"In the name of God"

HW2- programming language

Marjan Albouye-9131060

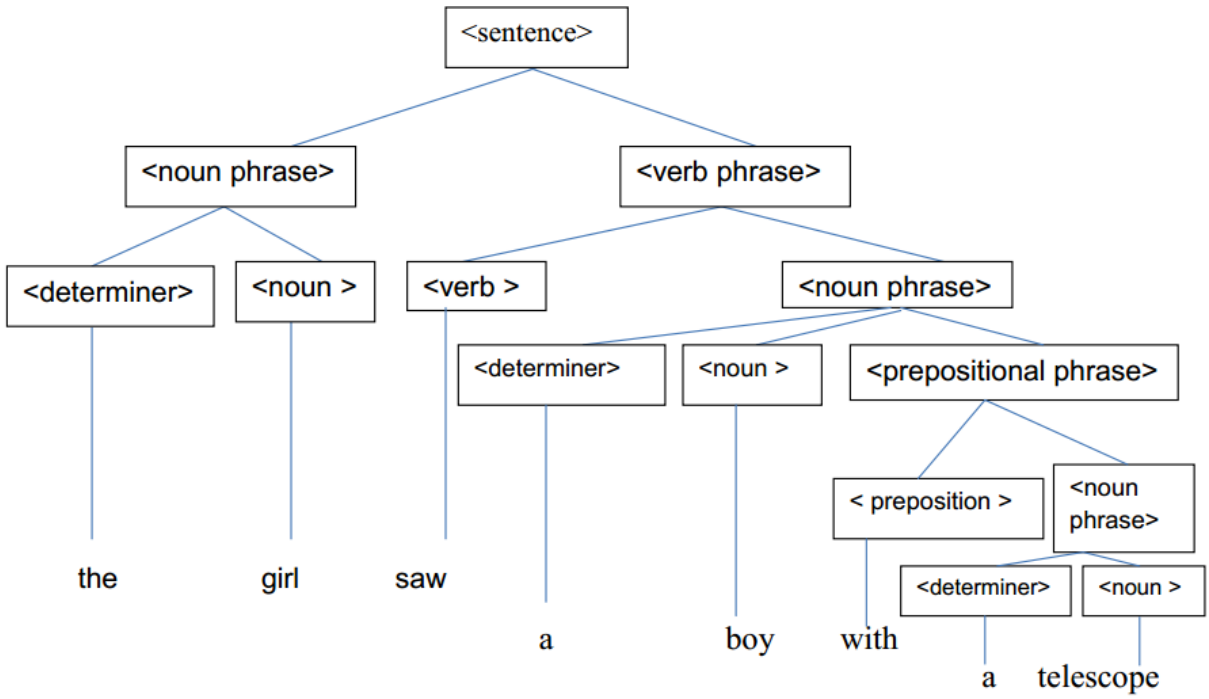
Page 8:

1. Find two derivation trees for the sentence " the girl saw a boy with a telescope." using the grammar in Figure 1.1 and show the derivations that correspond to the two trees.

