PCS-2056
Linguagens e Compiladores

Analisador Sintático

Renan Martins Zomignani Mendes
Tiago Schelp Lopes

Introdução

O analisador sintático é o componente central do compilador. A partir dele aciona-se tanto as funcionalidades do analisador léxico (busca do próximo token) quanto do analisador semântico (verificações referentes aos escopos, tradução para código de máquina, etc).

A função principal do analisador sintático é, no entanto, a validação da cadeia de tokens como pertencente à linguagem, além de apontar a localização dos erros de sintaxe.

Lista de submáquinas do APE

As submáquinas geradas na página da internet indicada no enunciado são as seguintes (algumas abreviações para as mais simples):

LETRA

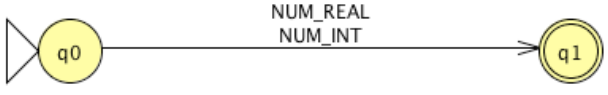
```
initial: 0
final: 1
(0, "A") -> 1
(0, "B") -> 1
  ...
(0, "Z") -> 1
(0, "a") -> 1
(0, "b") -> 1
  ...
(0, "z") -> 1
```

DIGITO

```
initial: 0
final: 1
(0, "0") -> 1
(0, "1") -> 1
  ...
(0, "9") -> 1
```

NUM

```
initial: 0
final: 1
(0, NUM_INT) -> 1
(0, NUM_REAL) -> 1
```

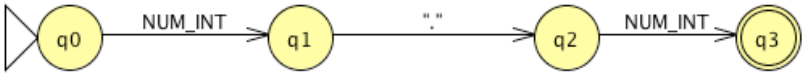


NUM_INT

initial: 0
final: 1
(0, DIGITO) -> 1
(1, DIGITO) -> 1

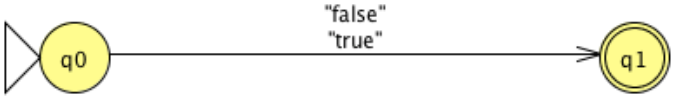
NUM_REAL

initial: 0
final: 3
(0, NUM_INT) -> 1
(1, ".") -> 2
(2, NUM_INT) -> 3



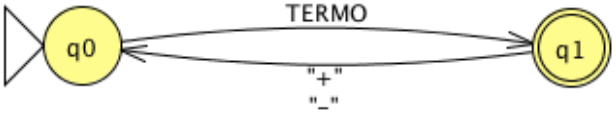
BOOLEAN

initial: 0
final: 1
(0, "true") -> 1
(0, "false") -> 1



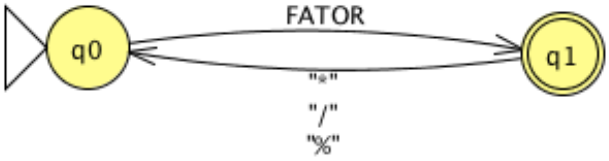
EXPR

initial: 0
final: 1
(0, TERMO) -> 1
(1, "+") -> 0
(1, "-") -> 0



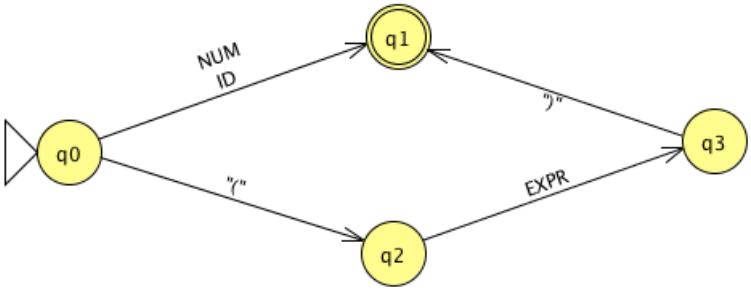
TERMO

initial: 0
final: 1
(0, FATOR) -> 1
(1, "*") -> 0
(1, "/") -> 0
(1, "%") -> 0



FATOR

initial: 0
final: 1
(0, ID) -> 1
(0, NUM) -> 1
(0, "(") -> 2
(2, EXPR) -> 3
(3, ")") -> 1

