PCS2427 – Lógica Computacional

Projeto sobre autômatos de pilha estruturados

Prof.: João José Neto

Alunos: Nome N. USP

Arthur K. M. Sato 7700786

Tomaz de S. Silva 7631261

Junho de 2014

Sumário

[1. Analisador Léxico 3](#_Toc391328374)

[2. Analisador sintático 5](#_Toc391328375)

[3. Caso de teste 1 7](#_Toc391328376)

[4. Sequência de testes para caso 1 9](#_Toc391328377)

[5. Caso de teste 2 16](#_Toc391328378)

[6. Sequência de testes para caso 2 18](#_Toc391328379)

[7. Manual 23](#_Toc391328380)

[8. Anexos 24](#_Toc391328381)

## Analisador Léxico

Primeiramente foi escolhido o interpretador léxico como ponto de partida do projeto. Ele segue basicamente o esquema abaixo:



Figura 1: Diagrama do funcionamento do Léxico.

Primeiro ele checa se a pilha está vazia, se estiver ele prossegue a leitura dos caracteres de entrada normalmente. Caso a pilha esteja com algum valor é porque um token acabou de ser entregue, portanto uma nova cadeia cujo primeiro termo está na pilha deve ser lida. Como pode-se notar, o analisador léxico lê caractere por caractere, ignora se esse caractere é um espaço ou um enter. Se existe transição no autômato do léxico ele continua a leitura, caso contrário ele empilha o caractere lido, volta para o estado inicial do léxico e entrega o token para o analisador sintático.

O analisador léxico é alimentado por um arquivo que contém a informação do autômato do analisador léxico. Esse arquivo está estruturado da seguinte maneira:

Estado inicial do autômato

Sequência de transições na forma:

Estado atual elemento lido próximo estado

!

Após a sequência de estados, o símbolo “!” indica o fim da matriz de transição e nesse caso, como é um autômato finito simples, o sinal indica também o término do arquivo. O estado final é indicado por um ”\*” antes do nome do estado (\*1 e \*2 por exemplo indica dois estados finais 1 e 2).

## Analisador sintático

Posteriormente foi feito o analisador sintático. Esse recebe como parâmetros o token (fornecido pelo léxico) e o autômato de pilha a ser simulado. Inicialmente foi pensado o Analisador sintático como mostrado na figura 2 logo abaixo, porém por simplificação não foi feita a rejeição da cadeia. Ou seja, ele funciona para cadeias de entrada válidas.



Figura 2: Funcionamento básico do analisador sintático.

O formato do arquivo do autômato de pilha é o sequinte:

Nome das sub-máquinas que compõe o autômato de pilha seguida de “;”

Estado inicial da primeira sub-máquina

Sequência de transições de primeira sub-máquina seguida de “!”

Estado inicial da segunda sub-máquina

Sequência de transições de segunda sub-máquina seguida de “!”

...

Estado inicial da n-ésima sub-máquina

Sequência de transições de n-ésima sub-máquina seguida de “!”

Finalmente, nós fizemos a cadeia de entrada sendo lida de um arquivo. Esse simplesmente contém a cadeia de entrada a ser lida seguido do símbolo “&” indicador do fim do arquivo.

## Caso de teste 1

Autômato de pilha estruturado visto em aula:



Figura: Máquina primária do autômato.



Figura: Máquina da expressão.

Para esse caso de teste, foi usado o seguinte autômato para o analisador léxico:



Figura: Autômato do analisador léxico desse exemplo.

Segue em anexo o conteúdo dos arquivos de entrada usados.

Para a cadeia de entrada usada x=a1-14 + (a-90)/(p-q); nos obtivemos o seguinte resultado:

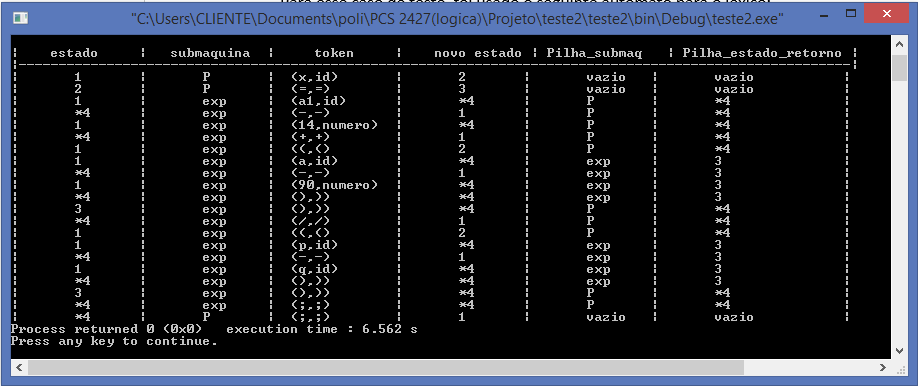


Figura: resultado para cadeia de entrada x=a1-14 + (a-90)/(p-q);

O “\*” antes do nome do estado indica que este é um estado final. Podemos notar, na terceira linha, que o nome a1 é reconhecido corretamente, e não como uma letra (id) e um número (numero) mostrando o correto funcionamento de analisador léxico. Pode-se notar também que o valor impresso nas duas pilhas corresponde ao topo da mesma. Quando há transição para sub-máquina esse valor é atualizado e no final, vê-se que a pilha está vazia como era o esperado. Além disso todos os tokens foram passados corretamente com a correta atualização do estado no autômato de pilha.

