3ª parte do projeto - construção de um reconhecedor sintático

PCS2056 - Linguagens e Compiladores

Prof. Ricardo Rocha

Felipe de Paiva Miranda 7630486
Thiago Ryu Niwa Murakami 7626689

1. Introdução

Após a etapa de análise léxica do compilador, o analisador sintático verifica a validade da sequência, isto é, se os átomos estão agrupados de forma sintaticamente corretas, de acordo com as regras especificadas. Essa atividade denomina-se reconhecimento sintático, e além dessa função de reconhecer erros de sintaxe, o analisador deve ter mecanismos de recuperação de erros sintáticos para que possa prosseguir a análise do programa mesmo na detecção de erros.

A função mais importante do analisador sintático, no entanto, é, a partir dos átomos, levantar a estrutura sintática do texto-fonte. Ou seja, a partir da sequência de *tokens* gerados pela análise léxica, construir a árvore sintática do texto-fonte, relacionando as construções às gramáticas que definem a linguagem. A árvore gerada pode ser parte da árvore completa do código-fonte e pode apresentar modificações, como eliminação de redundâncias e elementos supérfluos, tal árvore é denominada árvore sintática abstrata.

Logo, um analisador sintático deve verificar se a sintaxe está correta, recuperar-se de erros sintáticos e continuar o processo, e gerar uma árvore sintática do texto-fonte.

2. Lista de submáquinas do Autômatos de Pilha Estruturados

2.1. Lista de transições

Foram encontradas 12 submáquinas em nossa linguagem que são:

1. **PROGRAMA**
   initial: 0
   final: 2
   (0, "program") -> 1
   (1, **ESCOPO**) -> 2

2. **DEFINICAO**
   initial: 0
   final: 7
   (0, "deftype") -> 1
   (1, IDENTIFICADOR) -> 2
   (2, "{") -> 3
   (3, TIPO) -> 4
   (4, IDENTIFICADOR) -> 5
   (5, ",") -> 3
   (5, "}") -> 6
   (6, ";") -> 7

3. **FUNCAO**
   initial: 0
   final: 7
   (0, "function") -> 1
   (1, IDENTIFICADOR) -> 2
   (2, "(") -> 3
   (3, TIPO) -> 4
   (4, IDENTIFICADOR) -> 5
   (5, ",") -> 3
   (5, ")") -> 6
   (6, **ESCOPO**) -> 7
   (6, ":") -> 8
   (8, TIPO) -> 9
   (9, **ESCOPO**) -> 7

4. **ESCOPO**
   initial: 0
   final: 2
   (0, "{") -> 1
   (1, **COMANDO**) -> 1
   (1, "}") -> 2

5. **VARIAVEL**
   initial: 0
   final: 5
   (0, TIPO) -> 1
   (1, IDENTIFICADOR) -> 2
   (1, "[") -> 3
   (2, "=") -> 4
   (2, ";") -> 5
   (2, ",") -> 6
   (3, "]") -> 7
   (3, INTEIRO) -> 8
   (4, **EXPRESSAO**) -> 10
   (6, IDENTIFICADOR) -> 9
   (7, IDENTIFICADOR) -> 11
   (8, "]") -> 6
   (9, ";") -> 5
   (9, ",") -> 6
   (10, ";") -> 5
   (11, "=") -> 12
   (12, "{") -> 13
   (13, **EXPRESSAO**) -> 14
   (14, ",") -> 13
   (14, "}") -> 10

6. **CONDICIONAL**
   initial: 0
   final: 7
   (0, "(") -> 1
   (0, "if") -> 2
   (1, **EXPRESSAO**) -> 3
   (2, "(") -> 4
   (3, ")") -> 5
   (4, **EXPRESSAO**) -> 6
   (5, **ESCOPO**) -> 7
   (6, ")") -> 8
   (8, **ESCOPO**) -> 9
   (9, "elsif") -> 2
   (9, "else") -> 5