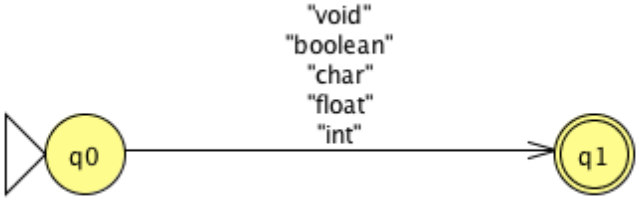


ID

```
initial: 0
final: 1
(0, LETRA) -> 1
(1, LETRA) -> 1
(1, DIGITO) -> 1
```

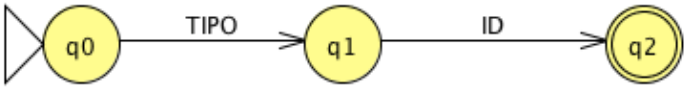
TIPO

```
initial: 0
final: 1
(0, "int") -> 1
(0, "float") -> 1
(0, "char") -> 1
(0, "boolean") -> 1
(0, "void") -> 1
```



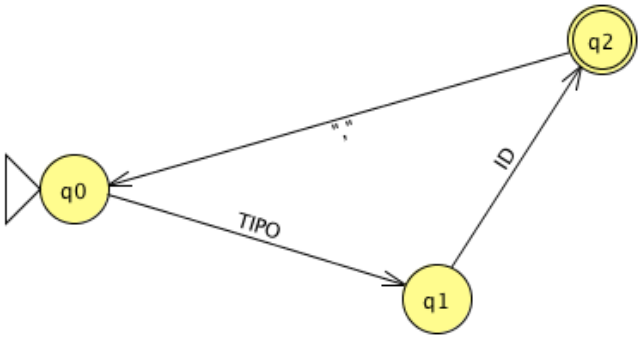
DECL_SIMP_VAR

```
initial: 0
final: 2
(0, TIPO) -> 1
(1, ID) -> 2
```



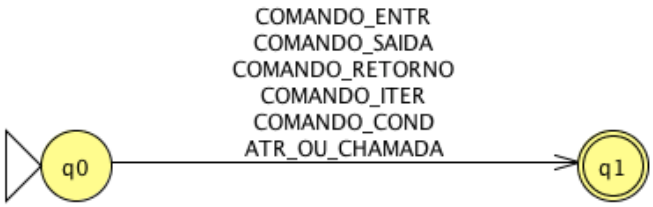
DECL_PARAMS

```
initial: 0
final: 2
(0, TIPO) -> 1
(1, ID) -> 2
(2, ",") -> 0
```



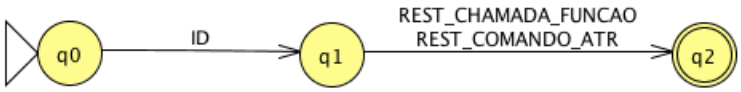
COMANDO

```
initial: 0
final: 1
(0, ATR_OU_CHAMADA) -> 1
(0, COMANDO_COND) -> 1
(0, COMANDO_ITER) -> 1
(0, COMANDO_RETORNO) -> 1
(0, COMANDO_SAIDA) -> 1
(0, COMANDO_ENTR) -> 1
```



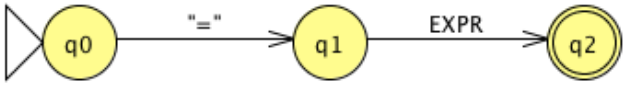
ATR_OU_CHAMADA

```
initial: 0
final: 2
(0, ID) -> 1
(1, REST_COMANDO_ATR) -> 2
(1, REST_CHAMADA_FUNCAO) -> 2
```