Conclui-se assim que para esse caso de teste o programa funcionou como esperado.

Posteriormente nós fizemos uma sequência exaustiva de testes para comprovar o correto funcionamento do programa. Essa se encontra a seguir.

## Sequência de testes para caso 1

Cadeia de entrada: x=x;

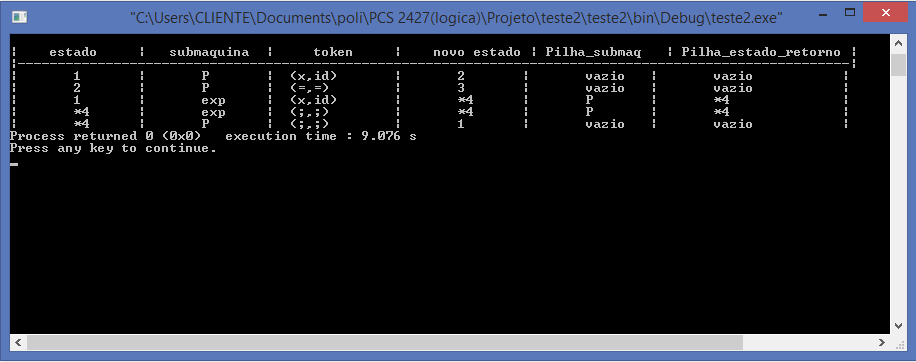


Figura: Teste 1 do autômato de pilha 1

Como pode ser visto, as transições foram feitas como o esperado.

Cadeia de entrada: x=1;

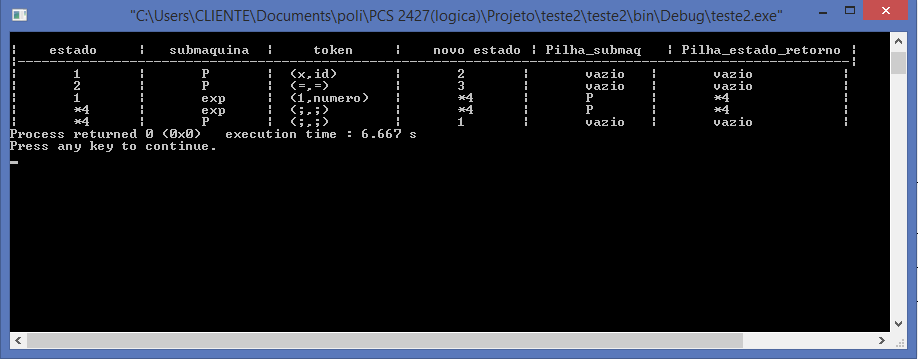


Figura: Teste 2 do autômato de pilha 1

Como pode ser visto, as transições foram feitas como o esperado.

Cadeia de entrada: x=(x);

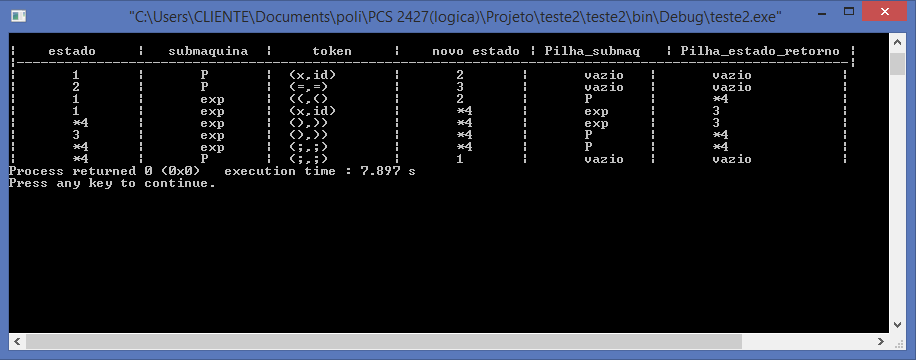


Figura: Teste 3 do autômato de pilha 1

Como pode ser visto, as transições foram feitas como o esperado.

Cadeia de entrada: x=x + (1) ;

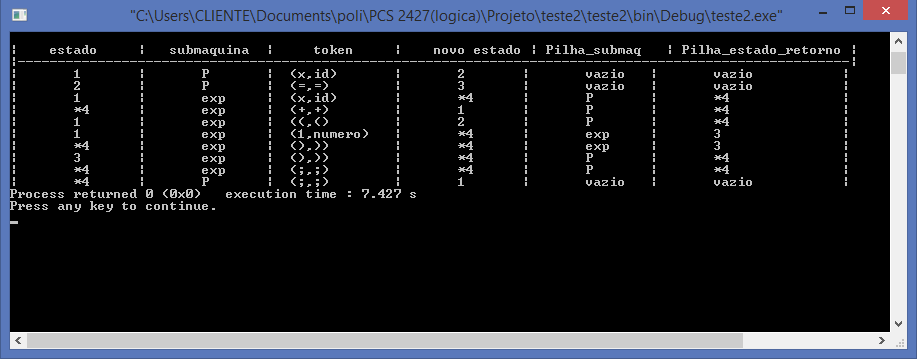


Figura: Teste 4 do autômato de pilha 1

Como pode ser visto, as transições foram feitas como o esperado.