7. **REPETICAO**
   initial: 0
   final: 4
   (0, "(") -> 1
   (1, **EXPRESSAO**) -> 2
   (2, ")") -> 3
   (3, **ESCOPO**) -> 4

8. **COMANDO**
   initial: 0
   final: 7
   (0, IDENTIFICADOR) -> 1
   (0, "return") -> 2
   (0, DEF_VARIAVEL) -> 3
   (0, "if") -> 4
   (0, "while") -> 5
   (1, "=") -> 2
   (1, "(") -> 6
   (2, **EXPRESSAO**) -> 9
   (3, **VARIAVEL**) -> 7
   (4, **CONDICIONAL**) -> 7
   (5, **REPETICAO**) -> 7
   (6, **EXPRESSAO**) -> 8
   (6, ")") -> 9
   (8, ",") -> 10
   (8, ")") -> 9
   (9, ";") -> 7
   (10, **EXPRESSAO**) -> 8

9. **EXPRESSAO**
   initial: 0
   final: 1, 3
   (0, **TERMODECIMAL**) -> 1
   (1, OPERADOR) -> 2
   (2, **EXPRESSAO**) -> 3

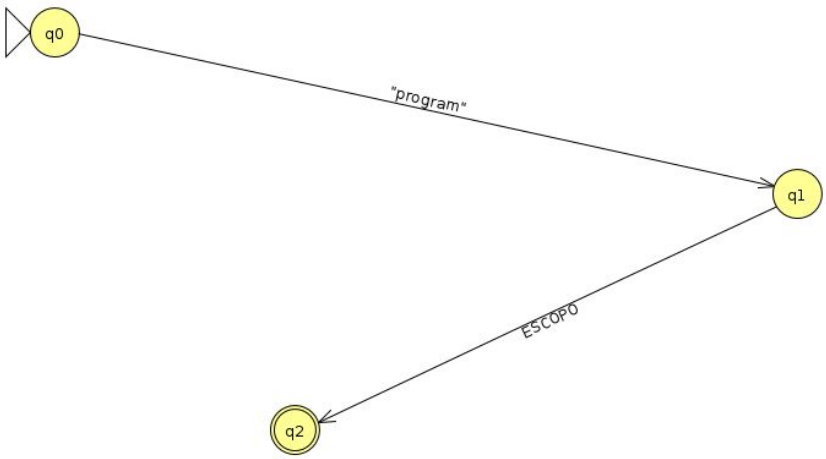
10. **TERMODECIMAL**
   initial: 0
   final: 1, 3
   (0, **TERMOCOMP**) -> 1
   (1, OPERADORBOOL) -> 2
   (2, **TERMODECIMAL**) -> 3

11. **TERMOCOMP**
   initial: 0
   final: 1, 3
   (0, **TERMOPRIMARIO**) -> 1
   (1, OPERADORCOMP) -> 2
   (2, **TERMOCOMP**) -> 3

