PL Assignment

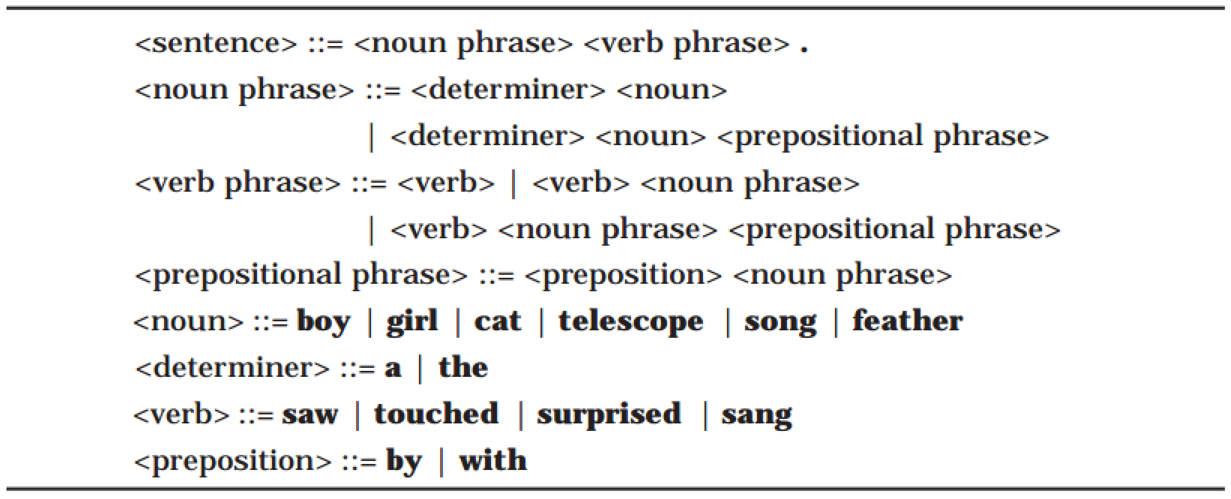
Arash Yadegari (94131092)

October 13, 2016

Formal syntax and semantics

Chapter 1, Page 8

Problem 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**

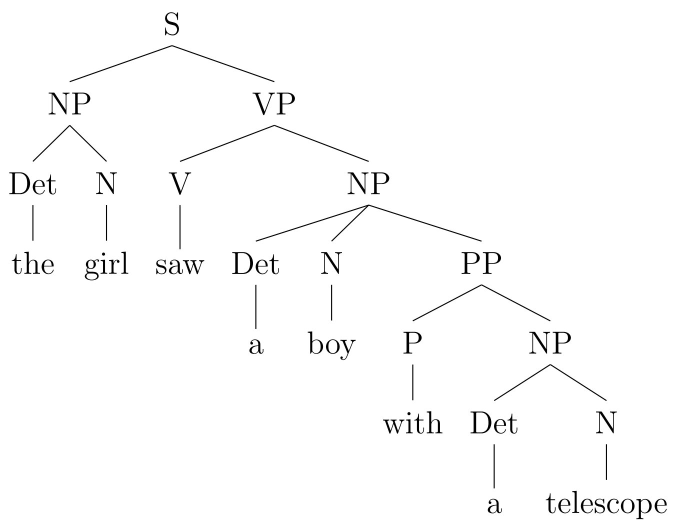
<determiner> ::= **a** | **the**

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

<preposition> ::= **by** | **with**

Figure 1.1: An English Grammar

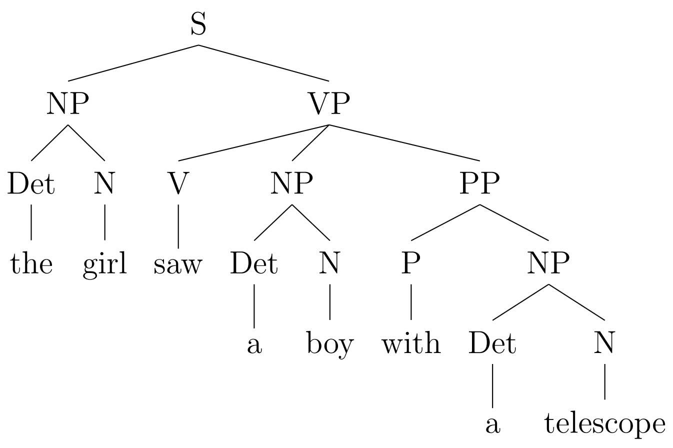
Solution :
Tree #1:



Derivation #1:
<sentence>

$\Longrightarrow$ <noun phrase> <verb phrase> .
$\Longrightarrow$ <determiner> <noun> <verb phrase> .
$\Longrightarrow$ the <noun> <verb phrase> .
$\Longrightarrow$ the girl <verb phrase> .
$\Longrightarrow$ the girl <verb> <noun phrase> .
$\Longrightarrow$ the girl saw <noun phrase> .
$\Longrightarrow$ the girl saw <determiner> <noun> <prepositional phrase> .
$\Longrightarrow$ the girl saw a <noun> <prepositional phrase> .
$\Longrightarrow$ the girl saw a boy <prepositional phrase> .
$\Longrightarrow$ the girl saw a boy <preposition> <noun phrase> .
$\Longrightarrow$ the girl saw a boy with <noun phrase> .
$\Longrightarrow$ the girl saw a boy with <determiner> <noun> .
$\Longrightarrow$ the girl saw a boy with a <noun> .
$\Longrightarrow$ the girl saw a boy with a telescope .

Tree #2:



Derivation #2:
<sentence>

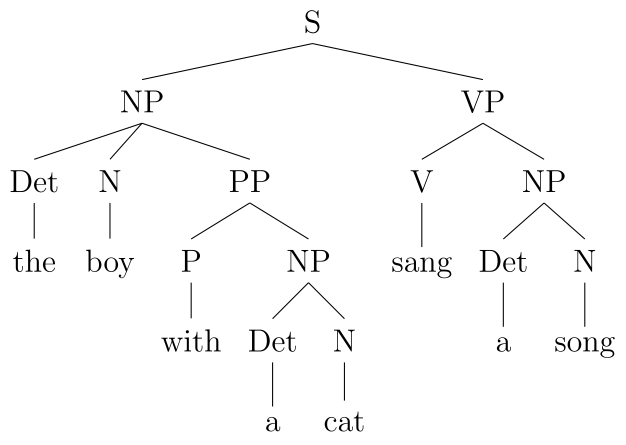
$\Longrightarrow$ <noun phrase> <verb phrase> .
$\Longrightarrow$ <determiner> <noun> <verb phrase> .
$\Longrightarrow$ the <noun> <verb phrase> .
$\Longrightarrow$ the girl <verb phrase> .
$\Longrightarrow$ the girl <verb> <noun phrase> <prepositional phrase> .
$\Longrightarrow$ the girl saw <noun phrase> <prepositional phrase> .
$\Longrightarrow$ the girl saw <noun phrase> <prepositional phrase> .
$\Longrightarrow$ the girl saw <determiner> <noun> <prepositional phrase> .
$\Longrightarrow$ the girl saw a <noun> <prepositional phrase> .
$\Longrightarrow$ the girl saw a boy <prepositional phrase> .
$\Longrightarrow$ the girl saw a boy <preposition> <noun phrase> .
$\Longrightarrow$ the girl saw a boy with <noun phrase> .
$\Longrightarrow$ the girl saw a boy with <determiner> <noun> .
$\Longrightarrow$ the girl saw a boy with a <noun> .
$\Longrightarrow$ the girl saw a boy with a telescope .

Problem 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.

Solution :
Derivation #1:
<sentence>
$\Longrightarrow$ <noun phrase> <verb phrase> .
$\Longrightarrow$ <determiner> <noun> <prepositional phrase> <verb phrase> .
$\Longrightarrow$ the <noun> <prepositional phrase> <verb phrase> .
$\Longrightarrow$ the boy <prepositional phrase> <verb phrase> .
$\Longrightarrow$ the boy <preposition> <noun phrase> <verb phrase> .
$\Longrightarrow$ the boy with <noun phrase> <verb phrase> .
$\Longrightarrow$ the boy with <determiner> <noun> <verb phrase> .
$\Longrightarrow$ the boy with a <noun> <verb phrase> .
$\Longrightarrow$ the boy with a cat <verb phrase> .
$\Longrightarrow$ the boy with a cat <verb> <noun phrase> .
$\Longrightarrow$ the boy with a cat sang <noun phrase> .
$\Longrightarrow$ the boy with a cat sang <determiner> <noun> .
$\Longrightarrow$ the boy with a cat sang a <noun> .
$\Longrightarrow$ the boy with a cat sang a song .
Derivation #2:
<sentence>
$\Longrightarrow$ <noun phrase> <verb phrase> .
$\Longrightarrow$ <noun phrase> <verb> <noun phrase> .
$\Longrightarrow$ <noun phrase> <verb> <determiner> <noun> .
$\Longrightarrow$ <noun phrase> <verb> <determiner> song .
$\Longrightarrow$ <noun phrase> <verb> a song .

$\Longrightarrow$ <noun phrase> sang a song .
$\Longrightarrow$ <determiner> <noun> <prepositional phrase> sang a song .
$\Longrightarrow$ <determiner> <noun> <preposition> <noun phrase> sang a song .
$\Longrightarrow$ <determiner> <noun> <preposition> <determiner> <noun> sang a song.
$\Longrightarrow$ <determiner> <noun> <preposition> <determiner> cat sang a song .
$\Longrightarrow$ <determiner> <noun> <preposition> a cat sang a song .
$\Longrightarrow$ <determiner> <noun> with a cat sang a song .
$\Longrightarrow$ <determiner> boy with a cat sang a song .
$\Longrightarrow$ the boy with a cat sang a song .

Both Derivation's Tree:



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

Solution :

a) $\{a^n b^n | n \in \mathbb{N}\}$
b) palindrome strings where length is $2K; K \in \mathbb{W}$
c) number of a's and b's in string is equal (greather than zero)

Problem 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>

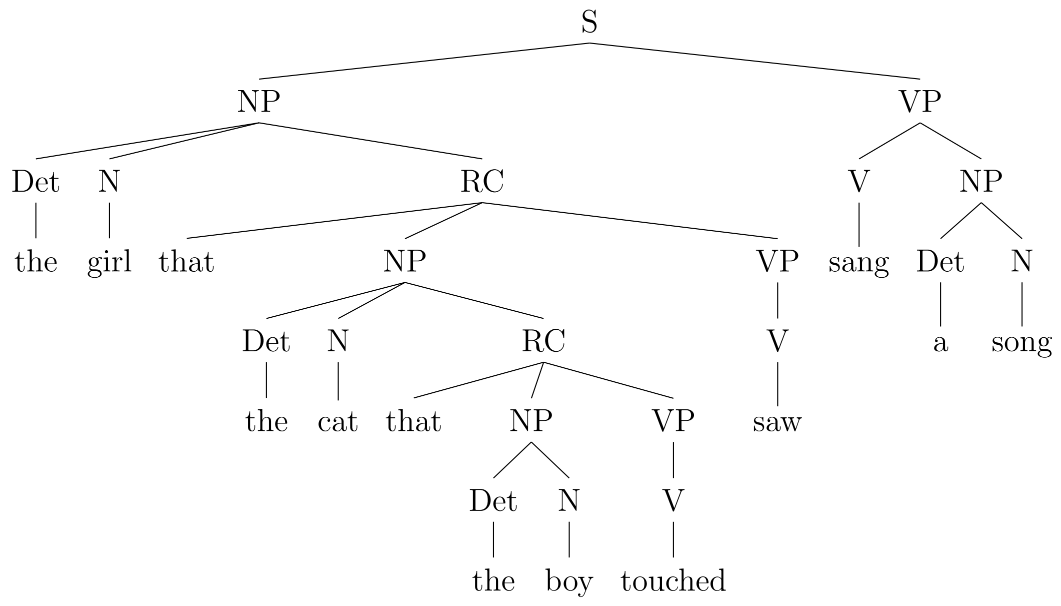
<noun> ::= **boy** | **girl** | **cat** | **telescope** | **song** | **feather**

<determiner> ::= **a** | **the**

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

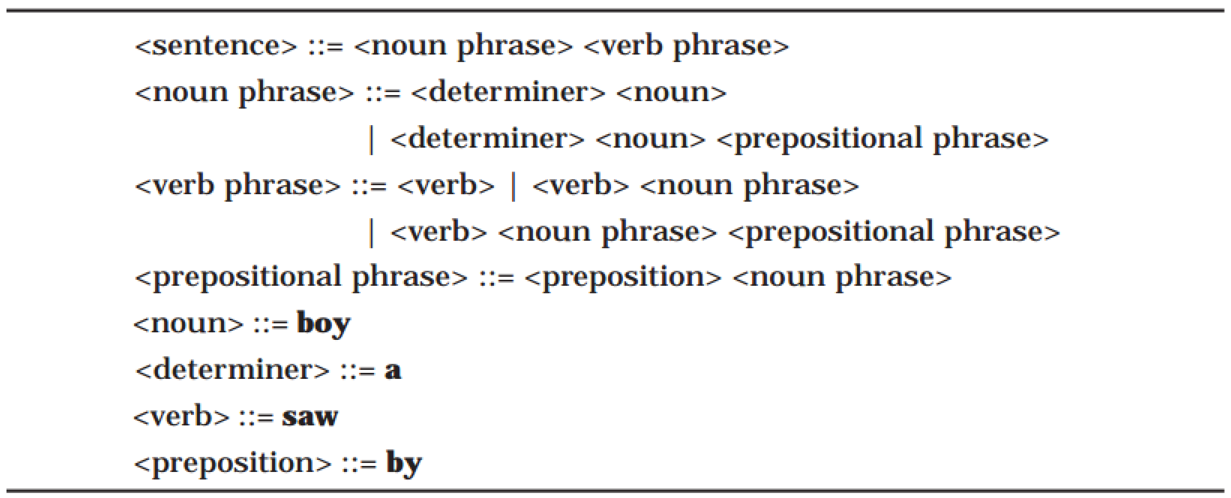
Readers familiar with computation theory may show that the language generated by this grammar is context-free but not regular.

Solution :
Derivation Tree:



left side of all rules has just one non terminal and right sides are strings, so grammar is context-free.
We use the pumping lemma to obtain a contradiction x="the girl that the cat that the boy touched", y="saw", z="sang a song", we see that xy^2z is not in L.

Problem 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>::= $a_1a_2a_3...a_n$ <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.



<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**

<determiner> ::= **a**

<verb> ::= **saw**

<preposition> ::= **by**

Figure 1.1: An English Grammar

Solution :
We can convert grammar to:

<sentence> ::= <noun phrase> <verb phrase>
<noun phrase> ::= a boy | a boy <prepositional phrase>
<verb phrase> ::= saw | saw a boy | saw a boy <prepositional phrase>
<prepositional phrase> ::= by a boy | by a boy <prepositional phrase>

Regular Expression of Language :

a boy (by a boy)* saw (a boy)? (by a boy)*

We can show regular grammars with regular expressions and concatenation of two regular expression is another regular expression.

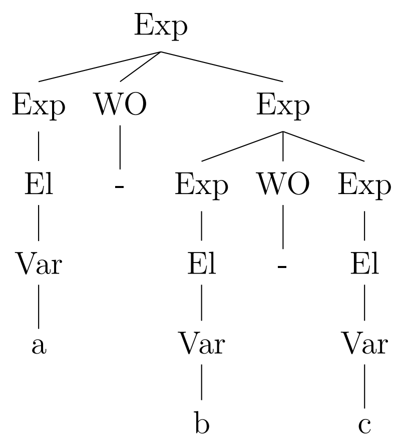
Chapter 1, Page 16

Problem 2. Consider the following specification of expressions:

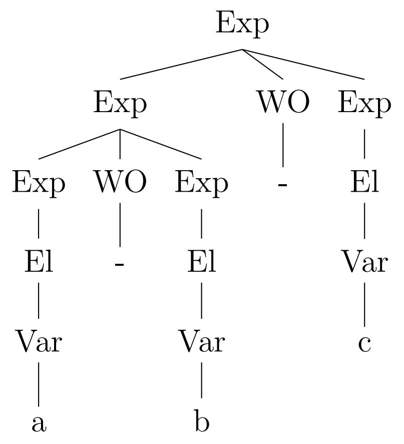
<expr> ::= <element> | <expr> <weak op> <expr>
<element> ::= <numeral> | <variable>
<weak op> ::= + | −

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

Solution :
Derivation Tree #1:



Derivation Tree #2:



Wren specification introduces a new non terminal called ”term” and provides one side (left) recursion for expressions. so expression and term rules are:
<expr> ::= <term> | <expr> <weak op> <term>
<term> ::= <element>

Problem 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
```

Solution :
Context-free Errors:
in ”program errors was”, ”was” must be ”is”

in "var p,b ; boolean;", ";" after "b" must be ":"
in "if b≠0 then p:=true else p:=(a+1);", "≠" must be "<>", ";" at the end must be eliminated, at the end of statement must be "end if"

Context-sensitive Errors:
variable "b" declared twice as integer and boolean
variable "b" is boolean but compared with integer ("0")
variable "p" is boolean but it's assigned with integer value (in else statement)

Semantic Errors:
type of "then" and "else" part of "if" statement are not consistent
variable "q" used but not declared
variable "b" and "p" (and also "q") are not initialized

Problem 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.

Solution :
a) − (higher), +, * (lower)
b) − : right-to-left, + : left-to-right, * : right-to-left
c) terminals can be in parantheses (makes no sense because can be eliminated), terminals with "−" operation between them can be in parantheses (makes no sense because minus has highest priority), nested parantheses can be in parantheses (makes no sense because can be eliminated). so we can't have "+" or "*" operation between elements in parantheses.

Problem 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>

Solution :
Eliminating left recursion in <identifier>:
<identifier> ::= <letter> | <letter> <identifier> | <letter> <identifier2> | <letter> <identifier>

<identifier2>
<identifier2> ::= <digit> <identifier2> | ε
<numeral> ::= <digit> | <digit> <numeral>
<letter> can be replaced by all letter terminals and <digit> can be replaced by all digit terminals.

Problem 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].

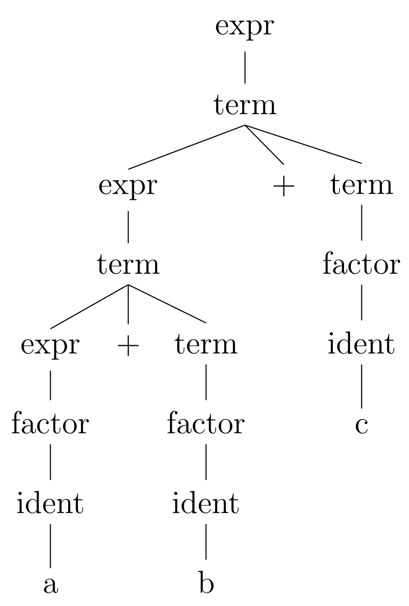
Write a BNF specification for this language of list expressions. Assume that <integer> has already been defined. The conformity of lists for the arithmetic operations is not handled by the BNF grammar since it is a context-sensitive issue.

Solution :
every operation with higher priority goes down in grammar levels. for right and left associativity we use right and left recursion, respectively.
<exp> ::= <term> @ <exp> | <term>
<term> ::= <term> + <obj> | <term> - <obj> | <obj>
<obj> ::= <obj> * <thing> | <thing>
<thing> ::= <list> | (<exp>)
<list> ::= [<items>] | [ ]
<items> ::= <integer> | <items> , <integer>

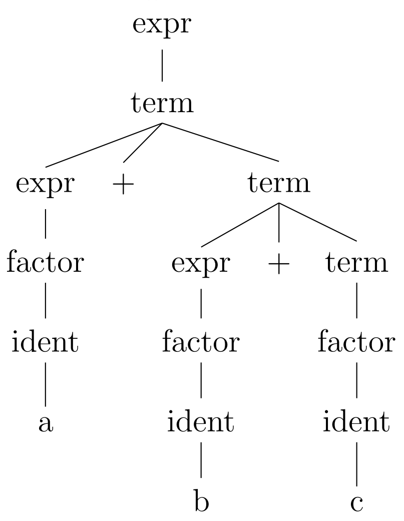
Problem 10. Show that the following grammar for expressions is ambiguous and provide an alternative unambiguous grammar that defines the same set of expressions.

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

Solution :
We consider following expression : a + b + c
We have more than one derivation trees :
Derivation Tree #1:



Derivation Tree #2:

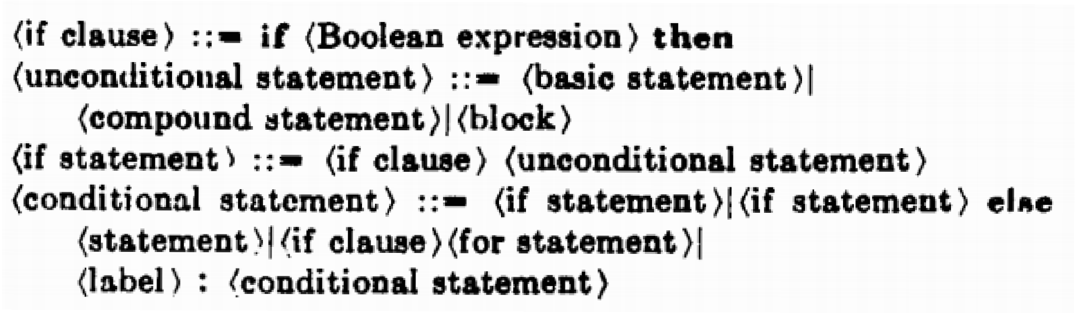


so grammar is ambiguous.
Alternative unambiguous grammar:

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

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

Solution :


⟨if clause⟩ ::= **if** ⟨Boolean expression⟩ **then**
⟨unconditional statement⟩ ::= ⟨basic statement⟩|
⟨compound statement⟩|⟨block⟩
⟨if statement⟩ ::= ⟨if clause⟩ ⟨unconditional statement⟩
⟨conditional statement⟩ ::= ⟨if statement⟩|⟨if statement⟩ **else**
⟨statement⟩|⟨if clause⟩⟨for statement⟩|
⟨label⟩ : ⟨conditional statement⟩

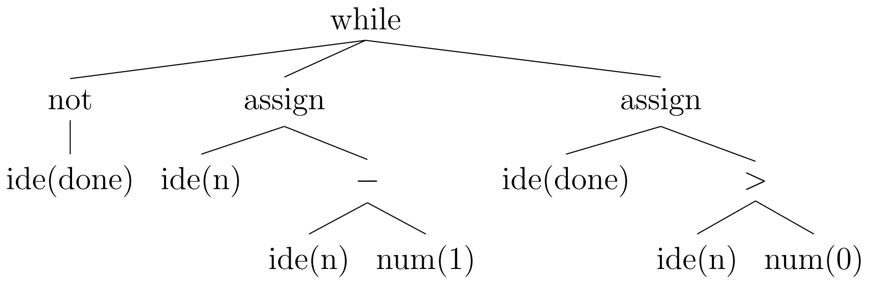
Algol60 defines "conditional statement" that includes "if" statement. inside "then" part of "if" can be "unconditional statement" that can be "basic statement" which doesn't contain condition, or "compound statement" which has block to specify scope, or block which specifies scope.
so we cannot write "if B1 then if B2 then S1 else S2;" and we must specify that "else" belongs to first or second "if" (ex. "if B1 then begin if B2 then S1 else S2; end").

Chapter 1, Page 29

Problem 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]

Solution :
Abstract Syntax Tree:



Chapter 3, Page 71

Problem 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 (≥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>.

Solution :

```
<string literal> ::= <numeral> H <string>
        condition : <numeral>.Val == <string>.Len
<numeral> ::= <digit>
                    <numeral>.Val ← <digit>.Val
                | <numeral>$_2$ <digit>
                    <numeral>.Val ← <numeral>$_2$.Val * 10 + <digit>.Val
<digit> ::=       0
                    <digit>.Val ← 0
                | 1
                    <digit>.Val ← 1
                | ...
                | 9
                    <digit>.Val ← 9
<string> ::=    <char>
                    <string>.Len ← <char>.Len
                | <char> <string>$_2$
                    <string>.Len ← <char>.Len + <string>$_2$.Len
<char> ::=        a
                    <char>.Len ← 1
                | b
                    <char>.Len ← 1
                | ...
                | z
                    <char>.Len ← 1
```

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

Solution :

```
<string literal> ::= <numeral> H <string>
        <string>.InhLen ← <numeral>.Val
<numeral> ::= <digit>
                    <numeral>.Val ← <digit>.Val
                | <numeral>$_2$ <digit>
                    <numeral>.Val ← <numeral>$_2$.Val * 10 + <digit>.Val
<digit> ::=       0
                    <digit>.Val ← 0
                | 1
                    <digit>.Val ← 1
                | ...
                | 9
                    <digit>.Val ← 9
<string> ::=    <char>
                    condition : <string>.InhLen == <char>.Len
                | <char> <string>_2
                    <string>_2.InhLen ← <string>.InhLen − <char>.Len
<char> ::=        a
                    <char>.Len ← 1
                | b
                    <char>.Len ← 1
                | ...
                | z
                    <char>.Len ← 1
```

Problem 4. The following BNF specification defines the language of Roman numerals less than 1000:

```
<roman> ::= <hundreds> <tens> <units>
<hundreds> ::= <low hundreds> | CD | D <low hundreds> | CM
<low hundreds> ::= ε | <low hundreds> C
<tens> ::= <low tens> | XL | L <low tens> | XC
<low tens> ::= ε | <low tens> X
<units> ::= <low units> | IV | V <low units> | IX
<low units> ::= ε | <low units> I
```

Define attributes for this grammar to carry out two tasks:

a) Restrict the number of X's in <low tens>, the I's in <low units>, and the C's in <low hundreds> to no more than three.

b) Provide an attribute for <roman> that gives the decimal value of the Roman numeral being defined.

Define any other attributes needed for these tasks, but do not change the BNF grammar.

Solution :

a)

```
<low tens> ::= ε
                    <low tens>.Len ← 0
                | <low tens>_2 X
                    <low tens>.Len ← <low tens>_2.Len + 1
                    condition: <low tens>.Len ≤ 3

<low units> ::= ε
                    <low units>.Len ← 0
                | <low units>_2 I
                    <low units>.Len ← <low units>_2.Len + 1
                    condition: <low units>.Len ≤ 3
```

```
    <low hundreds> ::= ε
                          <low hundreds>.Len ← 0
                        | <low hundreds>₂ C
                          <low hundreds>.Len ← <low hundreds>₂.Len + 1
                          condition: <low hundreds>.Len ≤ 3
```

b)

```
<roman> ::= <hundreds> <tens> <units>
                          <roman>.Val ← <hundreds>.Val + <tens>.Val + <units>.Val
<hundreds> ::= <low hundreds>
                          <hundreds>.Val ← <low hundreds>.Val
                        | CD
                          <hundreds>.Val ← 400
                        | D <low hundreds>
                          <hundreds>.Val ← 500 + <low hundreds>.Val
                        | CM
                          <hundreds>.Val ← 900
<low hundreds> ::= ε
                              <low hundreds> ← 0
                            | <low hundreds>₂ C
                              <low hundreds>.Val ← <low hundreds>₂.Val + 100
<tens> ::= <low tens>
                          <tens>.Val ← <low tens>.Val
                        | XL
                          <tens>.Val ← 40
                        | L <low tens>
                          <tens>.Val ← 50 + <low tens>.Val
                        | XC
                          <tens>.Val ← 90
<low tens> ::= ε
                          <low tens> ← 0
                        | <low tens>₂ X
                          <low tens>.Val ← <low tens>₂.Val + 10
<units> ::= <low units>
                          <units>.Val ← <low units>.Val
                        | IV
                          <units>.Val ← 4
                        | V <low units>
                          <units>.Val ← 5 + <low units>.Val
                        | IX
                          <units>.Val ← 9
<low units> ::= ε
                          <low units> ← 0
                        | <low units>₂ I
                          <low units>.Val ← <low units>₂.Val + 1
```