Cadeia de entrada: x=x\*(x+(1));

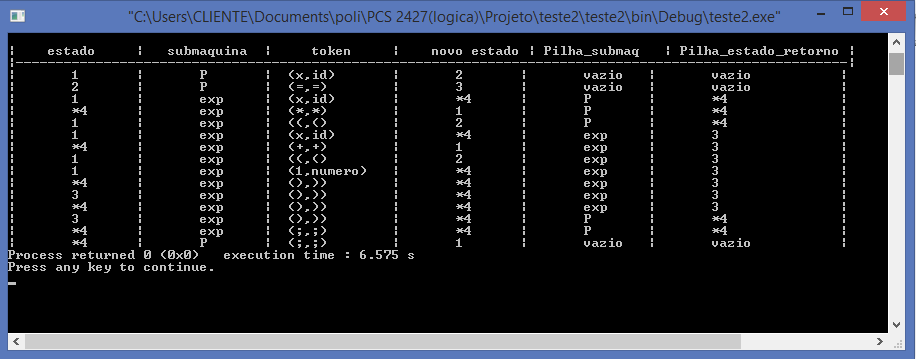


Figura: Teste 5 do autômato de pilha 1

Como pode ser visto, as transições foram feitas como o esperado.

Cadeia de entrada: x=(((x)\*x)-1)+x;

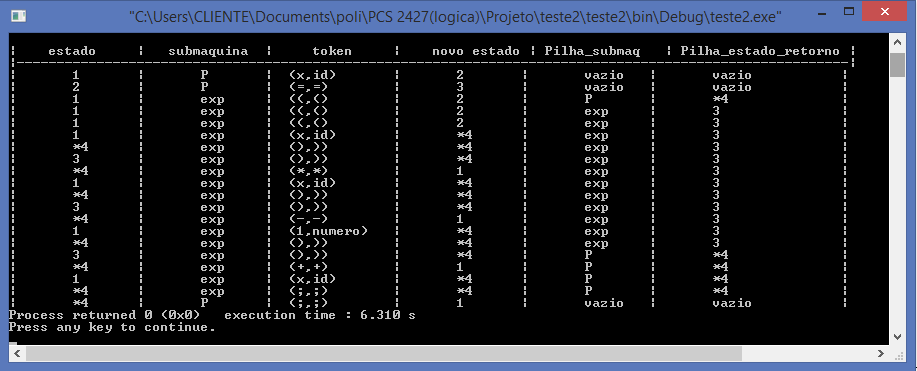


Figura: Teste 6 do autômato de pilha 1

Como pode ser visto, as transições foram feitas como o esperado.

Cadeia de entrada: x=x-(x+(1/x))/x + (1);

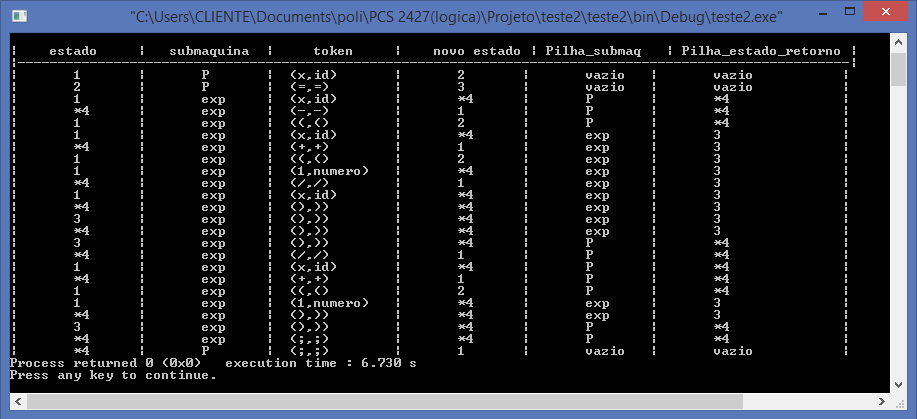


Figura: Teste 7 do autômato de pilha 1

Como pode ser visto, as transições foram feitas como o esperado.

Cadeia de entrada: x= ((x+(n\*1)/1)/(((x))\*1)-1);

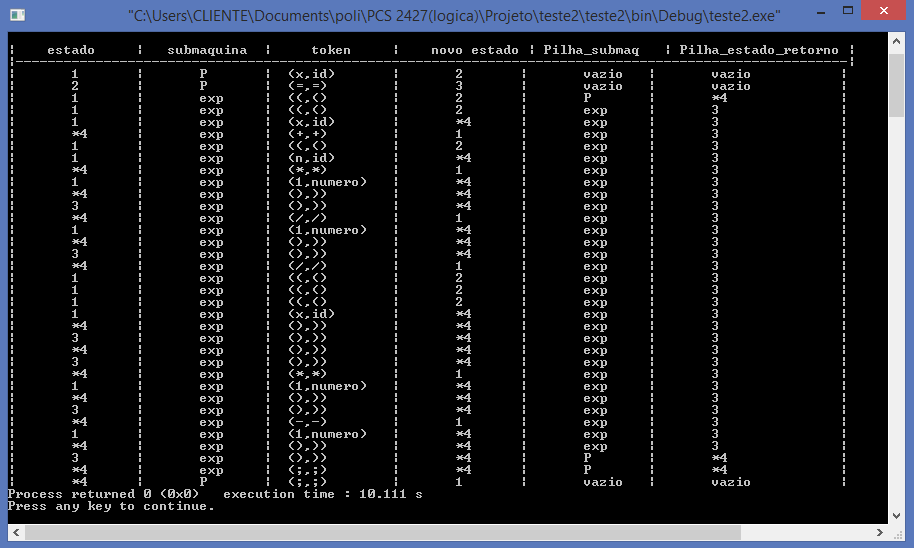


Figura: Teste 8 do autômato de pilha 1

Como pode ser visto, as transições foram feitas como o esperado.