REST_COMANDO_ATR

```
initial: 0
final: 2
(0, "=") -> 1
(1, EXPR) -> 2
```

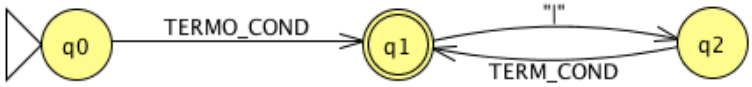


REST_CHAMADA_FUNCAO

```
initial: 0
final: 3
(0, "(") -> 1
(1, PARAM) -> 2
(2, ",") -> 1
(2, ")") -> 3
```

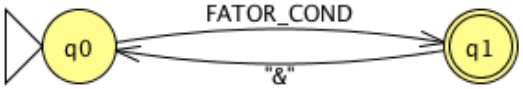
CONDICAO

```
initial: 0
final: 1
(0, TERMO_COND) -> 1
(1, "|") -> 2
(2, TERM_COND) -> 1
```



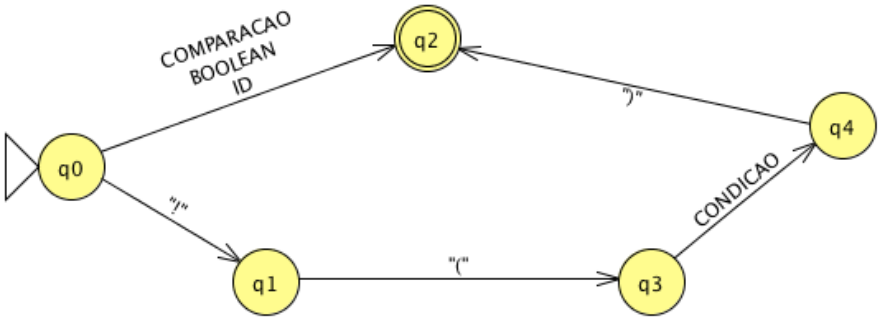
TERMO_COND

```
initial: 0
final: 1
(0, FATOR_COND) -> 1
(1, "&") -> 0
```



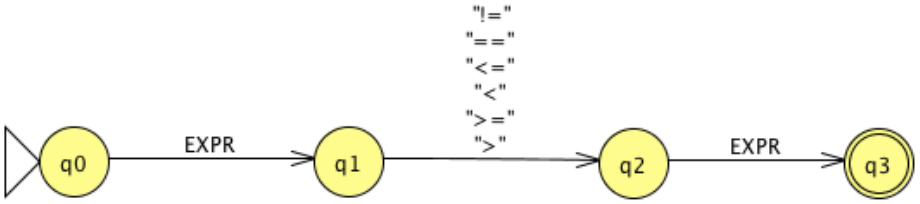
FATOR_COND

```
initial: 0
final: 2
(0, "!") -> 1
(0, ID) -> 2
(0, BOOLEAN) -> 2
(0, COMPARACAO) -> 2
(1, "(") -> 3
(3, CONDICAO) -> 4
(4, ")") -> 2
```



