به نام خدا

محمدمهدی آقاجانی

۹۳۳۱۰۵۶

تمرین اول

استاد : دکتر سلیمان فلاح

صفحه ۸

4. Remove the syntactic category <prepositional phrase> and all related productions from the grammar in Figure 1.1. Show that the resulting grammar defines a finite language by counting all the sentences in it.

گرامر آن به صورت زیر خواهد بود :

<sentence> ::= <noun phrase> <verb phrase> .

<noun phrase> ::= <determiner> <noun>

<verb phrase> ::= <verb> | <verb> <noun phrase>

<noun> ::= boy | girl | cat | telescope | song | father

<determiner> :: = a | the

<verb> ::= saw | touched | surprised | sang

در کل میتوانیم ۶۲۴ جمله بسازیم :

- تمام جملات ایجاد شده از **<noun phrase>** finite هستند زیرا <noun> و<determiner> میتواندد توسط ۴*۱۲ ترمینال ایجاد شوند.
- Verb phrase به رسیدن به ترمینال باید از verb و یا از noun phrase استفاده کند که ۵۲ حالت مختلف میتواند باشد.
- با استفاده از اصل ضرب **۶۲۴** (۵۲*۱۲) بدست می‌آید

5. Using the grammar in Figure 1.6, derive the <sentence> aaabbbccc .

<sentence> → a<thing>bc

→ ab<thing>c

→ ab<other>bcc

→ a<other>bbcc

→ aa<thing>bbcc

→ aab<thing>bcc

→ aabb<thing>cc

→ aabb<other>bccc

→ aab<other>bbccc

→ aa<other>bbbccc

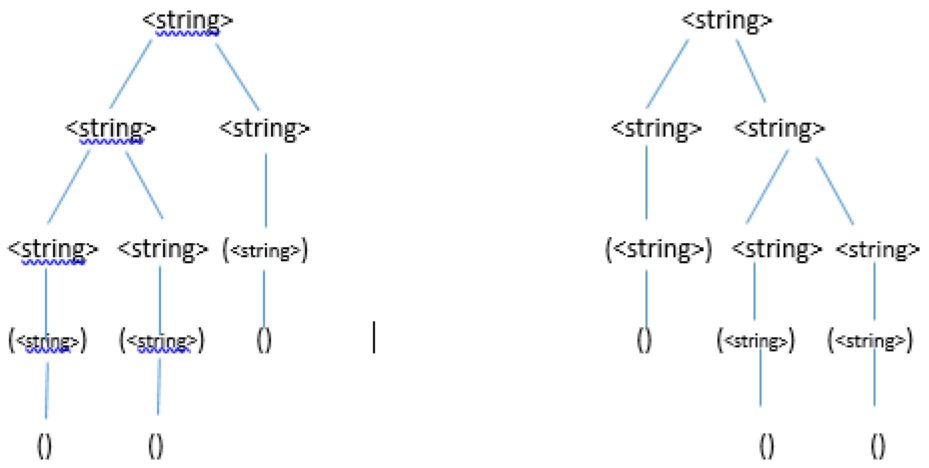
→ aaabbbccc

6. Consider the following two grammars, each of which generates strings of correctly balanced parentheses and brackets. Determine if either or both is ambiguous. The Greek letter e represents an empty string.

a) <string> ::= <string> <string> | (<string>) |[<string>] | ε

b) <string> ::= (<string>) <string> | [<string>] <string> | ε

در این سوال b مبهم نیست اما a مبهم میباشد و دو درخت وجود خواهد داشت :



7.Describe the languages over the terminal set { a, b } defined by each of the following grammars:
a) <string> ::= a <string> b | ab
b) <string> ::= a <string> a | b <string> b | e
c) <string>::= a <B> | b <A>
<A> ::= a | a <string> | b <A> <A>
<B> ::= b | b <string> | a <B> <B>

a. رشته هایی که با تعدادی a شروع شده اند و به همان تعداد b پایان یافته اند

b. رشته هایی که از a ,b هستند و متقارن میباشند (خودشان و قرینه آن ها با هم برابر است) که تهی میتواند باشد

c. رشته هایی با تعداد مساوی از a , b

9. Identify which productions in the English grammar of Figure 1.1 can be reformulated as type 3 productions. It can be proved that productions of the form <A> :: = a1 a2 a3 …an <B> are also allowable in regular grammars. Given this fact, prove the English grammar is regular—that is, it can be defined by a type 3 grammar. Reduce the size of the language by limiting the terminal vocabulary to boy, a, saw, and by and omit the period. This exercise requires showing that the concatenation of two regular grammars is regular.