Cadeia de entrada: x = (((x+(x\*1)/1+1)-((x)\*x)-(1))+x);

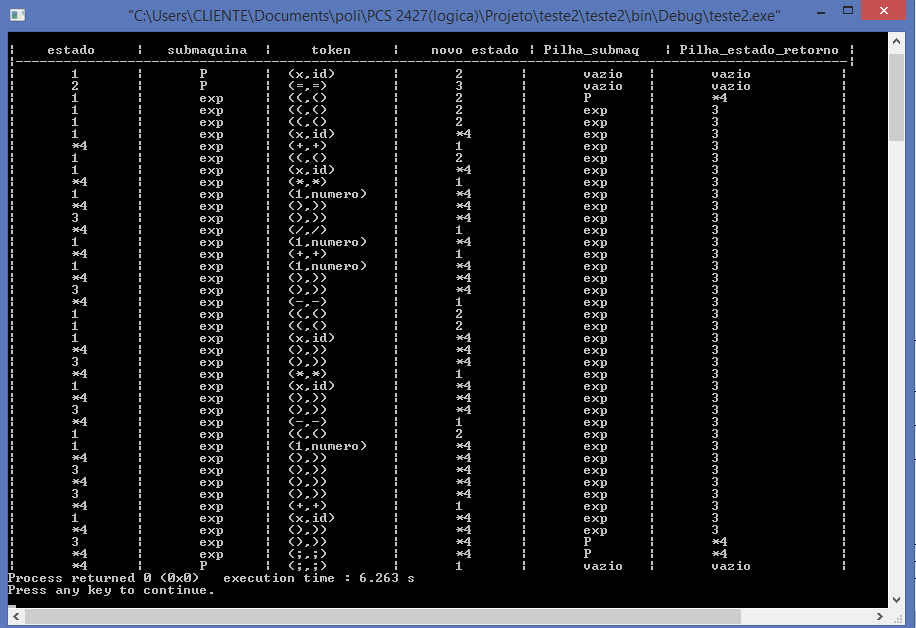


Figura: Teste 9 do autômato de pilha 1

Como pode ser visto, as transições foram feitas como o esperado.

Cadeia de entrada: x=(((x)+(x\*1))/1+1)-(x\*(1)-(x-1\*(1)));

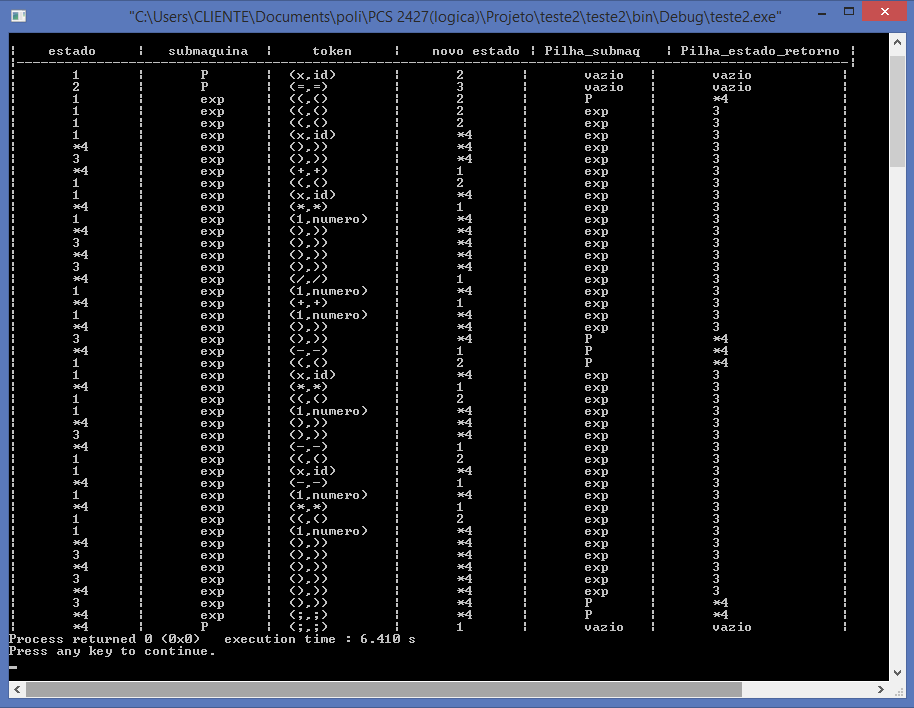


Figura: Teste 10 do autômato de pilha 1

Como pode ser visto, as transições foram feitas como o esperado.

## Caso de teste 2

Após testarmos o autômato visto em sala, nos testamos o autômato que aceita cadeias apenas com letras “a” ou cadeias completas dessa linguagem, aninhadas entre parêntesis, colchetes ou chaves. Nos alteramos o valor de “a” por “id”, ou seja, ao ler o “a” do arquivo o token gerado será “id”. Seu digrama pode ser visto na figura abaixo.

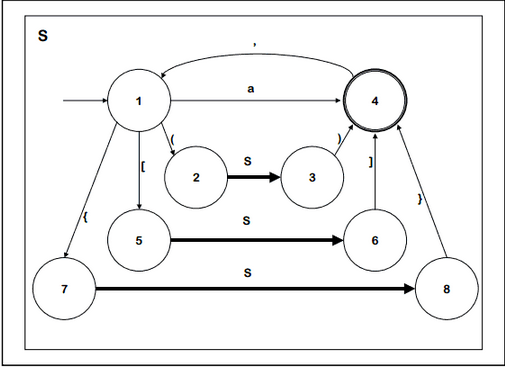


Figura: Autômato de pilha usado para teste.

