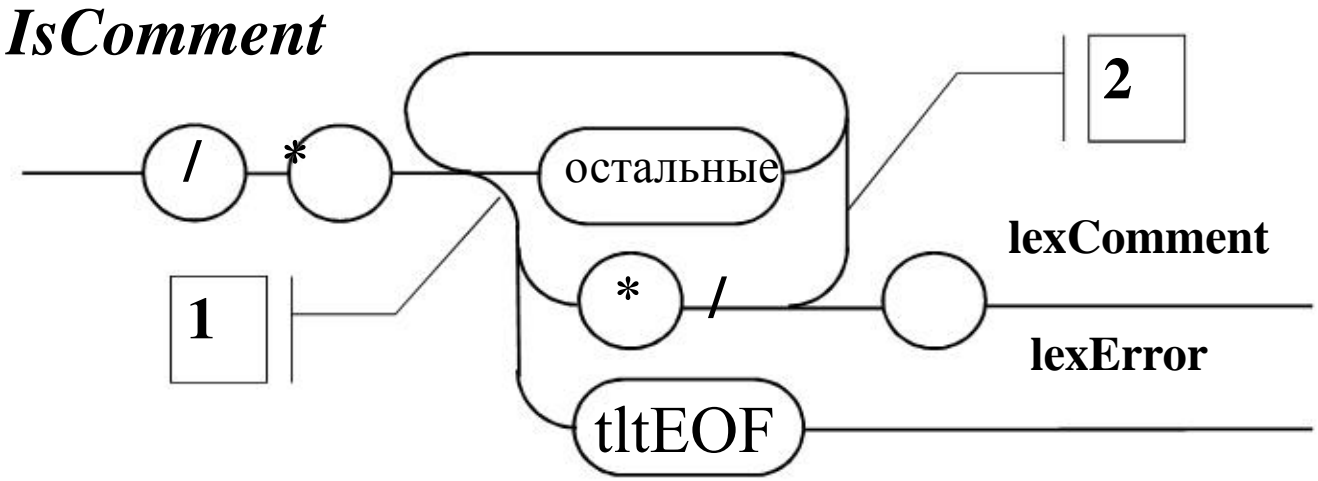

*Примечание 4* . Для двоичных и десятичных целых чисел необходима проверка того, что оно не сливается с теми цифрами, которые в них не содержатся.