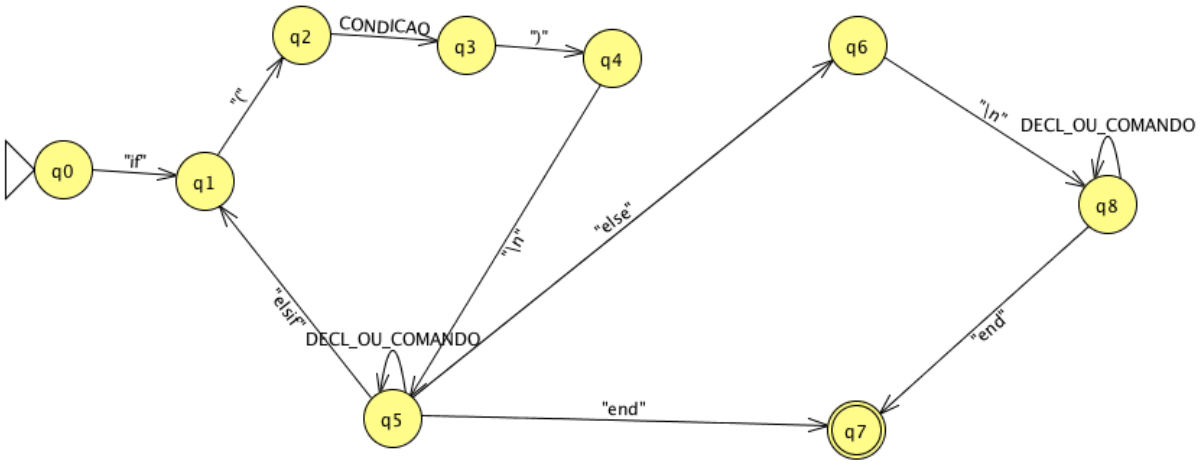
COMPARACAO

```
initial: 0
final: 3
(0, EXPR) -> 1
(1, ">") -> 2
(1, ">=") -> 2
(1, "<") -> 2
(1, "<=") -> 2
(1, "==") -> 2
(1, "!=") -> 2
(2, EXPR) -> 3
```



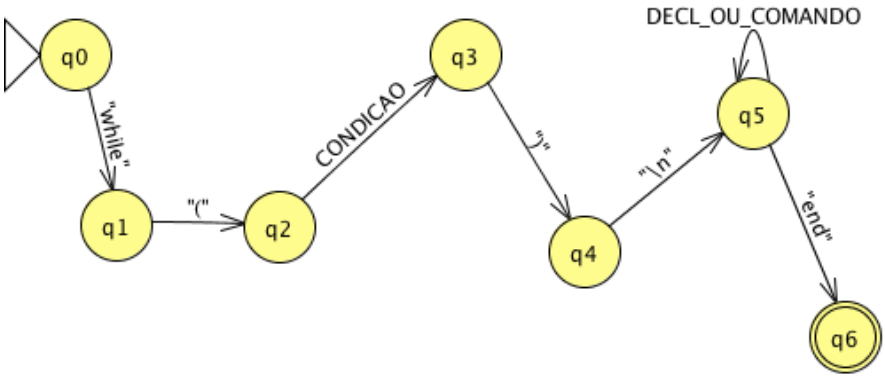
COMANDO_COND

```
initial: 0
final: 7
(0, "if") -> 1
(1, "(") -> 2
(2, CONDICAO) -> 3
(3, ")") -> 4
(4, "\n") -> 5
(5, DECL_OU_COMANDO) -> 5
(5, "elsif") -> 1
(5, "else") -> 6
(5, "end") -> 7
(6, "\n") -> 8
(8, DECL_OU_COMANDO) -> 8
(8, "end") -> 7
```



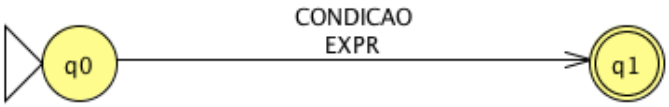
COMANDO_ITER

```
initial: 0
final: 6
(0, "while") -> 1
(1, "(") -> 2
(2, CONDICAO) -> 3
(3, ")") -> 4
(4, "\n") -> 5
(5, DECL_OU_COMANDO) -> 5
(5, "end") -> 6
```



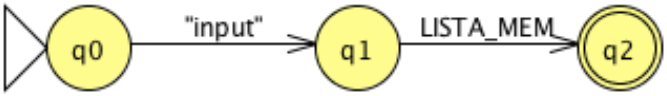
PARAM

```
initial: 0
final: 1
(0, EXPR) -> 1
(0, CONDICAO) -> 1
```