Nós usamos como autômato do analisador léxico o mesmo do exemplo anterior acrescido das transições de colchetes “[]“, chaves ”{}” e vírgula ”,” como pode ser visto abaixo. Ele é claramente mais abrangente que o necessário já que leva em conta todas as letras e números além dos operandos algébricos e sinal de igual “=”.



Figura: Diagrama do autômato do analisador léxico modificado.

O anexo 2 contém o conteúdo dos arquivos de entrada nesse caso de teste.

Novamente nos fizemos uma sequência de casos de teste, eles se encontram abaixo.

## Sequência de testes para caso 2

Cadeia de entrada: a

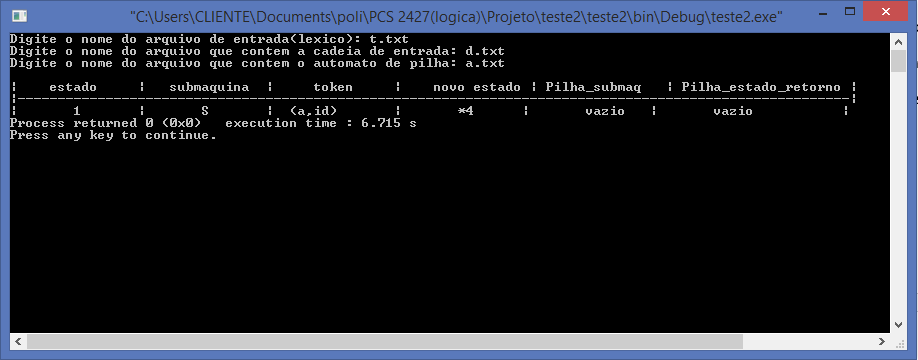


Figura: Teste 1 do autômato de pilha 2

Cadeia de entrada: a,a,a,a

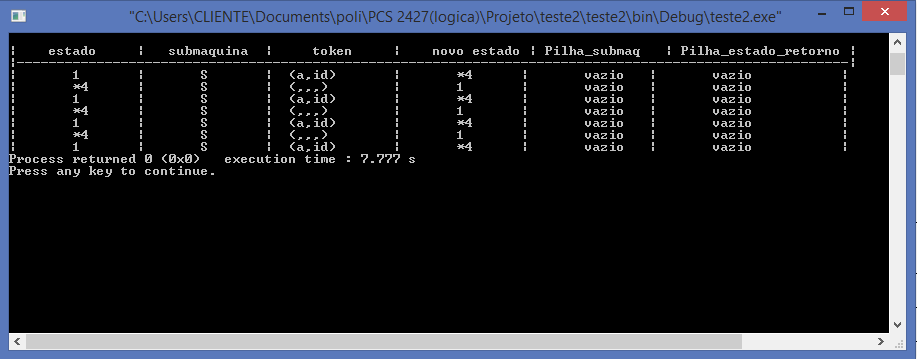


Figura: Teste 2 do autômato de pilha 2

Como esperado, a sub-máquina ”S” não é chamada.

Cadeia de entrada: (a)

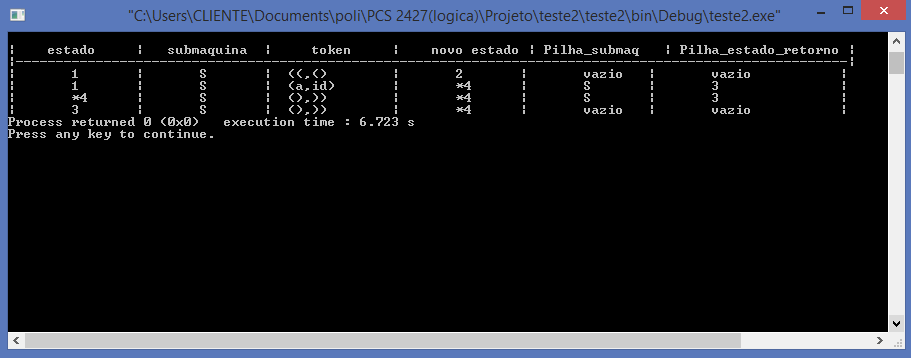


Figura: Teste 3 do autômato de pilha 2

Com o “(“ a sub-maquina “S” é chamada com estado de retorno 3. Resultado também foi perfeito.

Cadeia de entrada: [a,a]

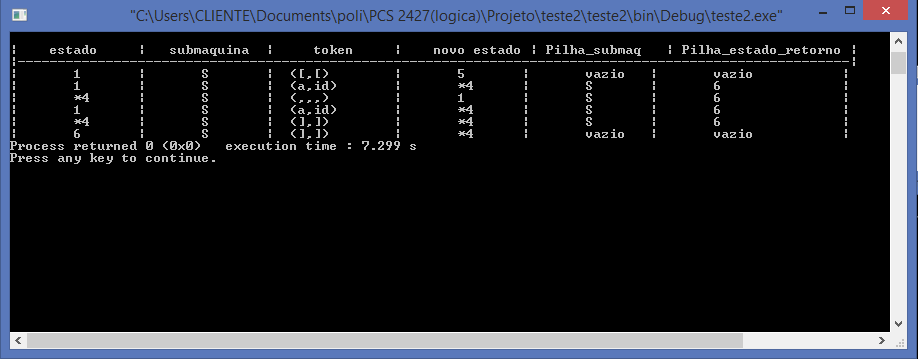


Figura: Teste 4 do autômato de pilha 2

Vemos que o “[“ chama a sub-máquina “S” corretamente com estado de retorno “6”.