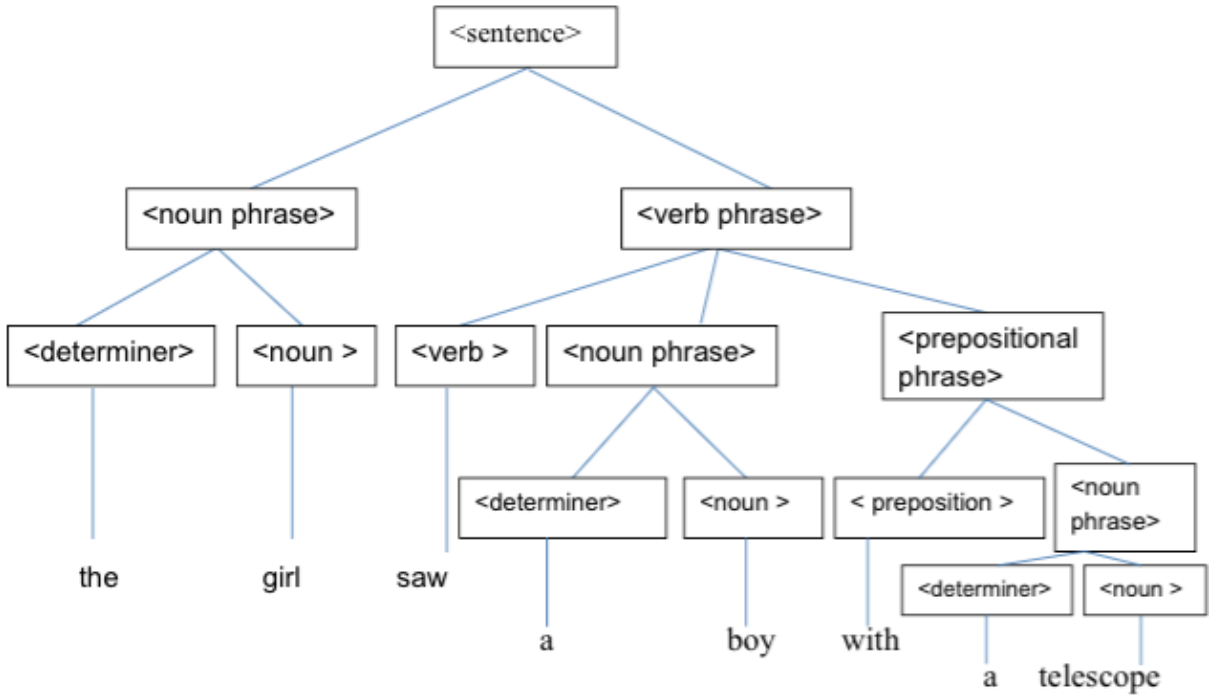
<sentence> ::= <noun phrase> <verb phrase> .
<noun phrase> ::= <determiner> <noun>| <determiner> <noun> <prepositional phrase>
<verb phrase> ::= <verb> | <verb> <noun phrase>| <verb> <noun phrase> <prepositional phrase>
<prepositional phrase> ::= <preposition> <noun phrase>
<noun> ::= boy | girl | cat | telescope | song | feather
<determiner> ::= a | the
<verb> ::= saw |touched | surprised | sang
<preposition> ::= by | with

Answer:

Derivation Tree 1:



Derivation Tree 2:



2. Give two different derivations of the sentence " the boy with a cat sang a song. ", but show that the derivations produce the same derivation tree.

Answer:

Derivation:

<sentence> ⇨ <noun phrase> <verb phrase>.
⇨ <determiner> <noun> <prepositional phrase> <verb phrase>.
⇨ The <noun> <preposition> <noun phrase> <verb phrase>.
⇨ The boy <preposition> <noun phrase> <verb phrase>.
⇨ The boy with <determiner> <noun> <verb phrase>.
⇨ The boy with a <noun> <verb phrase>.
⇨ The boy with a cat <verb> <noun phrase>.
⇨ The boy with a cat sang <noun phrase>.
⇨ The boy with a cat sang <determiner> <noun>.
⇨ The boy with a cat sang a <noun>.
⇨ The boy with a cat sang a song.

7. Describe the languages over the terminal set { a , b} defined by each of the following grammars:

a) <string> ::= a <string> b | ab

Answer: $\{a^n b^n | n \in \mathbb{N}\}$

b) <string> ::= a <string> a | b <string> b | ε

Answer: This defines a language with Polynomial sentences $\{w \in \{a,b\}^* | w^R = w\}$

w^R is reverse of R. this language defines sentences like ww^R.