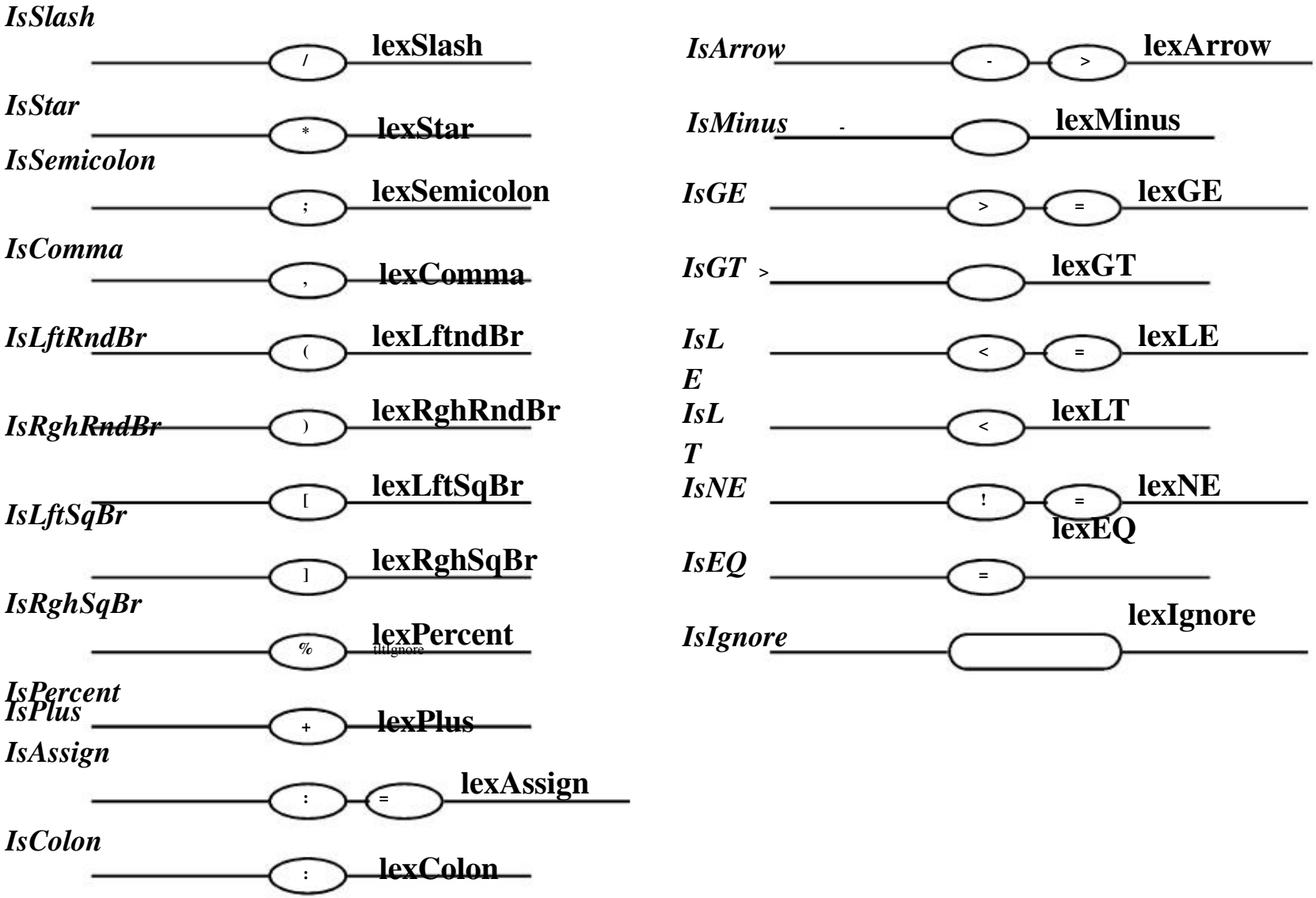

*Примечание 5*. Для целого десятичного числа без префикса анализ на недопустимость точки излишен, так как похожая ситуация должна была быть проанализирована раньше для действительного числа.



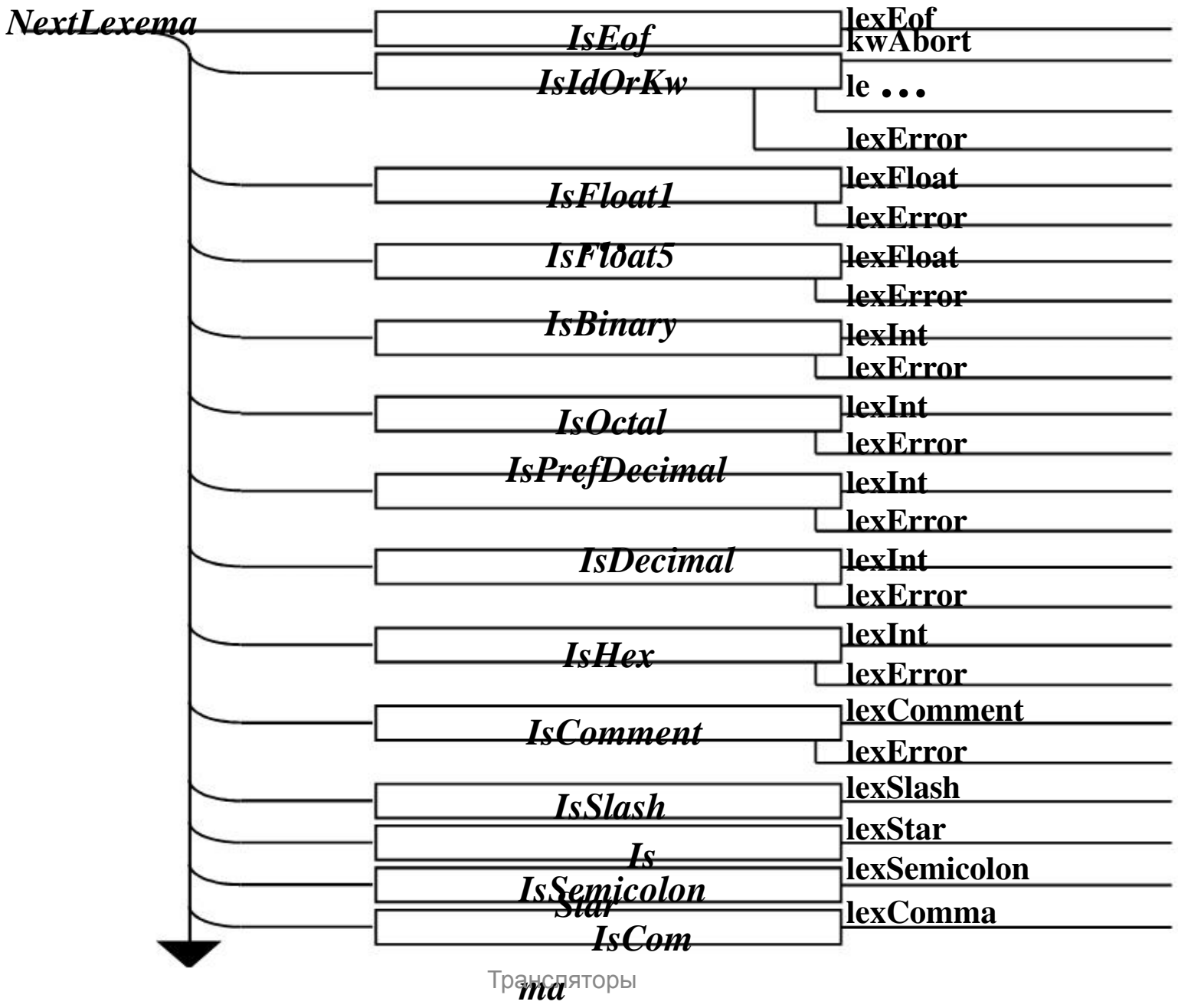

*Примечание 6*. Для целого шестнадцатеричного проверка на недопустимость должна исключать прописные и строчные буквы, используемые в самом числе. На представленных диаграммах это показано сокращенной записью путем задания диапазона. Это сделано для того, чтобы не загромождать диаграмму деталями.