Cadeia de entrada: {a}

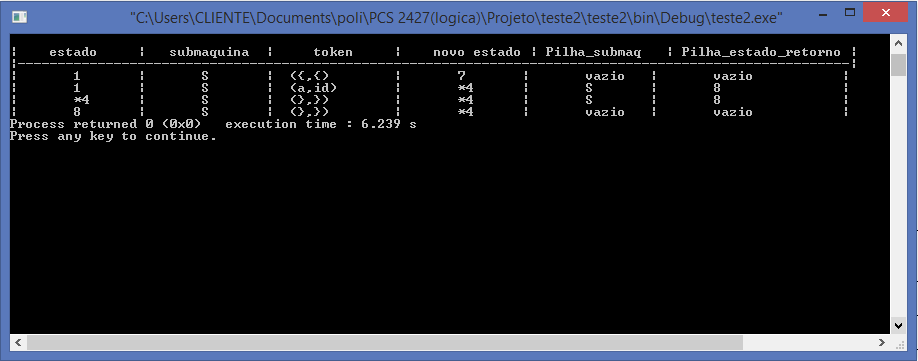


Figura: Teste 5 do autômato de pilha 2

Novamente o resultado é correto e percebe-se que o “{“ chama a sub-máquina “S” com estado de retorno 8.

Cadeia de entrada: (a) ,a,(a),[a],{a},a

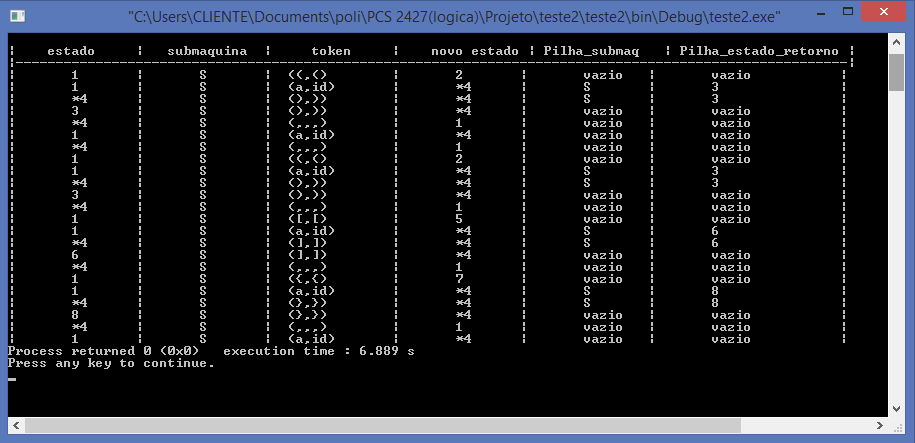


Figura: Teste 6 do autômato de pilha 2

Resumo das transições anteriores, vemos que funciona corretamente com esperado.

Cadeia de entrada: (((a))),[[a]],{{{{a}}}}

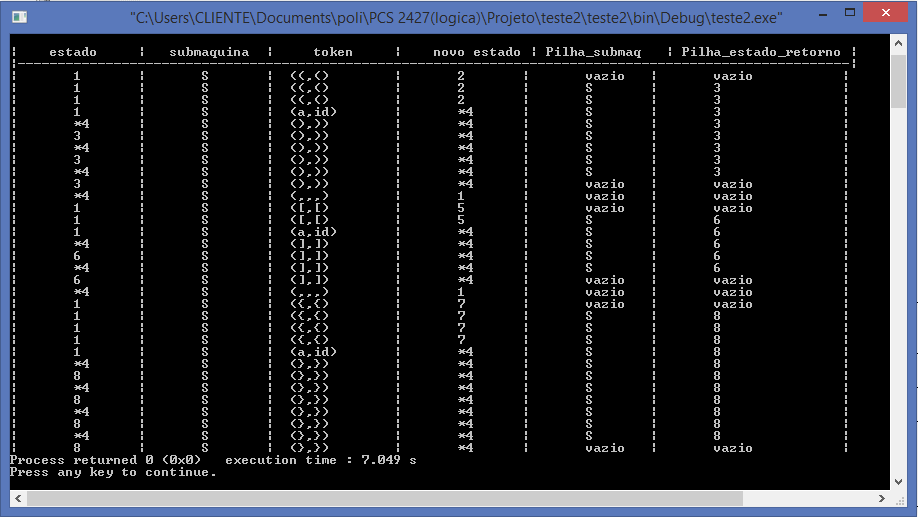


Figura: Teste 7 do autômato de pilha 2

Como vemos, sucessivas transições para sub-máquina também funcionam como esperado.

Cadeia de entrada: {[((a),[a],{a})]}

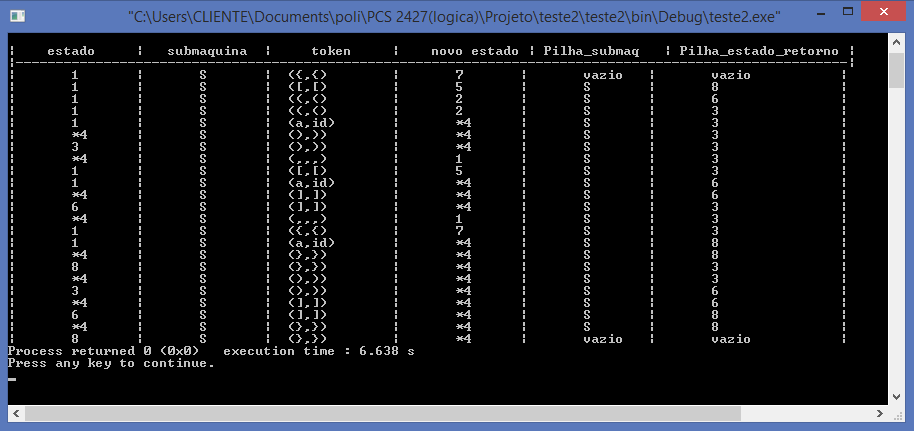


Figura: Teste 8 do autômato de pilha 2

Como vemos, sucessivas transições em estados diferentes para sub-máquina também funcionam como esperado.

Cadeia de entrada: a,([a],{a}),[(a),{{a}}],{[(a),a],a},a

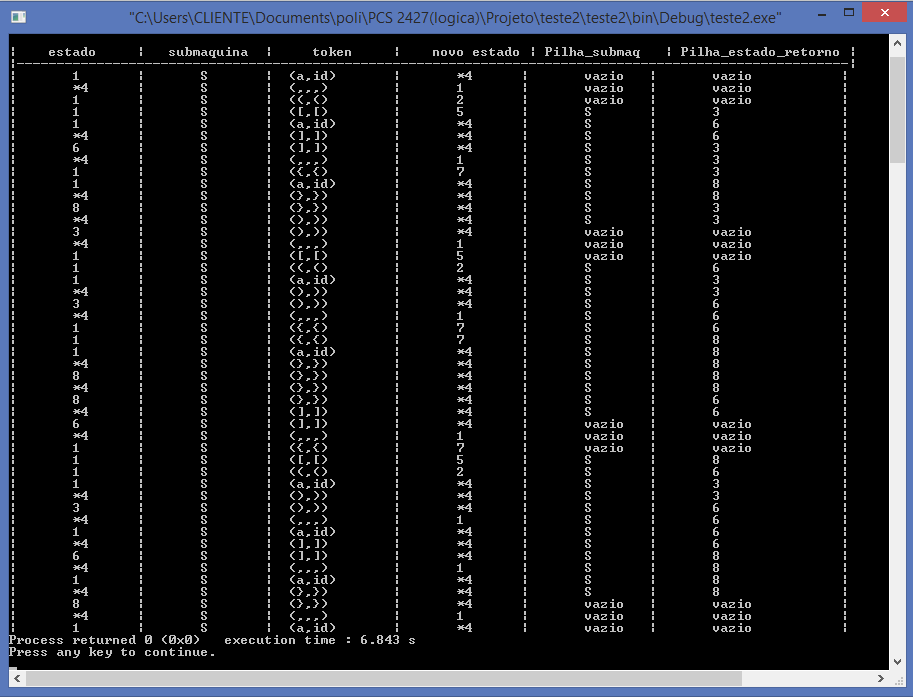


Figura: Teste 9 do autômato de pilha 2

Caso também funcionou como o esperado

Cadeia de entrada: a,{a,[a,(a)]},[{{a}},(a)],({a},[a]),a

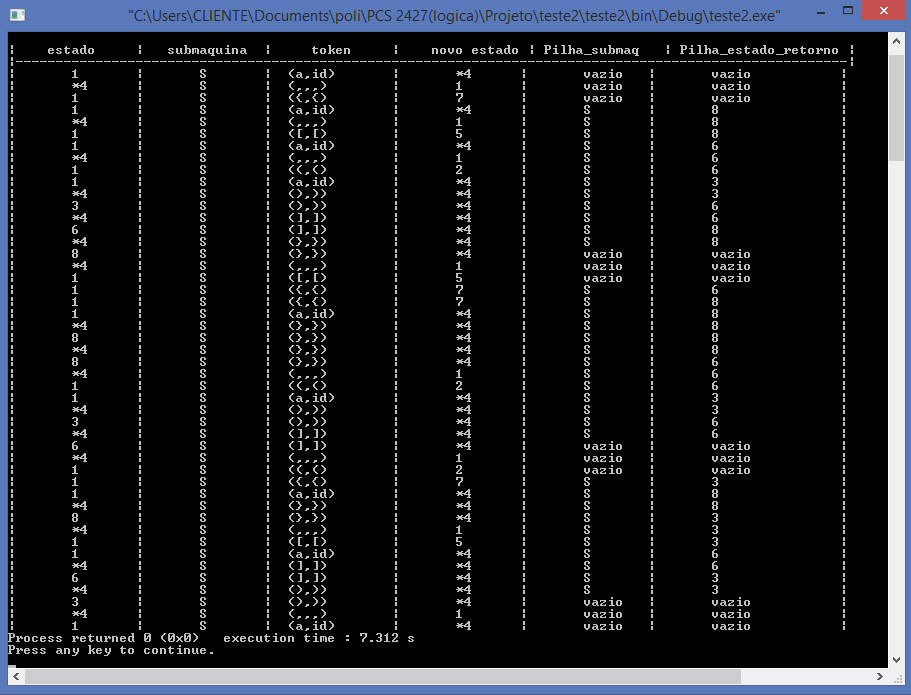


Figura: Teste 10 do autômato de pilha 2

Esse caso de teste também funcionou como deveria.

