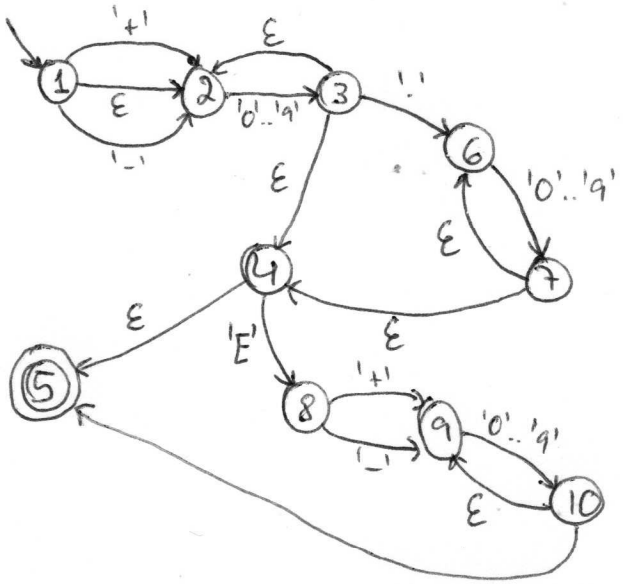
I

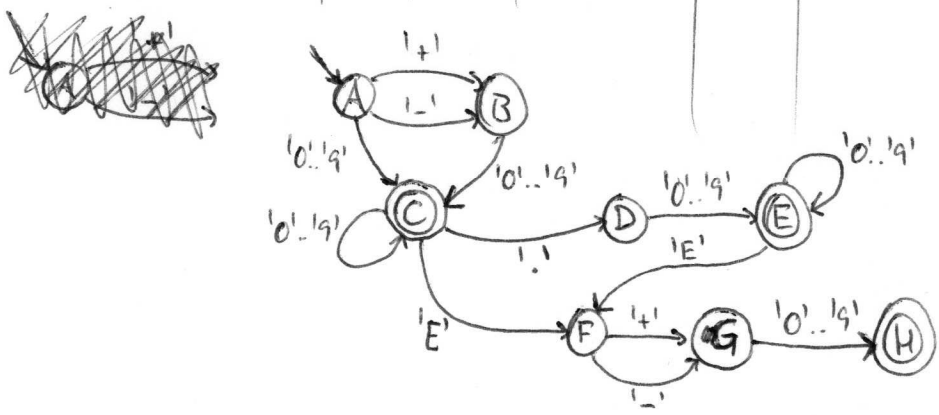
a) inteiro → (+|-)?[0-9]+
decimal → (\.[0-9]+)
expoente → (E(+|-)[0-9]+)
expressão → inteiro decimal? expoente?

b)



c)

	Estado	'+'	'-'	'0'..'9'	'.'	'E'
A	>1,2	2	2	3,2,4,5		
B	2			3,2,4,5		
C	* 3,2,4,5			3,2,4,5	6	8
D	6			7,6,4,5		
E	* 7,6,4,5			7,6,4,5		8
F	8	9	9			
G	9			10,9,5		
H	* 10,9,5			10,9,5		



d)

Z → Real '$'

Real → Inteiro Decimal Expoente

Inteiro → '+' num
| '-' num
| num

Decimal → ε
| '.' num

Expoente → ε
| 'E' '+' num
| 'E' '-' num

T = { [illegible] '+', '-', num, 'E', '.' }

2. a)

```
%x attr
%%
"<books"      printf("<books>");
"<book"       {printf("\t<book "); BEGIN attr;}
"<title"      printf("\t\t<title>");
"<author"     printf("\t\t<author>");
"</books"     printf("</books>");
"
```

teste PL 2013 ②

2 a e b)

```
%{
int S[100]={0};
int sp=0;
%}

%x attr
%%
"(books"        {printf("<books>"); S[sp++]=1;}
")books"        {printf("</books>"); if(S[sp--]!=1) erro();}
"(book"         {printf("\t<book "); S[sp++]=2; BEGIN attr;}
")book"         {printf("\t</book>"); if(S[sp--]!=2) erro();}
"(title"        {printf("\t\t<title>"); S[sp++]=3;}
")title"        {printf("</title>"); if(S[sp--]!=3) erro();}
"(author"       {printf("\t\t<author>"); S[sp++]=4;}
")author"       {printf("</author"); if(S[sp--]!=4) erro();}
"(isbn"         {printf("\t\t<isbn"); S[sp++]=5;}
")isbn"         {printf("</isbn>"); if(S[sp--]!=5) erro();}
\-\n            printf("\n");
\-.+            printf("%s", yytext+1);
[␣\n\t]         ;
<attr>A[^␣]+    printf(" %s=", yytext+1);
<attr>[␣].+     printf("'%s'", yytext+1);
<attr>[^␣A]     {BEGIN 0; printf(">");}
%%
int yywrap(){
    if(sp!=0) erro();
}
```

␣ - espaço em branco

Nota: tudo o que tem a ver com o array S é da alínea b.
usei uma stack em que cada tag tem um número associado.

3

a) P={ Z → Anv

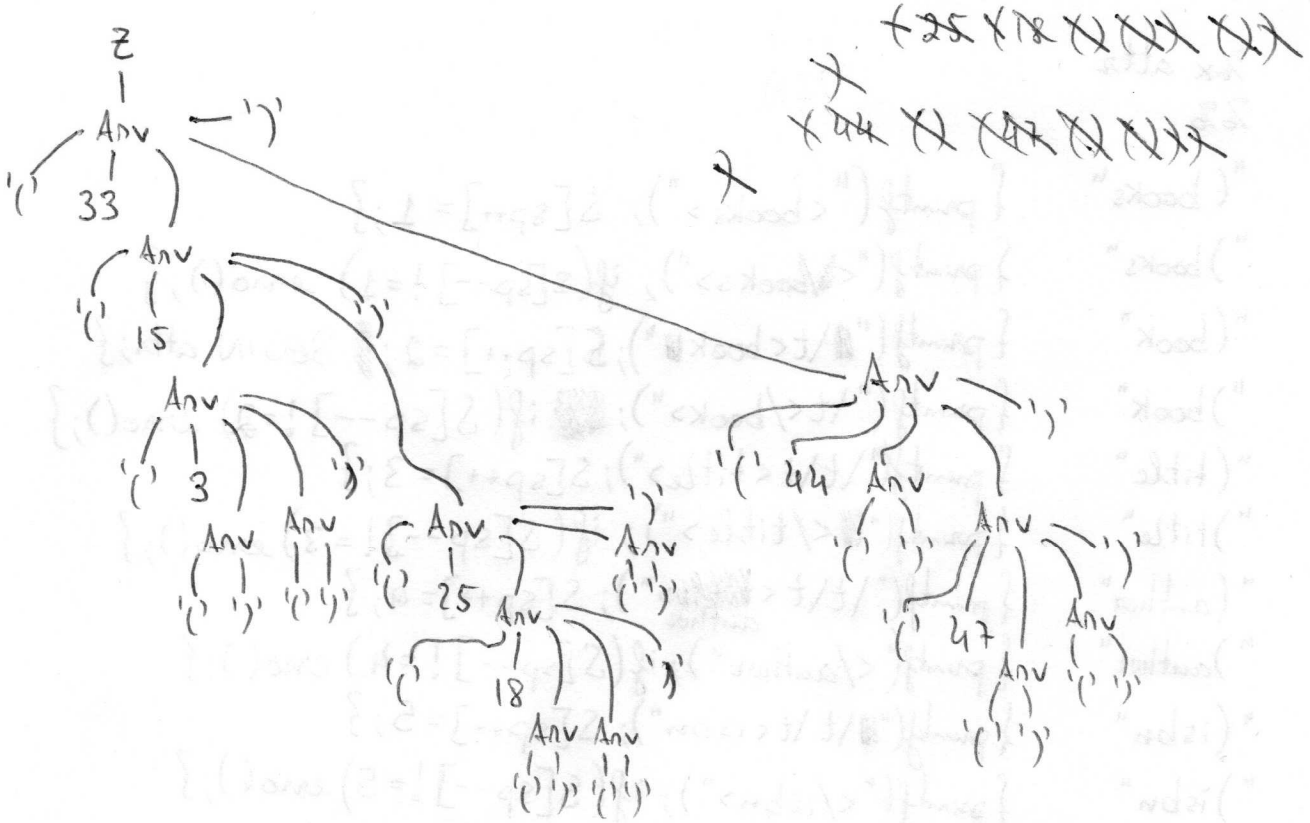
Anv → '(' ')'

| '(' num Anv Anv ')'

}

G = { T = {'(', ')', num},
NT = { Z, Anv },
S = Z, P }

b)



c, d, e)

árvore.l

```
%{
   #include "y.tab.h"
%}
%%
\(      return '(';
\)      return ')';
[0-9]+  {yylval.num = atoi(yytext); return NUM;}
[ \n\t] ;
<<EOF>>  return '$';
.       ;
```

teste PL 2013 ③

arvore.y

```
%{ #include "tree.h" }
%token NUM
%union {
    int num; Tree a;
}
%type<num> NUM
%type<a> Arv
%start Z
%%
Z : Arv { fazerCoisas($1); };
Arv : '(' ')' { $$ = NULL; }
    | '(' NUM Arv Arv ')' { $$ = tree_nova($2, $3, $4); }
    ;
%%
int main(){
    yyparse();
    return 0;
}
```

tree.h

```
#include<stdio.h>
#include<stdlib.h>
typedef struct sTree *Tree;
struct sTree {
    int n;
    Tree l;
    Tree r;
};
```

```
Tree tree_nova(int n, Tree l, Tree r){
    Tree res = (Tree) malloc(sizeof(struct sTree));
    res.n = n;
    res.l = l;
    res.r = r;
    return res;
}
```

```
void tree_print(Tree t){
    if(!t) return;
    tree_print(t->l);
    printf("%d ", t->n);
    tree_print(t->r);
}
```

```
int tree_soma(Tree t){
    if(!t) return 0;
    return
        tree_soma(t->l) +
        tree_soma(t->r) +
        t->n;
}
```

```
int tree_conta(Tree t){
    if(!t) return 0;

    return
        tree_conta(t->l)+
        tree_conta(t->r)+
        1;
}
```

```
int tree_val(Tree t){
    if(!t) return 1;
```

```
    if(s == '<')
```

```
int tree_val_right(Tree t, int n){
    if(!t) return 1;

    return
        t->n > n &&
        tree_val_right(t->l, n)&&
        tree_val_right(t->r, n);
}
```

```
void fazerCoisas(Tree t){
    if(!tree_val(t)){
        printf("
        erro();
        return;
    }
    tree_print(t);
    printf("\n");
    printf("somatório = %d\n média = %f\n",
            tree_soma(t),
            tree_media(t));
```

```
float tree_media(Tree t){ if(!t) return 0;
    return(
        ((float) tree_soma(t)) /
        ((float)tree_conta(t))
    );
}
```

```
int tree_val_left(Tree t, int n){
    if(!t) return 1;

    return
        t->n < n &&
        tree_val_left(t->l, n) &&
        tree_val_left(t->r, n);
}
```

```
int tree_val(Tree t){
    if(!t) return 1;
    if( tree_val_left(t->l, t->n)
        && tree_val_right(t->r, t->n))
        return
            tree_val(t->l) &&
            tree_val(t->r);
    return 0;
}
```

teste PL 2013 (4)

4 a)

$S \xrightarrow{P_1} A\ a\ S$ {c, d}
$P_2 \mid A$ {c, d}
$A \xrightarrow{P_3} B\ b\ A$ {c, d}
$P_4 \mid B$ {c, d}
$B \xrightarrow{P_5} c\ S\ c$ {c}
$P_6 \mid d$ {d}

b)

	a	b	c	d
S			P_1, P_2	P_1, P_2
A			P_3, P_4	P_3, P_4
B			P_5	P_6

~~tanto em S como em A, caso o lookahea~~

a intersecção de lookaheads de P_1 com P_2 e de P_3 com P_4 não é vazia, por isso existem conflitos e a gramática não é LL(1).

c)

$Z \xrightarrow{P_1} S\ \$$ {c, d}
$S \xrightarrow{P_2} A\ S'$ {c, d}
$S' \xrightarrow{P_3} a\ S$ {a}
$P_4 \mid \varepsilon$ {$, c} → ∩ dá vazio
$A \xrightarrow{P_5} B\ A'$ {c, d}
$A' \xrightarrow{P_6} b\ A$ {b}
$P_7 \mid \varepsilon$ {a, $, c} → ∩ dá vazio
$B \xrightarrow{P_8} c\ S\ c$ {c}
$P_9 \mid d$ {d} → ∩ dá vazio

gramática LL(1) válida

	a	b	c	d	$
Z			~~P_1~~	~~P_1~~	
S			P_2 ~~P_2~~		
S'	P_3		P_4		P_4
A			P_5 ~~P_5~~		
A'	P_7	P_6	P_7		P_7
B			P_8	P_9	

→ tabela para ajudar a fazer o recursivo descendente

d)

```
#include "tokens.h"
extern int yylex(); int prox_simb;
int main(){
    prox_simbolo = yylex();
    rec_Z();
    return 0;
}

int rec_S(){
    switch(prox_simb){
      case LC:
           LD: rec_A();
               rec_S2();
               break;
      default: erro(...);
    }
}
int rec_S2(){
    switch(prox_simb){
      case LA: rec_Term(LA);
               rec_S();
               break;
      case LC: break;
      case LEOF: break;
      default: erro(...);
    }
}
```

```
int rec_A(){
  switch(prox_simb){
    case LC:
    case LD: rec_B();
             rec_A2();
             break;
    default: erro(...);
  }
}
int rec_B(){
    switch(prox_simb){
      case LC: rec_Term(LC);
               rec_S();
               rec_Term(LC);
               break;
      case LD: rec_Term(LD);
               break;
      default: erro(...); } }
```

```
int rec_Term(int simb){
  if(prox_simb == simb)
     prox_simb = yylex();
  else
     erro(...);
  return 0;
}

int rec_Z(){
  switch(prox_simb){
    case LC:
    case LD: rec_S(); break;
    default: erro(...);
  }
}

int rec_A2(){
  switch(prox_simb){
    case LA:
    case LC:
    case LEOF: break;
    case LB: rec_Term(LB);
             rec_A();
             break;
    default: erro(...);
  }
}
```

tokens.h

```
// Letras
#define LA 1000
#define LB 1001
#define LC 1002
#define LD 1003
#define LEOF 1999
```