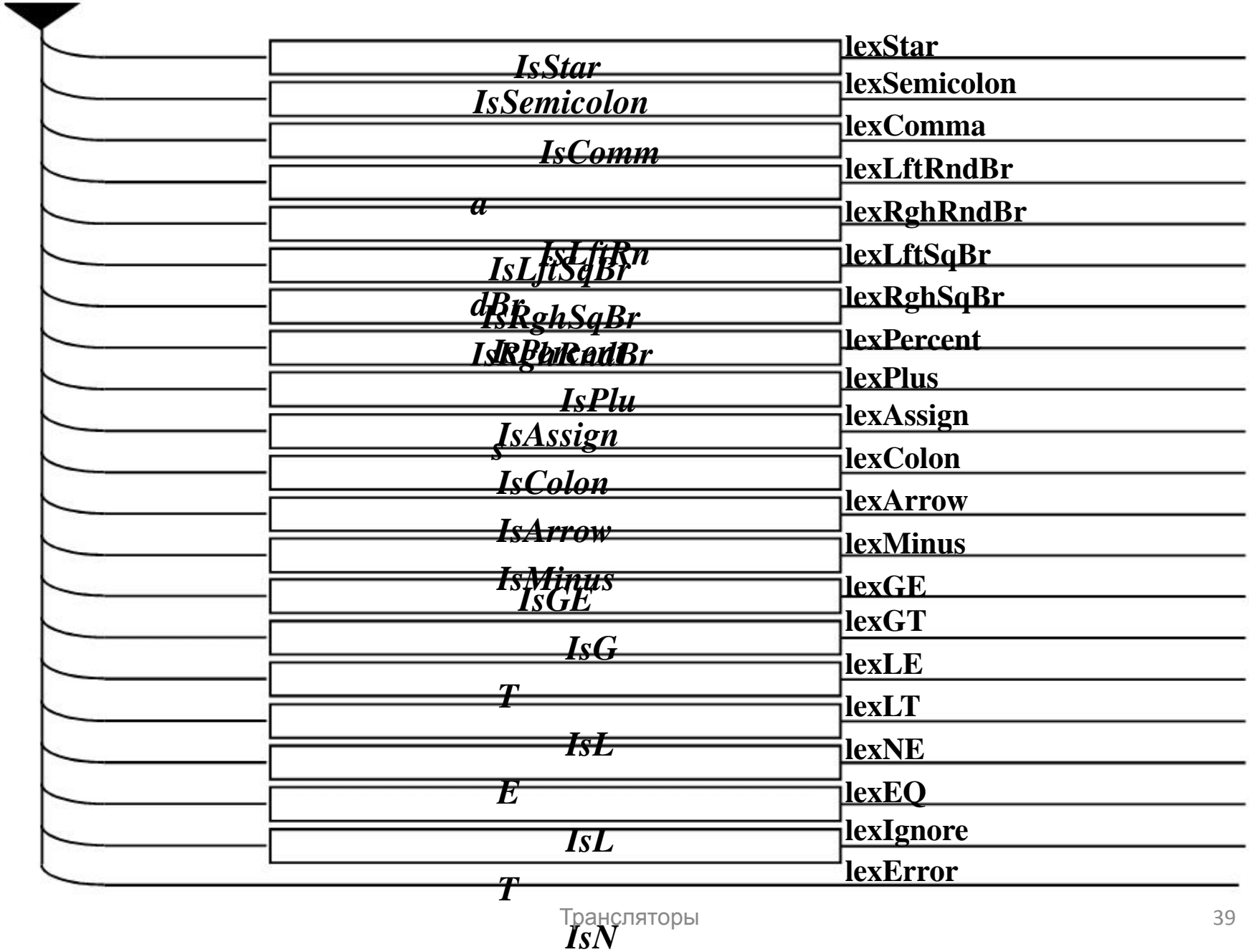



*Примечание 7.* Под остальными понимаются символы не рассматриваемые непосредственно в текущей точке. В точке 1 - это не “*” и не конец файла; в точке 2 - это не “*”, не конец файла и не “/”