## Manual

O programa foi feito em C, ao ser executado é pedido que o usuário digite o nome de três arquivos (um de cada vez):

Arquivo que contém o autômato do léxico.

Arquivo que contém a cadeia de entrada (apenas uma cadeia).

Arquivo que contém o autômato de pilha.

Esses arquivos devem estar no diretório do executável. Após pressionar enter depois de inserir o terceiro arquivo a tabela será gerada. É importante que sejam inseridas cadeias válidas e que os arquivos estejam no formato já especificado acima.

## Anexos

1. Caso de teste 1
   1. Conteúdo do arquivo com autômato de pilha:

P exp ;

1

1 id 2

2 = 3

3 exp \*4

\*4 ; 1

!

1

1 id \*4

1 numero \*4

1 ( 2

2 exp 3

3 ) \*4

\*4 + 1

\*4 - 1

\*4 \* 1

\*4 / 1

!

* 1. Conteúdo do arquivo com cadeia de entrada:

x=a1-14 + (a-90)/(p-q); &

1.3) Conteúdo do arquivo com autômato do léxico:

1

1 + \*+

1 - \*-

1 \* \*\*

1 / \*/

1 ( \*(

1 ) \*)

1 = \*=

1 ; \*;

1 a \*id

1 b \*id

1 c \*id

1 d \*id

1 e \*id

1 f \*id

1 g \*id

1 h \*id

1 i \*id

1 j \*id

1 k \*id

1 l \*id

1 m \*id

1 n \*id

1 o \*id

1 p \*id

1 q \*id

1 r \*id

1 s \*id

1 t \*id

1 u \*id

1 v \*id

1 x \*id

1 z \*id

\*id a \*id

\*id b \*id

\*id c \*id

\*id d \*id

\*id e \*id

\*id f \*id

\*id g \*id

\*id h \*id

\*id i \*id

\*id j \*id

\*id k \*id

\*id l \*id

\*id m \*id

\*id n \*id

\*id o \*id

\*id p \*id

\*id q \*id

\*id r \*id

\*id s \*id

\*id t \*id

\*id u \*id

\*id v \*id

\*id x \*id

\*id z \*id

\*id 0 \*id

\*id 1 \*id

\*id 2 \*id

\*id 3 \*id

\*id 4 \*id

\*id 5 \*id

\*id 6 \*id

\*id 7 \*id

\*id 8 \*id

\*id 9 \*id

1 1 \*numero

1 2 \*numero

1 3 \*numero

1 4 \*numero

1 5 \*numero

1 6 \*numero

1 7 \*numero

1 8 \*numero

1 9 \*numero

1 0 \*numero

\*numero 0 \*numero

\*numero 1 \*numero

\*numero 2 \*numero

\*numero 3 \*numero

\*numero 4 \*numero

\*numero 5 \*numero

\*numero 6 \*numero

\*numero 7 \*numero

\*numero 8 \*numero

\*numero 9 \*numero

!

1. Caso de teste 2

2.1) Conteúdo do arquivo com autômato de pilha:

S ;

1

1 id \*4

1 ( 2

1 [ 5

1 { 7

2 S 3

5 S 6

7 S 8

3 ) \*4

6 ] \*4

8 } \*4

\*4 , 1

!

2.2) Conteúdo do arquivo com cadeia de entrada:

a,{a,[a,(a)]},[{{a}},(a)],({a},[a]),a &

2.3) Conteúdo do arquivo com autômato do lexico:

1

1 + \*+

1 - \*-

1 \* \*\*

1 / \*/

1 ( \*(

1 ) \*)

1 = \*=

1 ; \*;

1 , \*,

1 [ \*[

1 ] \*]

1 { \*{

1 } \*}

1 a \*id

1 b \*id

1 c \*id

1 d \*id

1 e \*id

1 f \*id

1 g \*id

1 h \*id

1 i \*id

1 j \*id

1 k \*id

1 l \*id

1 m \*id

1 n \*id

1 o \*id

1 p \*id

1 q \*id

1 r \*id

1 s \*id

1 t \*id

1 u \*id

1 v \*id

1 x \*id

1 z \*id

\*id a \*id

\*id b \*id

\*id c \*id

\*id d \*id

\*id e \*id

\*id f \*id

\*id g \*id

\*id h \*id

\*id i \*id

\*id j \*id

\*id k \*id

\*id l \*id

\*id m \*id

\*id n \*id

\*id o \*id

\*id p \*id

\*id q \*id

\*id r \*id

\*id s \*id

\*id t \*id

\*id u \*id

\*id v \*id

\*id x \*id

\*id z \*id

\*id 0 \*id

\*id 1 \*id

\*id 2 \*id

\*id 3 \*id

\*id 4 \*id

\*id 5 \*id

\*id 6 \*id

\*id 7 \*id

\*id 8 \*id

\*id 9 \*id

1 1 \*numero

1 2 \*numero

1 3 \*numero

1 4 \*numero

1 5 \*numero

1 6 \*numero

1 7 \*numero

1 8 \*numero

1 9 \*numero

1 0 \*numero

\*numero 0 \*numero

\*numero 1 \*numero

\*numero 2 \*numero

\*numero 3 \*numero

\*numero 4 \*numero

\*numero 5 \*numero

\*numero 6 \*numero

\*numero 7 \*numero

\*numero 8 \*numero

\*numero 9 \*numero

!