c) <string>::= a <B> | b <A>

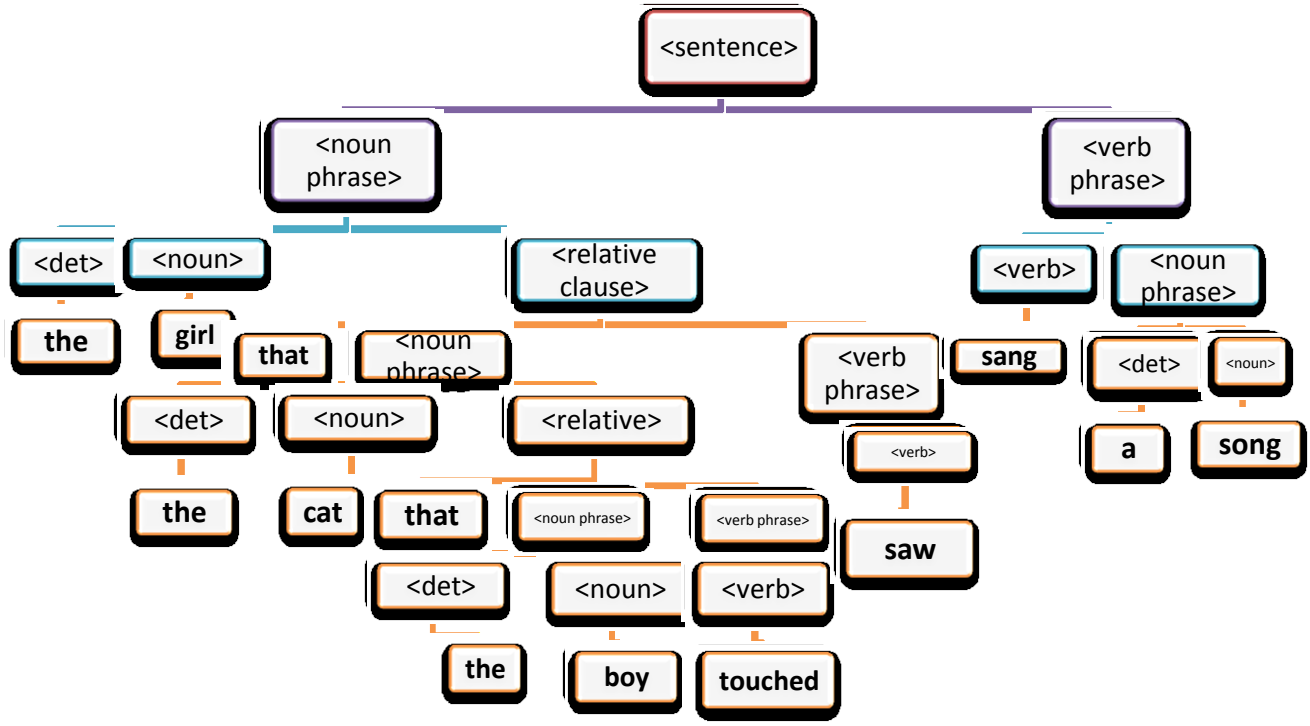
<A> ::= a | a <string> | b <A> <A>

<B> ::= b | b <string> | a <B> <B>

Answer: $\{w \in \{a,b\}^* | N_a(|w|) = N_b(|w|)\, , N_a(|w|) \geq 1\, , N_b(|w|) \geq 1\}$

8. Use the following grammar to construct a derivation tree for the sentence " the girl that the cat that the boy touched saw sang a song."

<sentence> ::= <noun phrase> <verb phrase> .
<noun phrase> ::= <determiner> <noun> | <determiner> <noun> <relative clause>
<verb phrase> ::= <verb> | <verb> <noun phrase>
<relative clause> ::= that <noun phrase> <verb phrase>
<noun> ::= boy | girl | cat | telescope | song | feather
<determiner> ::= a | the
<verb> ::= saw | touched | surprised | sang



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> ::= a1a2a3…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.

Answer:

<noun phrase> ::= **a boy** | **a boy** <prepositional phrase>

<noun phrase> ::= a boy | a boy <prepositional phrase>

<prepositional phrase> ::= **by** <noun phrase>

<term> ::= <prepositional phrase><verb phrase> (the concatenation of two regular grammars is regular)

<sentence>

→ <noun phrase><verb phrase>

→ **a boy** < verb phrase> | **a boy** <prepositional phrase><verb phrase>

→ **a boy** < verb phrase> | **a boy** <term>

Page 16:

2. Consider the following specification of expressions:

<expr> ::= <element> | <expr> <weak op> <expr>

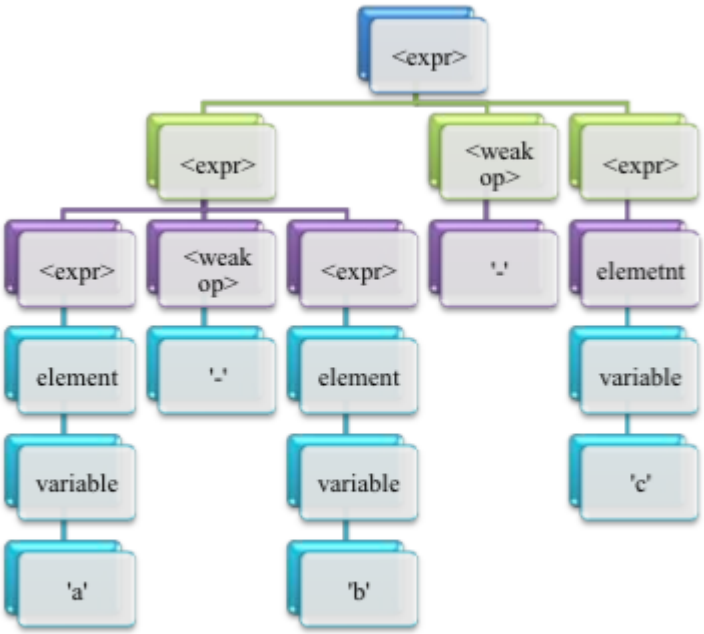
<element> ::= <numeral> | <variable>

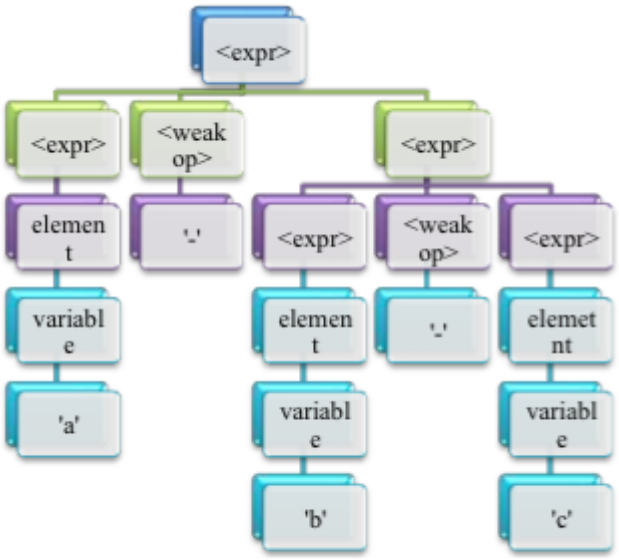
<weak op> ::= +| –

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

Answer:

Derivation Trees:


<expr>
<expr>
<weak op>
<expr>
<expr>
<weak op>
<expr>
'-'
elemetnt
element
'-'
element
variable
variable
variable
'c'
'a'
'b'


<expr>
<expr>
<weak op>
<expr>
elemen t
'-'
<expr>
<weak op>
<expr>
variabl e
elemen t
'-'
elemet nt
variabl e
variabl e
'a'
'b'
'c'

To avoid this problem in wren specification of expressions is like below:

<integer expr> ::= <term> | <integer expr> <weak op> <term>

<term> ::= <element> | <term> <strong op> <element>

<element> ::= <numeral> | <variable> | (<integer expr>)| –<element>

<weak op> ::= +| –

<strong op> ::= *| /

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 b≠0 then p := true else p := (a+1);
        write p; write q
   end
```

1- was must be is (context free)

2- var p,b : Boolean (context free)

3- b declared twice (integer and Boolean) (context sensitive)

4-b <> 0 (context free)

5- if should have endif (context free)

6- p is Boolean but (a+1) is integer so p:=(a+1) is an error (context sensitive)

7-write p is incorrect because write comes with integer but p is boolean type (context sensitive)

8- q is an undefined variable(context sensitive)

9- write q should have ;

10- b≠ 0 b is not initialized (semantic)

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

Answer: - > + > *

b) Give the order (left-to-right or right-to-left) of execution for each operation.

Answer:

$- \rightarrow right\ to\ left$

$+ \rightarrow left\ to\ right$

$* \rightarrow right\ to\ left$

c) Explain how the parentheses defined for the nonterminal <element> may be used in these expressions. Describe their limitations.

Answer: parentheses are used to set precedence only in expressions that have just –

For example : $(a - b) - c$

6. Explain how the Wren productions for <identifier> and <numeral> can be written in the forms allowed for regular grammars (type 3)—namely, <A> ::= a or <A> ::= a<B>.

Answer:

<identifier>::= a | b | c | d | e | f | g | h | i | j | k | l | m| n | o | p | q | r | s | t | u | v | w | x | y | z|<identifier>a|<identifier>b|…|<identifier> z | <identifier>0 | <identifier>1 | …| <identifier>9

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

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.

Answer:

<expr> ::= <thing> @ <expr> | <thing>

<thing> ::= <thing> + <term> | <thing> - <term> | <term>

<term> ::= <term> * <element> | <element>

<element> ::= [ ] | [<constant>] | (<expr>)

<constant> ::= <integer> | < integer> , <integer>

<integer> :: = in exercise is mentioned that <integer> has already been defined.

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>

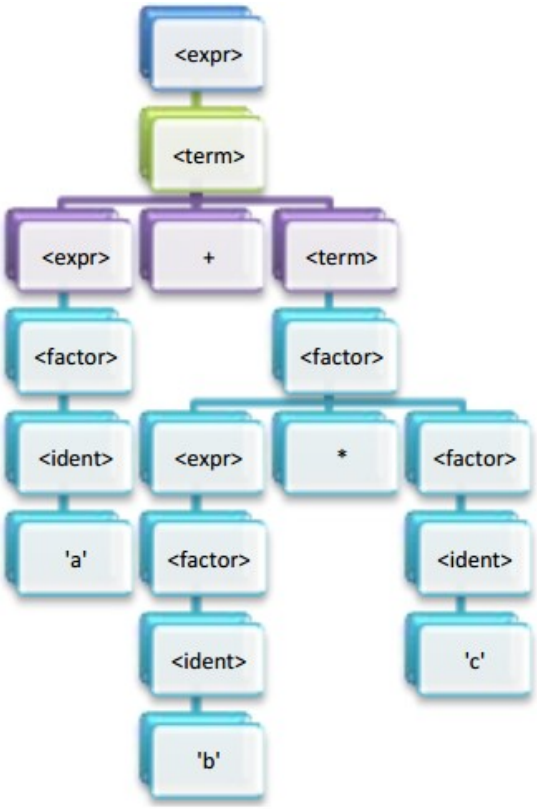
<term> ::= <factor> | <expr>+<term>

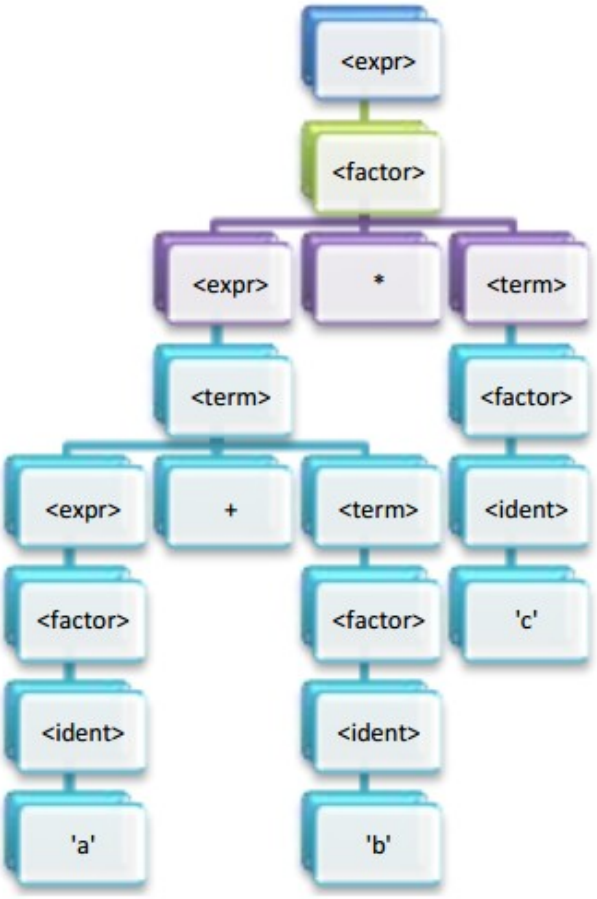
<factor> ::= <ident> | (<expr>)| <expr> *<factor>

<ident> ::= a| b| c

Answer:

For example for the expression a+b*c we can find two different derivation tree.





Make it Unambiguous:

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

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

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

<ident> ::= **a**| **b** | **c**

11. Consult [Naur63] to see how Algol solves the dangling else problem.

Answer:

Dangling `else` problem -- There are two possible meanings for the statement

`if` <cond1> `then if` <cond2> `then` <stmt1> `else` <stmt2>

```
1. if <cond1> then
       if <cond2> then
          <stmt1>
       else
            <stmt2>
```

```
2. if <cond1> then
       if <cond2> then
          <stmt1>
   else
     <stmt2>
```

Algol's simplistic solution: A conditional statement (if) is not allowed as the statement in a conditional statement. Instead, the intended structure must be explicitly shown using a *block* (begin-end pair):

```
1. if <cond1> then
       begin
          if <cond2> then
               <stmt1>
          else
            <stmt2>
   end
```

```
2. if <cond1> then
      begin
         if <cond2> then
         <stmt1>
      end
   else
      <stmt2>
```

Other languages solve the problem by:

- Adopting a convention, usually that an `else` goes with the nearest preceding unmatched `if` (e.g., PL/I, C).
- Using fully bracketed structures (e.g., Modula-2, Ada).

```
IF <cond1> THEN
   IF <cond2> THEN
      <stmt1>
   ELSE
      <stmt2>
   END
END
```

```
IF <cond1> THEN
   IF <cond2> THEN
      <stmt1>
   END
ELSE
   <stmt2>
END
```

page 29:

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]

Answer: