In the name of God

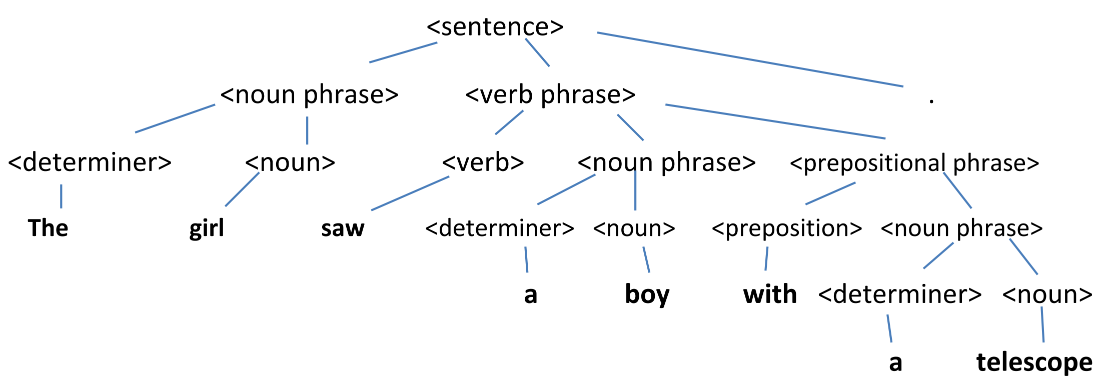
PL homework #2

Seyed Mohammad Mehdi Ahmadpanah – 9031806

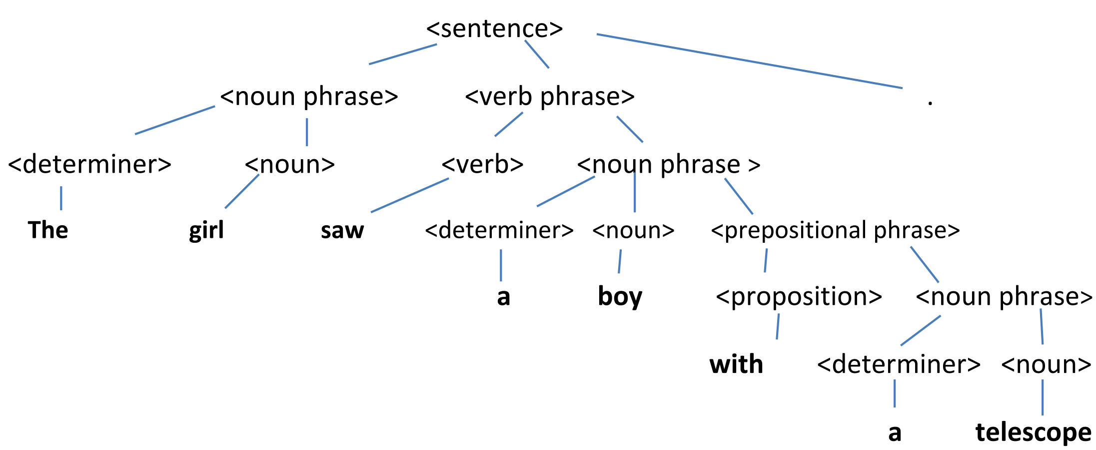
Page 8 :

1) The girl saw a boy with a telescope

I . first derivation tree :



II . second derivation tree :



3)

linguistics : the science of language / is the scientific study of human language (زبان شناسی)

semiotics : is study of signs & sign processes. (نماد شناسی)

grammar : is the set of structural rules that governs the composition of clauses , phrases & words in any given natural language. (دستور زبان ، گرامر)

syntax : is the study of the principles & processes by which sentences are constructed in particular languages. (نحو ، ترکیب)

semantics : is the study of meaning. (معناشناخت)

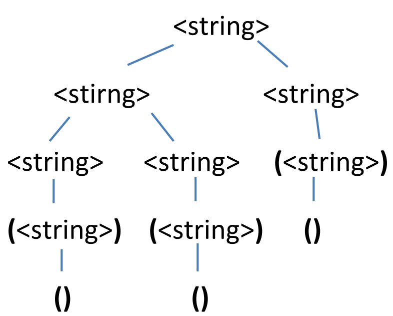
pragmatics : is a subfield of linguistics which studies the ways in which context contributes to meaning. (کاربست ، واقع بینی)

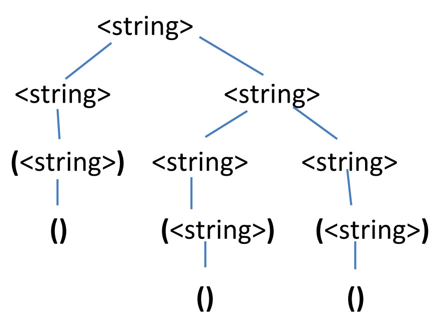
5) <sentence> => **a**<thing>**bc** => **ab**<thing>**c** => **ab**<other>**bcc** => **a**<other>**bbcc** => **aa**<thing>**bbcc** => **aab**<thing>**bcc** => **aabb**<thing>**cc** => **aabb**<other>**bccc** => **aab**<other>**bbccc** => **aa**<other>**bbbccc** => **aaabbbccc**

6)

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

a is ambiguous! (two derivation tree for “()()()” & b is not ambiguous.





7)

T = { a , b }

a) <string> ::= **a**<string>**b** | **ab** => L = { $a^n b^n$ | n >= 1 }

b) <string> ::= **a**<string>**a** | **b**<string>**b** | ε => L = { xx^R | x ∈ { a , b }* }

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

<A> ::= **a** | **a**<string> | **b**<A><A> => L = { x | $N_a(x) = N_b(x)$ }

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

9)

<sentence> ::= <noun phrase> <verb phrase> **.**
<noun phrase> ::= <determiner> <noun>| <determiner> <noun> <prepositional phrase>
<verb phrase> ::= <verb> | <verb> <noun phrase> | <verb><nounphrase><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**

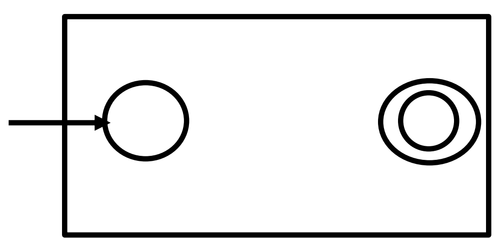
قاعده دوم را می توان با جایگذاری Terminal های <determiner> و <noun> و گرامر چهارم با جایگذاری <preposition> به یک گرامر منظم تبدیل کرد :

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

اگر R1 و R2 عبارت منظم باشند ، پس می توانیم برای آنها یک DFA بسازیم که آنها را می پذیرد:

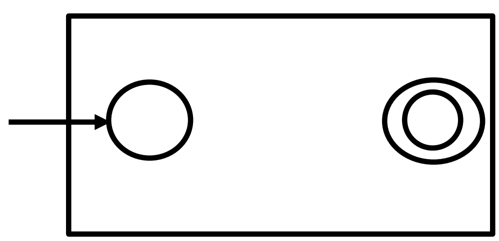
R1:

ماشینی که R1 را می پذیرد.



R2:

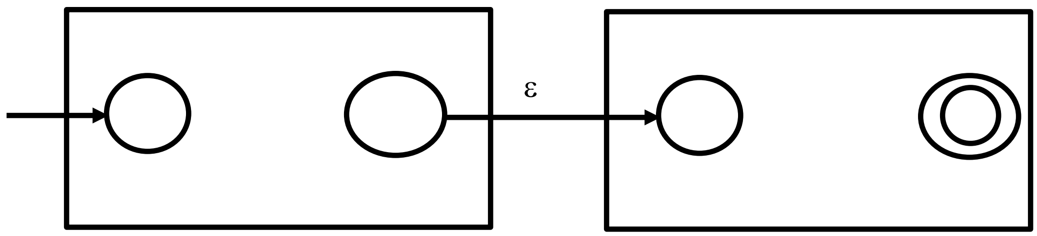
ماشینی که R2 را می پذیرد.



ما می توانیم یک ماشین DFA بسازیم که(concatenation(R1,R2 را بپذیرد:

concatenation(R1,R2):

ماشینی که R1 را می پذیرد.　　　　ماشینی که R2 را می پذیرد



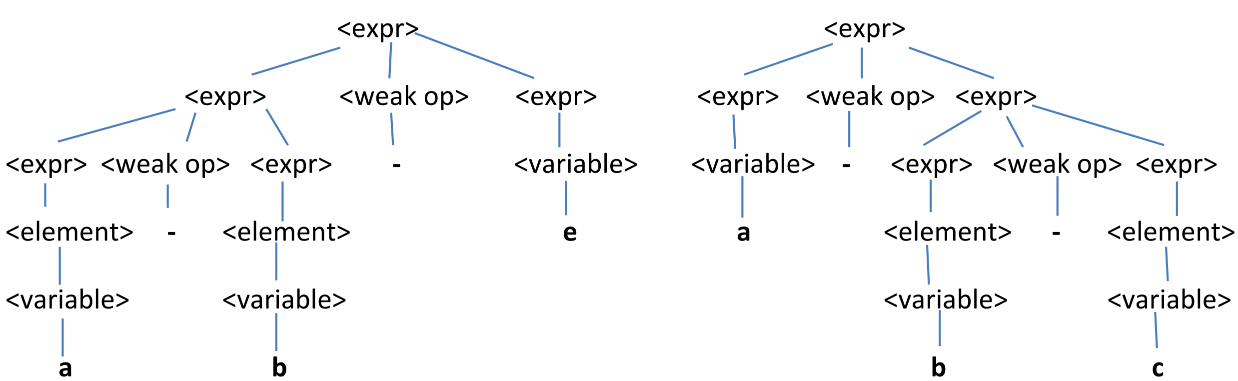
پس concatenate دو گرامر منظم یک گرامر منظم است.

پس گرامر سوم که از پیوند گرامر دوم و چهارم ساخته شده یک گرامر منظم است. گرامر اول که از پیوند گرامر دوم و سوم ساخته شده هم یک گرامر منظم است. پس گرامر زبان انگلیسی یک گرامر نوع سوم است.

Page 16:

2)

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



Two derivation tree for “a-b-c” => ambiguity!

در زبان wren چون در گرامرش ابهامی وجود ندارد و عملگرها سطح بندی و نحوه محاسبه دو عملگر منفی از چپ به راست است پس مشکل بالا را ندارد. یعنی ابتدا حاصل a-b محاسبه می شود سپس حاصل از c کم می شود.

3)

```
 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
```

context-free errors :

was -> **is**

var p,b ; **boolean** -> **var** p,b : **Boolean**

else p:=(a+1) **end if** ;

if b≠0 **then** -> **if** b<>0 **then**

context-sensitive :

b is **integer** and **boolean !!?!**

p := (a+1); -> wrong (p is boolean and a+1 is integer)

write q -> what is q?!

write p -> p is not integer!

semantics : b & p is not initialized !

5)

a) operations priority : 1) - 2) + 3) *

b) * : Right to Left + : Left to Right - : Right to Left

c) چون محاسبه پرانتز اولویت اول است ، پس در دورترین سطح از ریشه قرار می گیرد.

درون پرانتز فقط می توان از عملگر "-" استفاده کرد چون

<element>::= a | b | c | d | (<object>)

<object>::=<element> | <element> - <object>

پس هیچ وقت دو گرامر اخیر به <expr> یا <thing> نمی رسد که از عملگر های * و + بتوان استفاده کرد.

7)

در identifier به جای nonterminal <letter> , <digit> ، می توان تک تک terminalهای حروف و ارقام را گذاشت :

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

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

9)

<expr>::=<weak op> **|** <var> @ <expr>

<weak op>::=<strong op> **|** <weak op> + <var> **|** <weak op> - <var>

<strong op>::=<var> **|** <strong op> * <var>

<var>::=[<term>] **|** | (<expr>) **|** []

<term>::=<integer> , <term>**|**<num>

<num>::=0 | 1 | … | 9

10)

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

<expr> → <term> → <factor> → <ident> → **a**

<expr> → <factor> → <ident> → **a**

- ⇨ **Two derivation tree for an expression! (ambiguous)**

Not ambiguous grammar :

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

11)

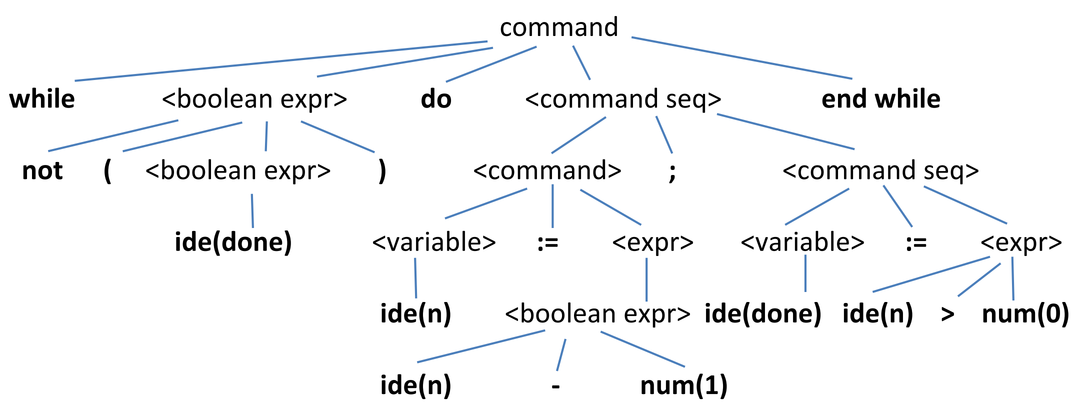
The problem can be solved by making explicit the link between an else and its if, within the syntax. This usually helps avoid human errors. ALGOL solutions are:

Having an "end if" statement or Disallowing the statement following a then to be an if itself (it may however be a pair of statement brackets containing nothing but an if-then-close).

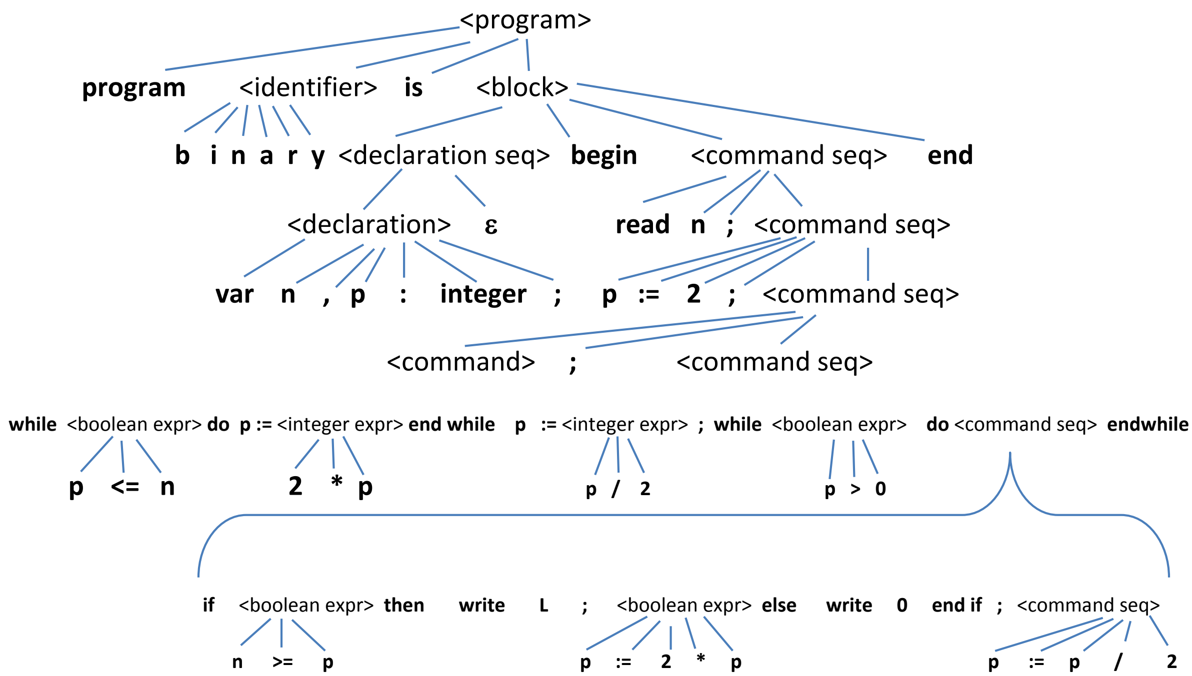
(بطور خلاصه ، یا با گذاشتن عبارت ذخیره شده end if یا با گذاشتن آکلاد باز و بسته ، این مشکل در ALGOL های نسخه های مختلف حل شده است.)

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]



3)
Draw an abstract syntax tree for the following Wren program:
program binary **is**
var n,p : **integer** ;
begin
read n; p := 2;
while p<=n **do** p := 2*p **end while** ;
p := p/2;
while p>0 do
if n>= p **then write** 1; n := n–p **else write** 0 **end if** ;
p := p/2
end while
end



Chapter 3 of Sebesta's book (Concepts of Programming Languages 10th-Sebesta):

19.
Write an attribute grammar whose BNF basis is that of Example 3.6 in Section 3.4.5 but whose language rules are as follows: Data types cannot be mixed in expressions, but assignment statements need not have the same types on both sides of the assignment operator.

Replace the second semantic rule with:

<var>[2].env ←<expr>.env
<var>[3].env ←<expr>.env
<expr>.actual_type ←<var>[2].actual_type
predicate: <var>[2].actual_type = <var>[3].actual_type

20.
Write an attribute grammar whose base BNF is that of Example 3.2 and whose type rules are the same as for the assignment statement example of Section 3.4.5.

A Grammar for Simple Assignment Statements
<assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr>
| <id> * <expr>
| (<expr>)
| <id>

Attribute Grammar for above grammar :
1. Syntax rule: <assign> → <id> = <expr>
Semantic rule: <expr>.expected_type ← <id>.actual_type
2. Syntax rule: <expr>[1] → <id> + <expr>[2]
Semantic rule: <expr>[1].actual_type ← if (<id>.actual_type = int) and(<expr>[2].actual_type = int)
then int
else real
end if
Predicate: <expr>[1].actual_type == <expr>[1].expected_type
3. Syntax rule: <expr> → <id> * <expr>
Semantic rule: <expr>[1].actual_type ← if (<id>.actual_type = int) and(<expr>[2].actual_type = int)
then int
else real
end if
Predicate: <expr>[1].actual_type == <expr>[1].expected_type
4. Syntax rule: <expr>[1] → (<expr>[2])
Semantic rule: <expr>[1].actual_type == <expr>[2].actual_type
Predicate: <expr>[1].actual_type == <expr>[1].expected_type
5.Syntax rule: <expr> → <id>
Semantic rule: <expr>.actual_type ← <id>.actual_type
Predicate: <expr>.actual_type == <expr>.expected_type
6. Syntax rule: <id> → A | B | C
Semantic rule: <id>.actual_type ← look-up(<id>.string)