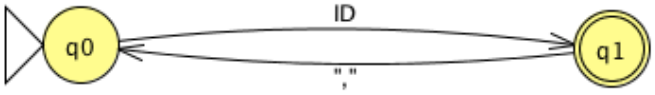
COMANDO_ENTR

```
initial: 0
final: 2
(0, "input") -> 1
(1, LISTA_MEM) -> 2
```



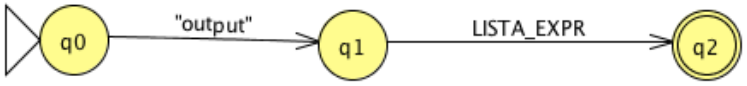
LISTA_MEM

```
initial: 0
final: 1
(0, ID) -> 1
(1, ",") -> 0
```



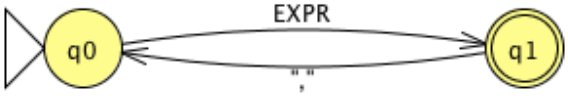
COMANDO_SAIDA

```
initial: 0
final: 2
(0, "output") -> 1
(1, LISTA_EXPR) -> 2
```



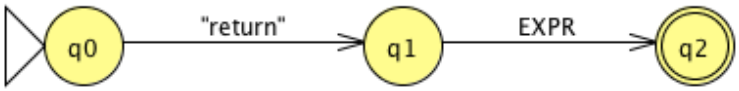
LISTA_EXPR

```
initial: 0
final: 1
(0, EXPR) -> 1
(1, ",") -> 0
```



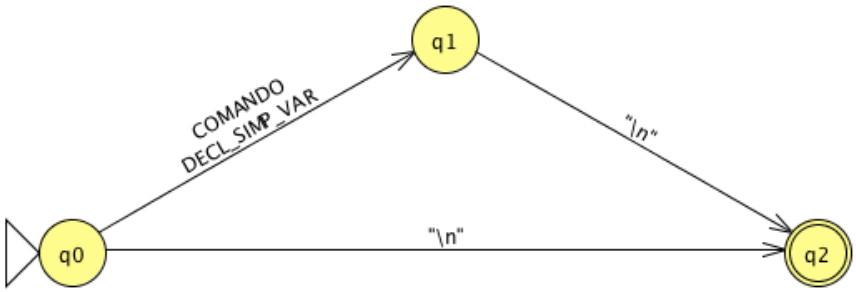
COMANDO_RETORNO

initial: 0
final: 2
(0, "return") -> 1
(1, EXPR) -> 2



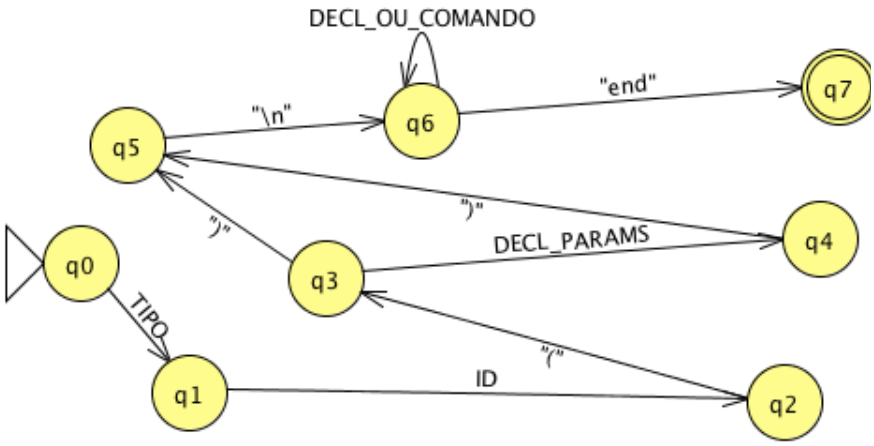
DECL_OU_COMANDO

initial: 0
final: 2
(0, DECL_SIMP_VAR) -> 1
(0, COMANDO) -> 1
(0, "\n") -> 2
(1, "\n") -> 2



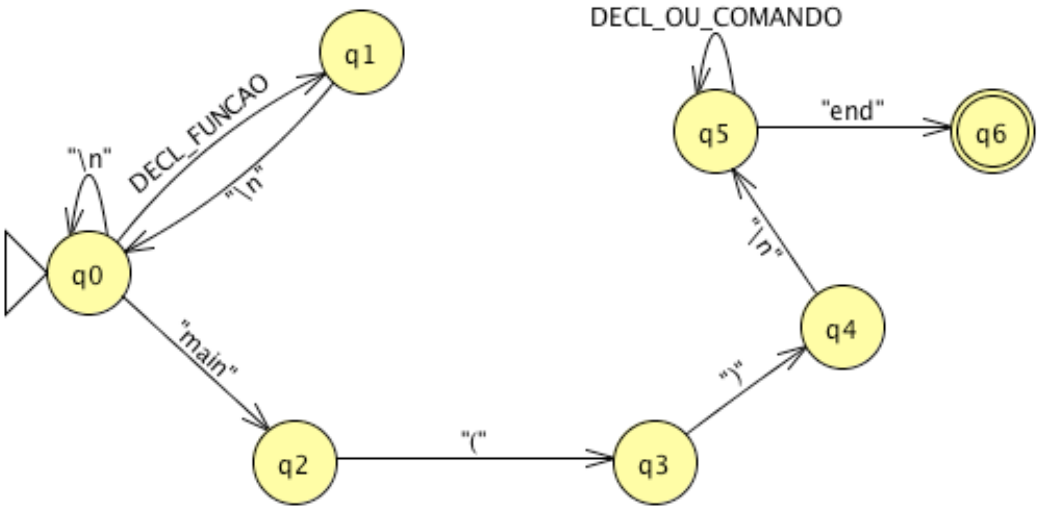
DECL_FUNCAO

initial: 0
final: 7
(0, TIPO) -> 1
(1, ID) -> 2
(2, "(") -> 3
(3, DECL_PARAMS) -> 4
(3, ")") -> 5
(4, ")") -> 5
(5, "\n") -> 6
(6, DECL_OU_COMANDO) -> 6
(6, "end") -> 7



PROGRAM

initial: 0
final: 6
(0, DECL_FUNCAO) -> 1
(0, "\n") -> 0
(0, "main") -> 2
(1, "\n") -> 0
(2, "(") -> 3
(3, ")") -> 4
(4, "\n") -> 5
(5, DECL_OU_COMANDO) -> 5
(5, "end") -> 6



Comentários sobre a implementação

A implementação do analisador sintático foi realizada segundo o método do Autômato de Pilha Estruturado, em que os diferentes componentes da gramática são reconhecidos por meio de submáquinas.

Para tal, foi necessário primeiramente identificar todas as transições possíveis, tanto as transições internas quanto as chamadas de submáquina. As primeiras foram guardadas na tabela `transicoes` e as últimas na tabela `chamadas`.

Dado um token e um estado, o analisador sintático verifica se existe uma transição interna adequada. Se existir, consome-se o token, ou seja, analisa-se o próximo estado com o próximo token. Caso não exista transição interna, procura-se uma chamada de submáquina apropriada. Se encontrada, empilha-se o estado de retorno e realiza a transição. Nesse caso o token não se consome.

Existem, no entanto, situações em que não haverá nem transição interna nem chamada de submáquina adequadas. Nesse caso, verifica-se se o estado corrente corresponde a um estado final de alguma submáquina utilizando-se a função `estadoFinal()`. Em caso afirmativo, retorna-se ao estado desempilhado do topo da pilha.

Em último caso, indica-se um erro de sintaxe na linha do token atual.

Algumas modificações tiveram que ser feitas na gramática exibida na entrega parcial precedente a esta para que fosse possível a implementação do analisador sintático utilizando o método do APE. Essa gramática encontra-se explicitada no arquivo `WIRTH.txt`.