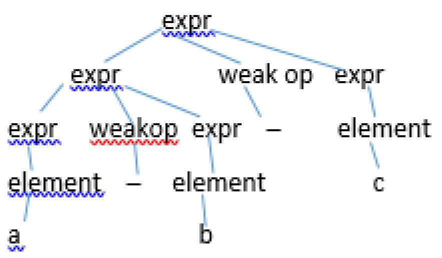
در یک گرامر ساختاری منظم وجود خواهد داشت. همچنین concat رشته های منظم نیز منظم خواهد بود و اگر گرامر را چند قسمت نماییم در نتیجه پیوند این جملات هم منظم میشود.

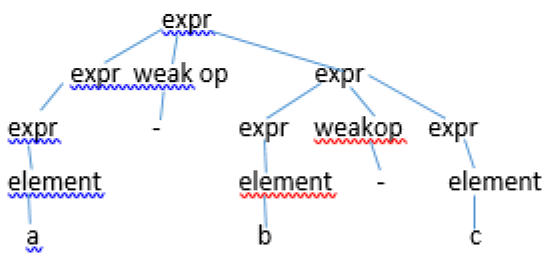
صفحه ۱۶

2.Consider the following specification of expressions:

<expr> ::= <element> | <expr> <weak op> <expr>
<element> ::= <numeral> | <variable>
<weak op> ::= **+** | **–**

Demonstrate its ambiguity by displaying two derivation trees for the expression “**a–b–c** ”. Explain how the Wren specification avoids this problem.





در زبان wren اشتقاق از سمت راست انجام میشود و اینگونه مشکل رفع میگردد.

3. This Wren program has a number of errors. Classify them as context free, context-sensitive, or semantic.

```
program errors was
        var a,b : integer ;
        var p,b ; boolean ;
    begin
        a := 34;
        if b¹0 then p := true else p := (a+1);
        write p; write q
end
```

- Semantics
  - b و p initialize نشده اند
- Context sensitive
  - P نوع integer ندارد ولی writeشده است.
  - p:=a+1 ، نوع دو طرف تساوی همسان نیست یکی integer و دیگریBoolean
  - تعریف دوبار b : یک بار interger و بار دیگر Boolean
  - بدون آنکه q تعریف بشود write شده
- Context free
  - به کار بردن نامساوی در if b≠0 then، غلط است
  - شرطif خاتمه پیدا نکرده است.
  - استفاده از ; به جای : (در خط دوم)
  - استفاده از is به جای was

5.This BNF grammar defines expressions with three operations, *, -, and +, and the variables “a”, “b”, “c”, and “d”.

```
<expr> ::= <thing> | <thing> * <expr>
<object> ::= <element> | <element> – <object>
<thing> ::= <object> | <thing> + <object>
<element> ::= a | b | c | d | (<object>)
```

a) Give the order of precedence among the three operations.
b) Give the order (left-to-right or right-to-left) of execution for each operation.
c) Explain how the parentheses defined for the nonterminal <element> may be used in these expressions. Describe their limitations.

a. اولویت – از * و + از * بیشتر است

b. – از راست به چپ ، + از چپ به راست ، * از راست به چپ

7. Explain the relation between left or right recursion in definition of expressions
and terms, and the associativity of the binary operations (left-to- right or right-to-left).

در identifier می توان terminal ها را به جای <letter> , <digit> nonterminal قرار داد:

<identifier>::=a|b|…|z|>identifier>a|>identifier>b|…|>identifier>z|>identifier>0|…|>identifier>9

<numeral>::=0|1|…|9|0>numeral>|…|9>numeral>

8. Write a BNF specification of the syntax of the Roman numerals less than 100. Use this grammar to derive the string “XLVII”.
<number> ::= X< strCL >< strIV > | L< strX >|< strX >

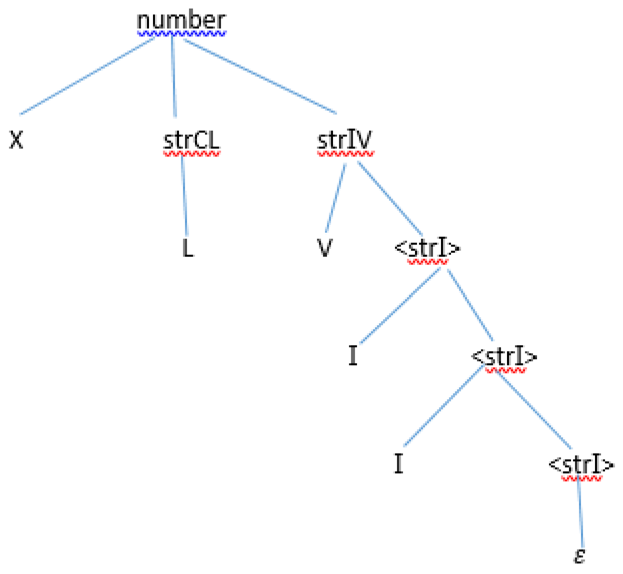
<strCL> ::= C|L

<strX> ::= X< strX > | < strIV >

<strIV> ::= I < strXV >|V < strI > | < strI >

<strXV> ::= X|V

<strI> ::= I < strI >



9. Consider a language of expressions over lists of integers. List constants have the form: [3,-6,1], [86], [ ]. General list expressions may be formed using the binary infix operators
+, –, *, and @ (for concatenation), where * has the highest precedence, + and - have the same next lower precedence, and @ has the lowest precedence. @ is to be right associative and the other operations are to be left associative. Parentheses may
be used to override these rules. Example: [1,2,3] + [2,2,3] * [5,-1,0] @ [8,21] evaluates to [11,0,3,8,21]. Write a BNF specification for this language of list expressions. Assume that <integer> has already been defined. The conformity of lists for the arithmetic operations is not handled by the BNF grammar since it is a context-sensitive issue.

<expr> := < highOp > | <expr> @ <list>

<lowOp> := <highOp> | < lowOp > + <list> | < lowOp > - <list>

<highOp> := <list> | < highOp> * <list>

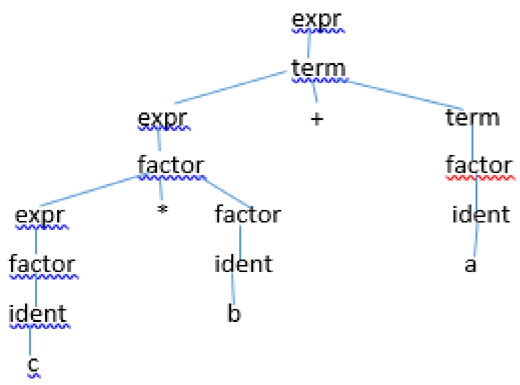
<list> := [ ] | [<content>] | (<expr>)

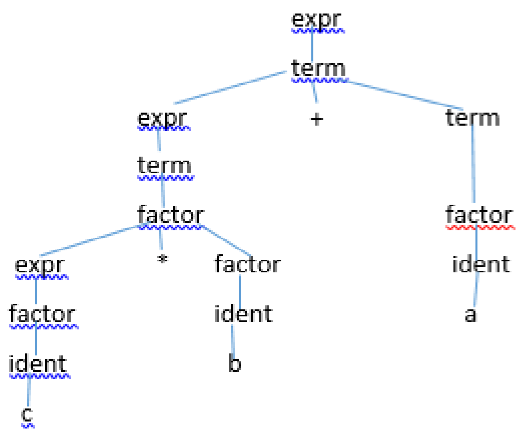
<content> := <int> | <content>, <int>

<int> := 1|2|3|4|5|6|7|8|9|0

10. Show that the following grammar for expressions is ambiguous and provide an alternative unambiguous grammar that defines the same set of expressions.

<expr> ::= <term> | <factor>
<term> ::= <factor> | <expr> + <term>
<factor> ::= <ident> | (<expr>) | <expr> * <factor>
<ident> ::= a | b | c





<expr> ::= <term> | <expr> + <term>

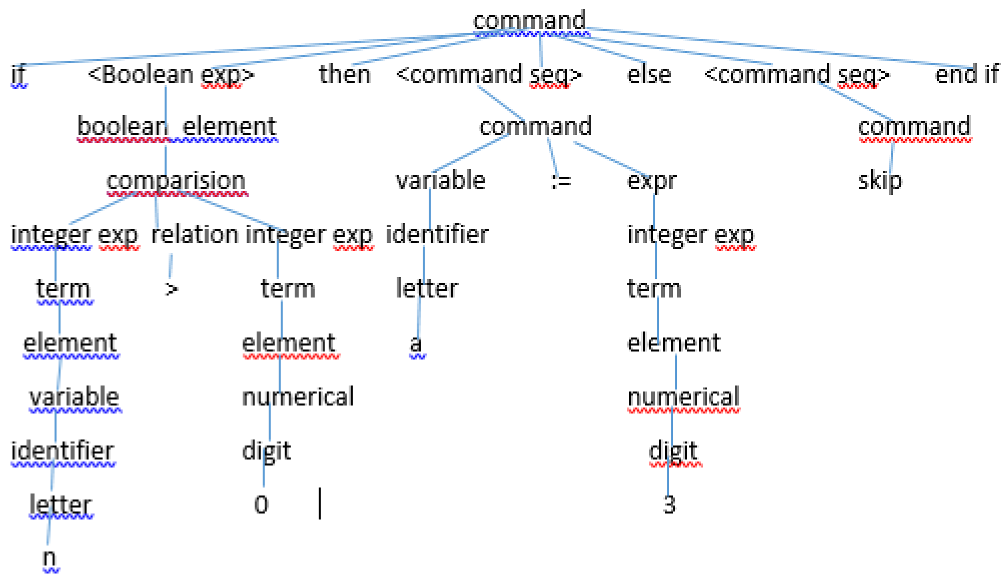
<term> ::= <factor> | <term> * <factor>

<factor> ::= <ident> | (<expr>)

<ident> ::= a| b | c

صفحه ۲۹

1. Construct a derivation tree and an abstract syntax tree for the Wren command
“if n>0 then a := 3 else skip end if ”. Also write the abstract tree as a Prolog structure.



2. Parse the following token list to produce an abstract syntax tree:
[while, not, lparen, ide(done), rparen, do, ide(n), assign, ide(n), minus, num(1), semicolon, ide(done), assign, ide(n), greater, num(0), end, while]

صفحه ۷۱ :

1.In old versions of Fortran that did not have the character data type, character strings were expressed in the following format: <string literal> ::= <numeral> H <string>
where the <numeral> is a base-ten integer (³ 1), H is a keyword (named after Herman Hollerith), and <string> is a sequence of characters. The semantics of this string literal is correct if the numeric value of the baseten numeral matches the length of the string. Write an attribute grammar using only synthesized attributes for the nonterminals in the definition of <string literal>.

value : یک attribute که با <numeral> ارتباط دارد و مقدار numeral را معین مینماید

Length : یک attribute که با <string> ارتباط دارد و طول یک string را مشخص میکند

<string literal> ::= <numeral> H <string>

<numeral>.value ← calculate_value(<numeral>)

<string> :length ← calculate_length(<string>)

Predicate: <numeral>.value == <string>.length

2. Repeat exercise 1, using a synthesized attribute for <numeral> and an inherited attribute for <string>.

value : یک attribute که با <numeral> ارتباط دارد و مقدار numeral را معین مینماید

Actual length : یک attribute که با <string> ارتباط دارد و طول یک string را مشخص میکند

expected length : یک inherited attribute که با <string> ارتباط دارد و طول یک string را مشخص میکند

<string literal> ::= <numeral> H <string>

<numeral>.value ← calculate_value(<numeral>)

<string> :length ← calculate_length(<string>)

Predicate: <numeral>.value == <string>.length

5. Expand the binary numeral attribute grammar (either version) to allow for binary numerals with no binary point (1101), binary fractions with no fraction part (101.), and binary fractions with no whole number part (.101).

<binary numeral> ::= <binary digit>

```
    val (<binary numeral>) ←val (<binary digit>)
```

|<binary digit>.

```
    val (<binary numeral>) ← val (<binary digit>)
```

|.<binary digit>

val (<binary numeral>) ←val (<binary digit>) / $2^{\text{len (<binary digit>)}}$

<binary digit> ::= <binary digit>$_1$<bit>

```
    val (<binary digit>) ←2 * val (<binary digit>)₁ + val(<bit>)

    len (<binary digit>) ←len (<binary digit>)₁ + 1
```

| bit

```
    val (<binary digit>) ← val (<bit>)

    len (<binary digit>) ←1
```

11. A binary tree consists of a root containing a value that is an integer, a (possibly empty) left subtree, and a (possibly empty) right subtree. Such a binary tree can be represented by a triple (Left subtree, Root, Right subtree). Let the symbol nil denote an empty tree. Examples of binary trees include:
(nil,13,nil)
    represents a tree with one node labeled with the value 13.
((nil,3,nil),8,nil)
    represents a tree with 8 at the root, an empty right subtree, and a nonempty left subtree with root labeled by 3 and empty subtrees.
The following BNF specification describes this representation of binary trees.

```
<binary tree> ::= nil | ( <binary tree> <value> <binary tree> )
<value> ::= <digit> | <value> <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Augment this grammar with attributes that carry out the following tasks:
a) A binary tree is balanced if the heights of the subtrees at each interior node are within one of each other. Accept only balanced binary trees.
b) A binary search tree is a binary tree with the property that all the values in the left subtree of any node N are less than the value at N, and all the value in the right subtree of N are greater than or equal to the value at node N. Accept only binary search trees.

<binary tree> ::=

nil

```
height(<binary tree>) ← -1 ;

max_number(<binary tree>) ← -1;

min_number(<binary tree>) ← -1;
```

| (<binary tree>$_2$ <value> <binary tree>$_3$)

```
height(<binary tree>) ←

    max{height(<binary tree>2),height(<binary
    tree>3)}+1;
```

```
        if (min_number(<binary tree>2) == -1) then
              min_number(<binary tree>)←number(<value>)
        else
              min_number(<binary tree>)←
                    min_number(<binary tree>2)
        end if ;
        if (max_number(<binary tree>3) == -1) then
                    max_number(<binary tree>)←number(<value>)
        else
              max_number(<binary tree>)←
                    max_number(<binary tree>3) end if ;
        condition :
              [ number(<value>) > max_number(<binary tree>2) ]
              AND [(min_number(<binary tree>3) == -1) ) OR (
              number(<value>) =< min_number(<binary tree>3) ) ]
              AND [ abs(height(<binary tree>3)-height(<binary
              tree>2)) =< 1 ];
```

<value> ::= <digit>

```
        number(<value>) ← number(<digit>) ;
```

| <value>2 <digit>

```
        number(<value>)←number(<value>2) *10 + number(<digit>) ;
```

<digit> ::= 0

```
        number(<digit>)←0;| 1 number(<digit>)←1;
| 2
        number(<digit>)←2;
| 3
        number(<digit>)←3;
| 4
        number(<digit>)←4;
| 5
        number(<digit>)←5;
| 6
        number(<digit>)←6;
| 7
        number(<digit>)←7;
| 8
        number(<digit>)←8;
| 9
        number(<digit>)←9;
```