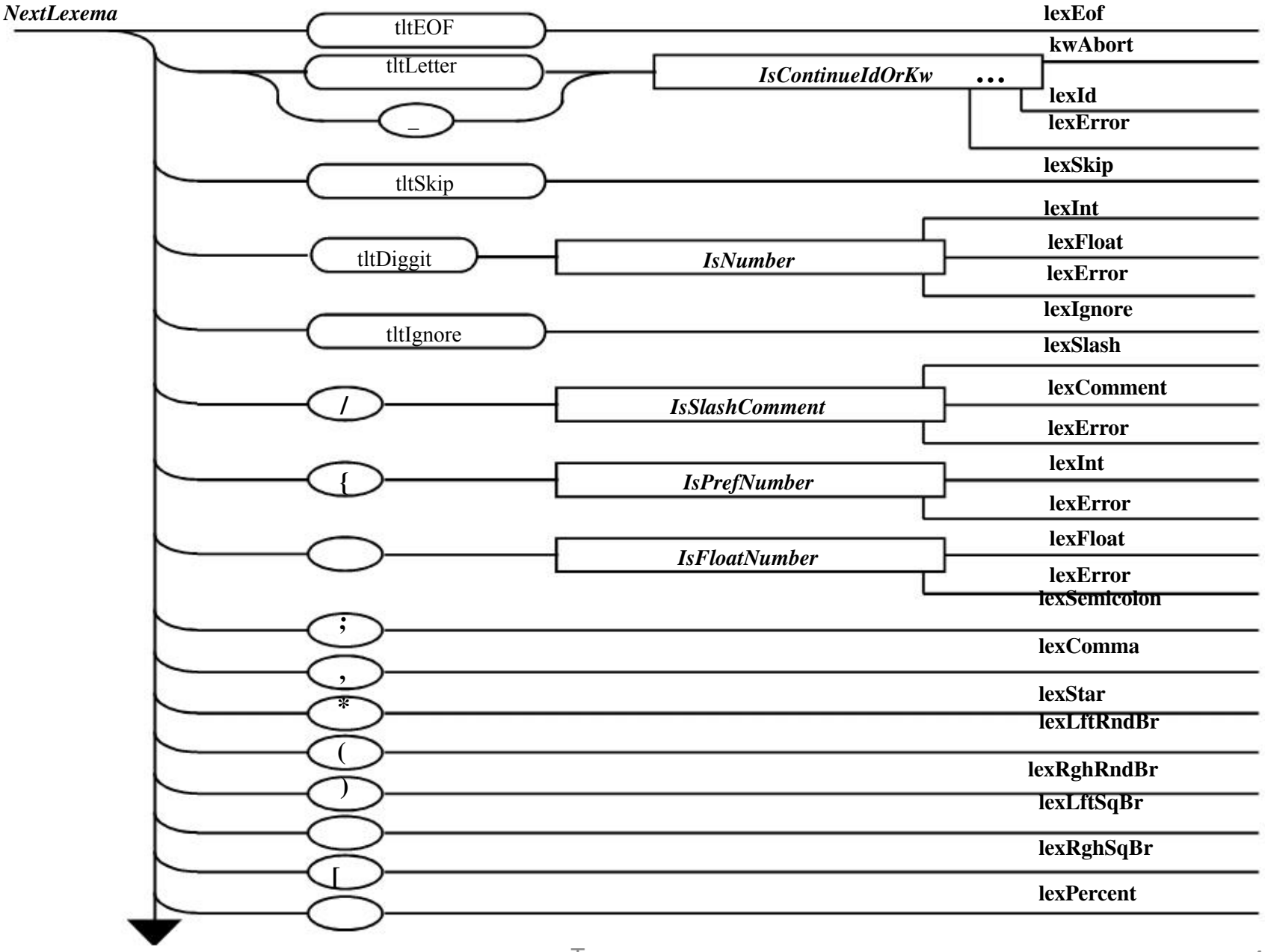


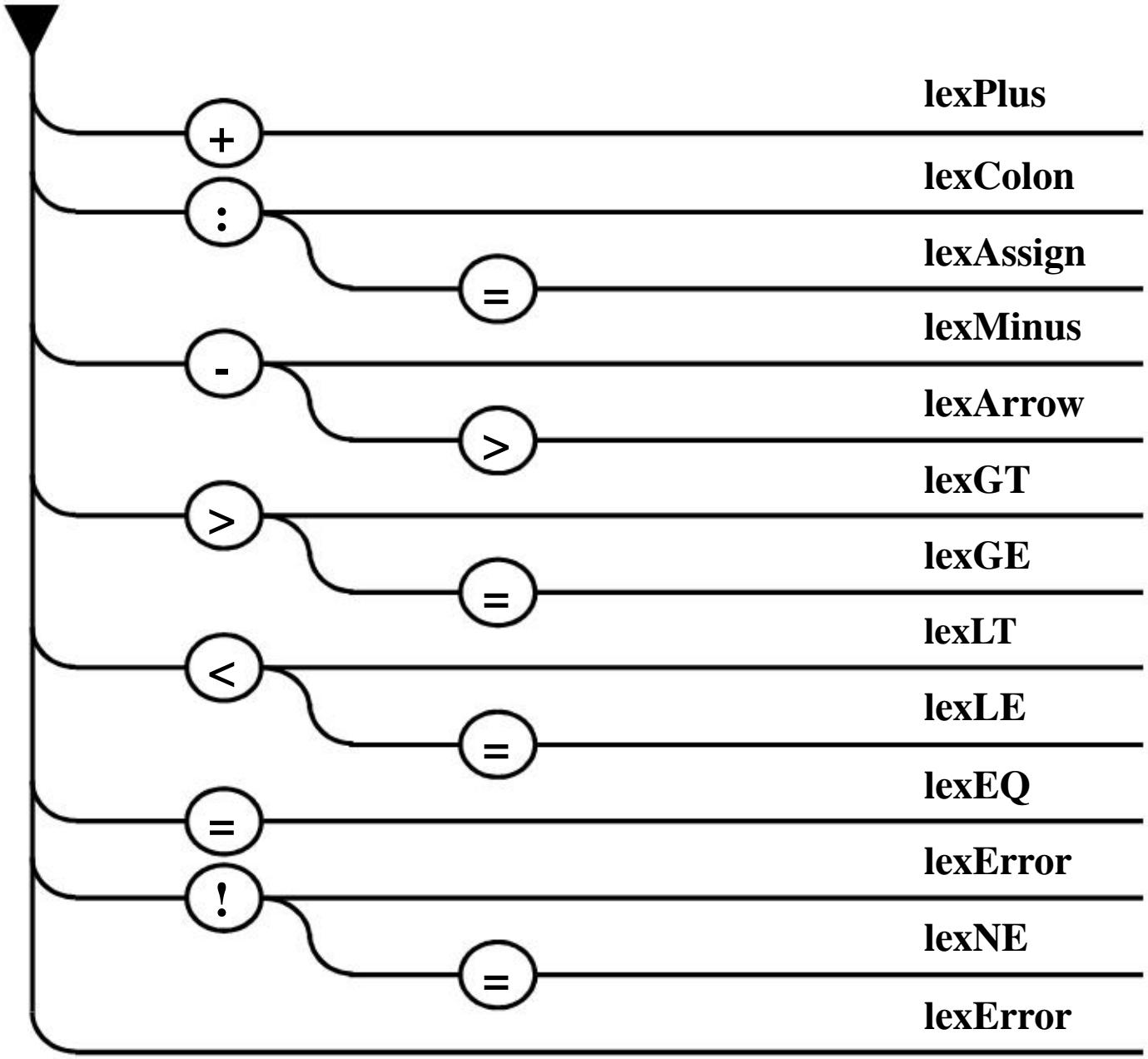

*Примечание 8.* Лексемы, определяющие разделительные символы, расположены в соответствии с их приоритетом при анализе сверху вниз и слева направо.

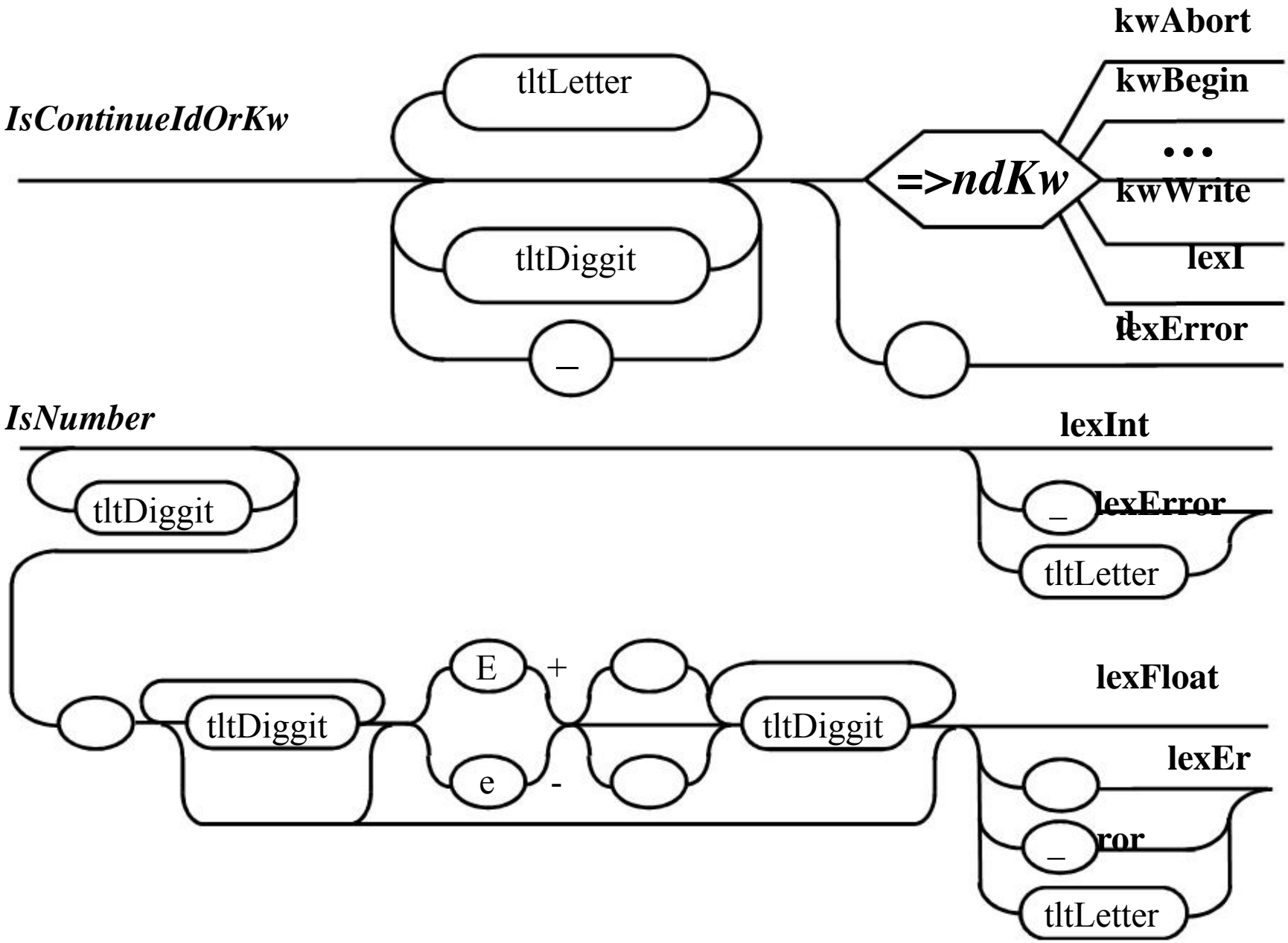

NextLexema
IsEof
lexEof
IsIdOrKw
kwAbort
le ...
lexError
IsFloat1
lexFloat
lexError
IsFloat5
lexFloat
lexError
IsBinary
lexInt
lexError
IsOctal
lexInt
lexError
IsPrefDecimal
lexInt
lexError
IsDecimal
lexInt
lexError
IsHex
lexInt
lexError
IsComment
lexComment
lexError
IsSlash
lexSlash
IsStar
lexStar
IsSemicolon
lexSemicolon
IsComma
lexComma


IsStar
IsSemicolon
IsComma
IsLftRndBr
IsRghRndBr
IsLftSqBr
IsRghSqBr
IsPercent
IsPlus
IsAssign
IsColon
IsArrow
IsMinus
IsGE
IsGT
IsLE
IsLT
IsNE
IsEQ
IsIgnore
IsError
lexStar
lexSemicolon
lexComma
lexLftRndBr
lexRghRndBr
lexLftSqBr
lexRghSqBr
lexPercent
lexPlus
lexAssign
lexColon
lexArrow
lexMinus
lexGE
lexGT
lexLE
lexLT
lexNE
lexEQ
lexIgnore
lexError

# Разработка прямого анализатора лексического

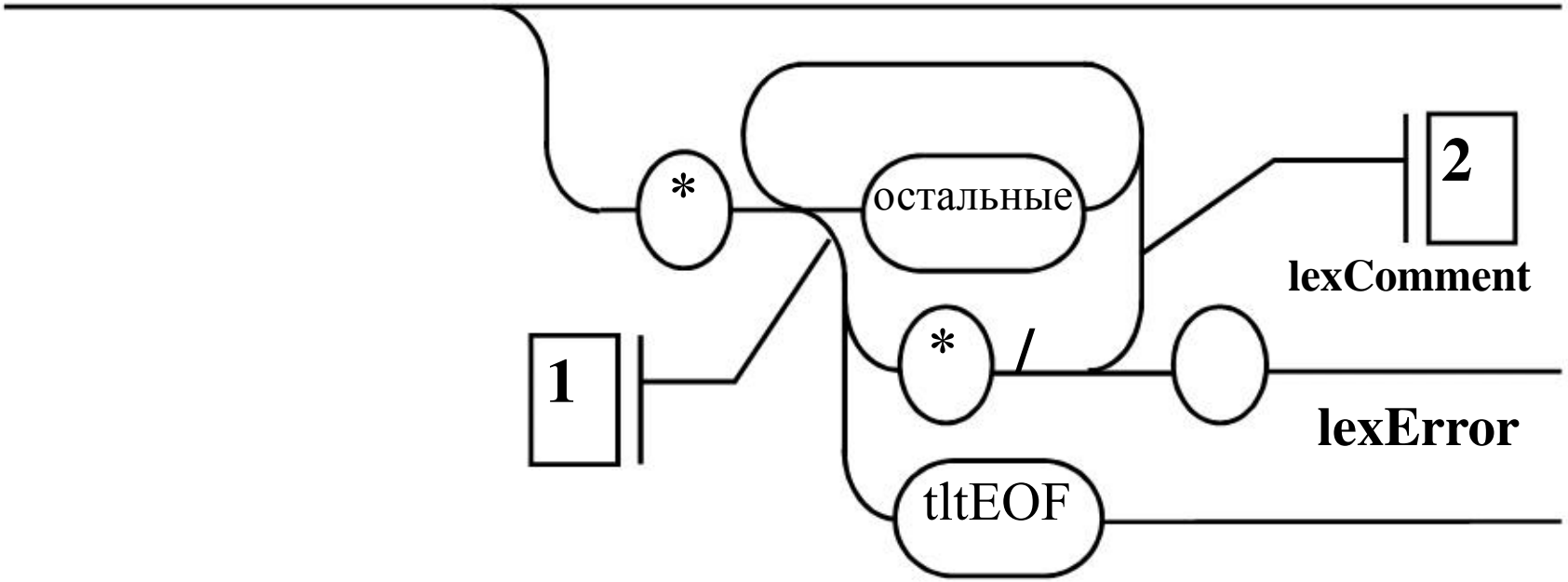

NextLexema
tltEOF
tltLetter
_
tltSkip
tltDiggit
tltIgnore
/
{
;
,
*
(
)
[
]
~
IsContinueIdOrKw
IsNumber
IsSlashComment
IsPrefNumber
IsFloatNumber
lexEof
kwAbort
lexId
lexError
lexSkip
lexInt
lexFloat
lexError
lexIgnore
lexSlash
lexComment
lexError
lexInt
lexError
lexFloat
lexError
lexSemicolon
lexComma
lexStar
lexLftRndBr
lexRghRndBr
lexLftSqBr
lexRghSqBr
lexPercent

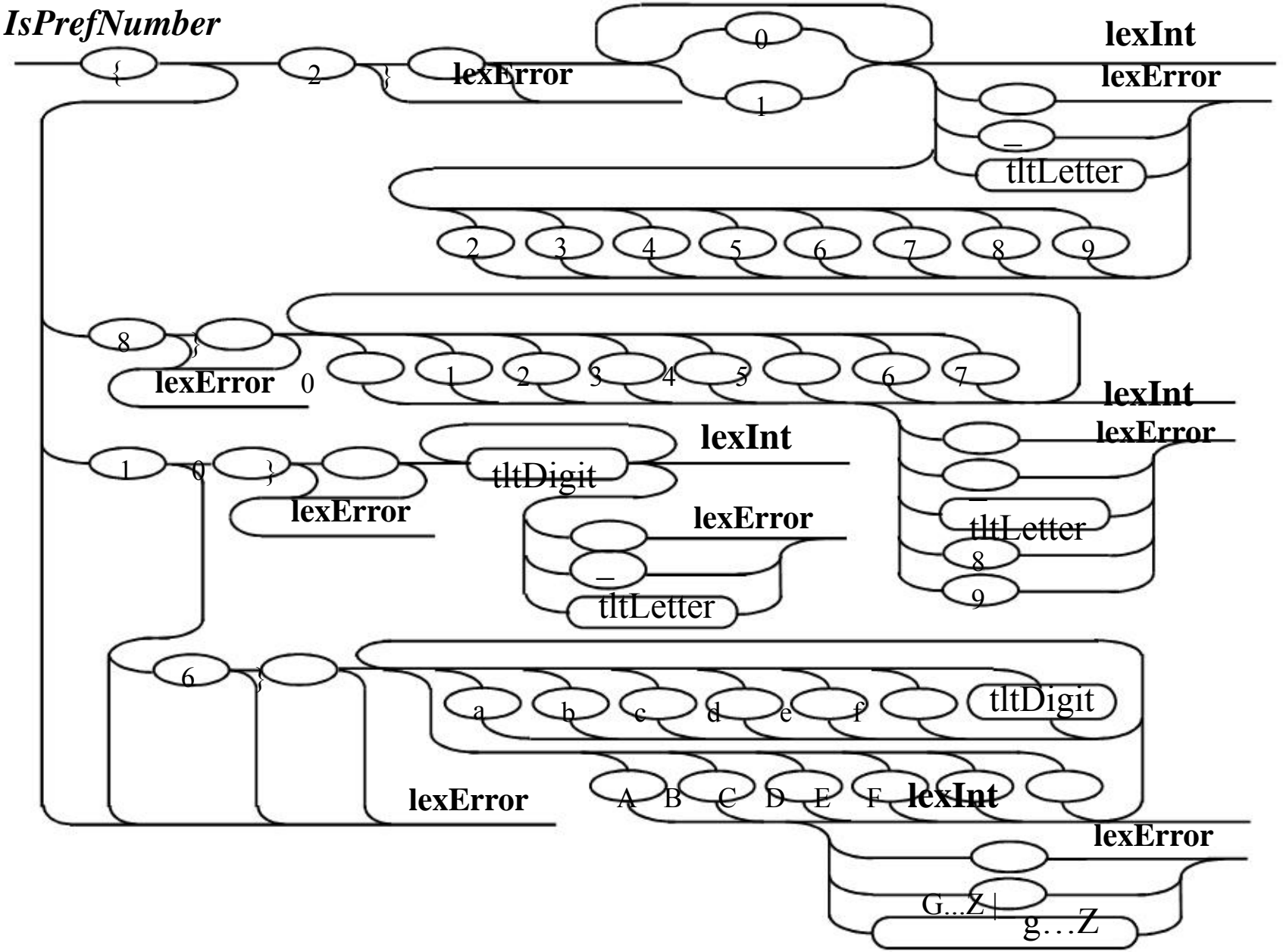

+
lexPlus
:
lexColon
=
lexAssign
-
lexMinus
>
lexArrow
>
lexGT
=
lexGE
<
lexLT
=
lexLE
=
lexEQ
!
lexError
=
lexNE
lexError

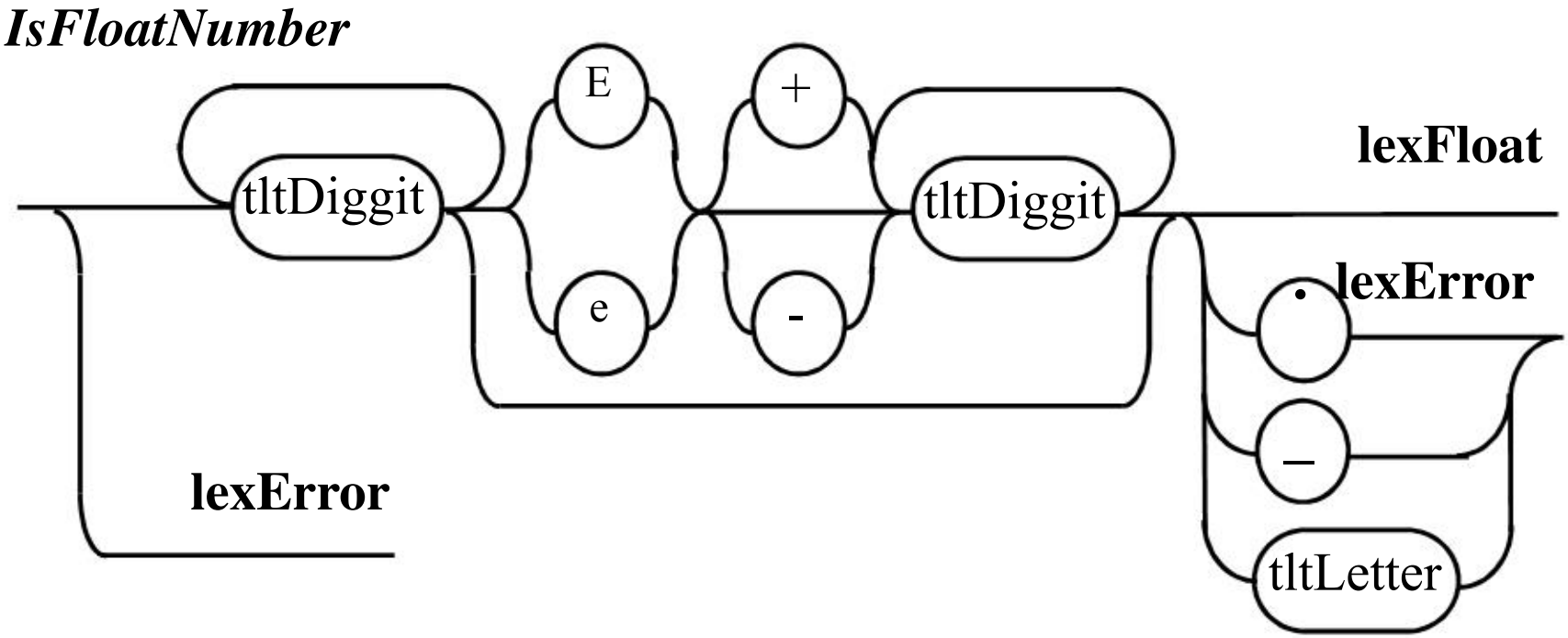

IsContinueIdOrKw
tltLetter
tltDiggit
_
=>ndKw
kwAbort
kwBegin
• • •
kwWrite
lexI
lexError
IsNumber
tltDiggit
lexInt
_
lexError
tltLetter
tltDiggit
E
+
e
-
tltDiggit
lexFloat
lexEr
_
ror
tltLetter

***IsSlashComment***

**lexSlash**




IsPrefNumber
{
2
}
lexError
0
1
lexInt
lexError
_
tltLetter
2
3
4
5
6
7
8
9
8
}
lexError
0
1
2
3
4
5
6
7
lexInt
lexError
1
0
}
lexError
tltDigit
lexInt
lexError
_
tltLetter
lexError
_
tltLetter
8
9
6
}
a
b
c
d
e
f
tltDigit
lexError
A
B
C
D
E
F
lexInt
lexError
G...Z
g…Z


IsFloatNumber
tltDiggit
E
e
+
-
tltDiggit
lexFloat
lexError
.
_
tltLetter
lexError