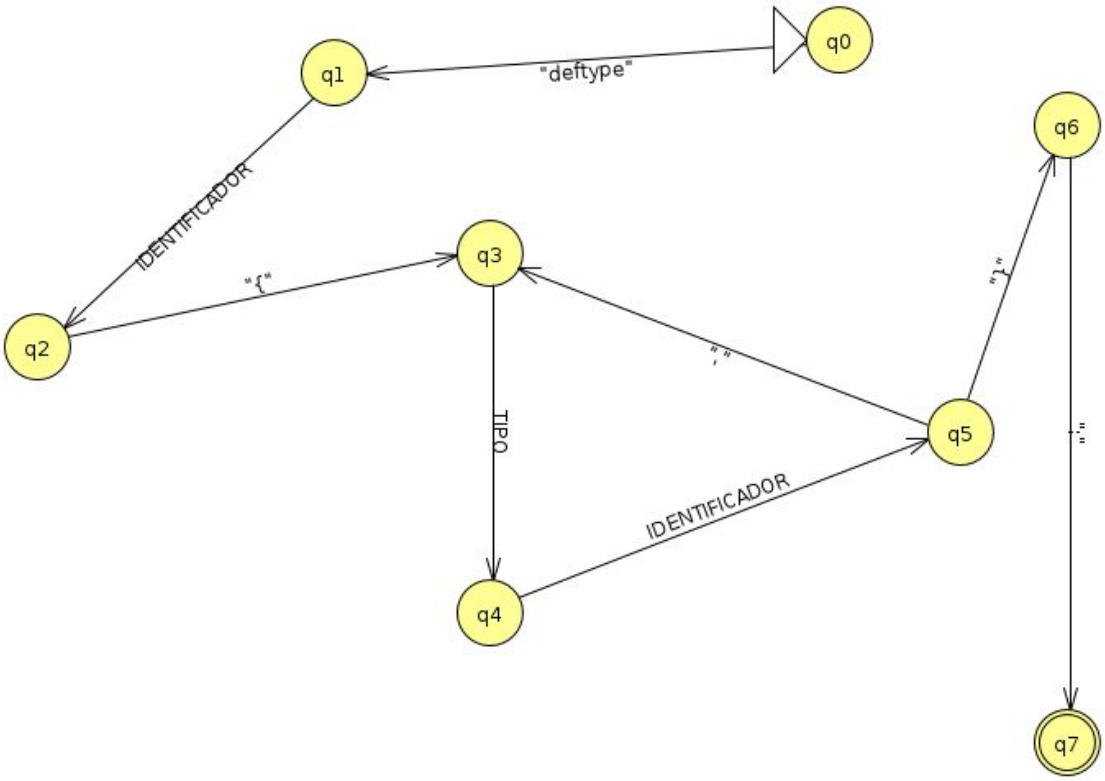
12. **TERMOPRIMARIO**
   initial: 0
   final: 1, 3
   (0, IDENTIFICADOR) -> 1
   (0, "(") -> 2
   (0, DECIMAL) -> 3
   (0, FLOAT) -> 3
   (0, BOOL) -> 3
   (0, STRING) -> 3
   (1, "(") -> 4
   (2, **EXPRESSAO**) -> 7
   (4, **EXPRESSAO**) -> 5
   (4, ")") -> 3
   (5, ",") -> 6
   (5, ")") -> 3
   (6, **EXPRESSAO**) -> 5
   (7, ")") -> 3

2.2. Lista de autômatos

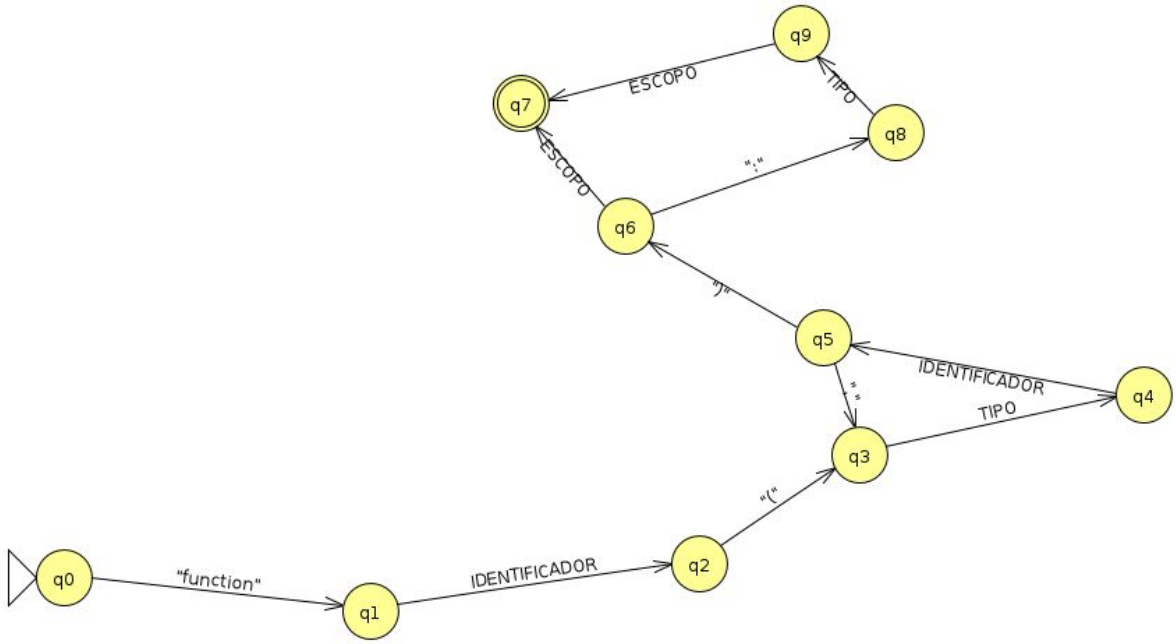
1. PROGRAMA



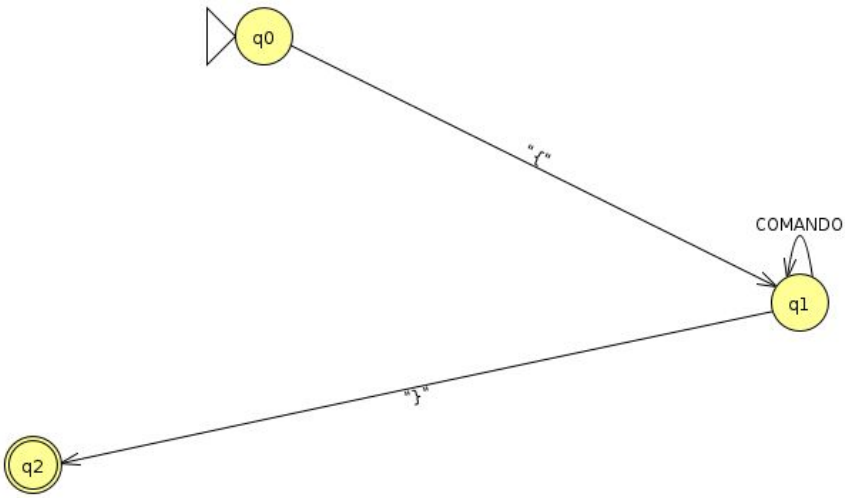
2. DEFINICAO



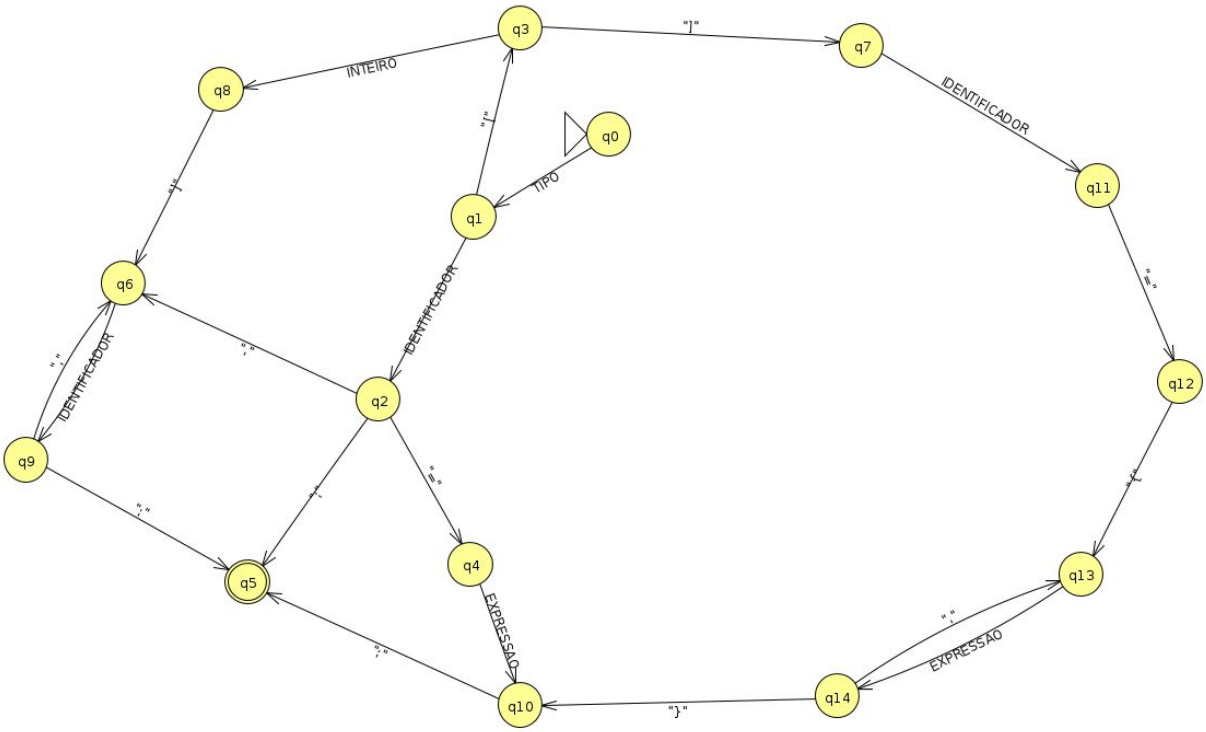
3. FUNCAO



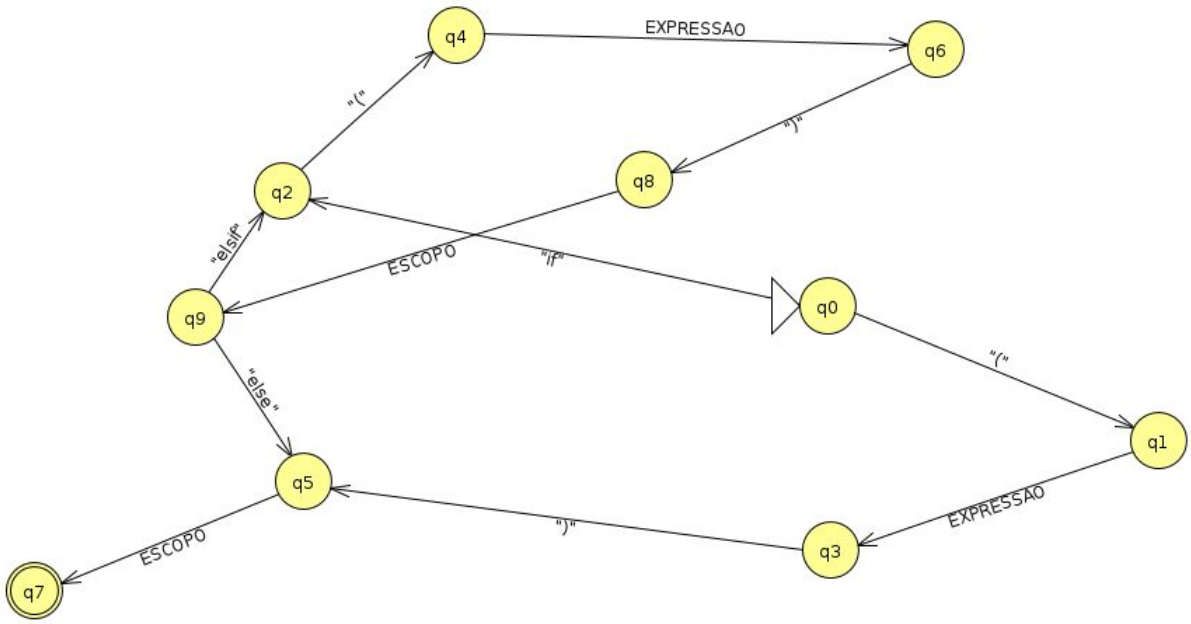
4. ESCOPO



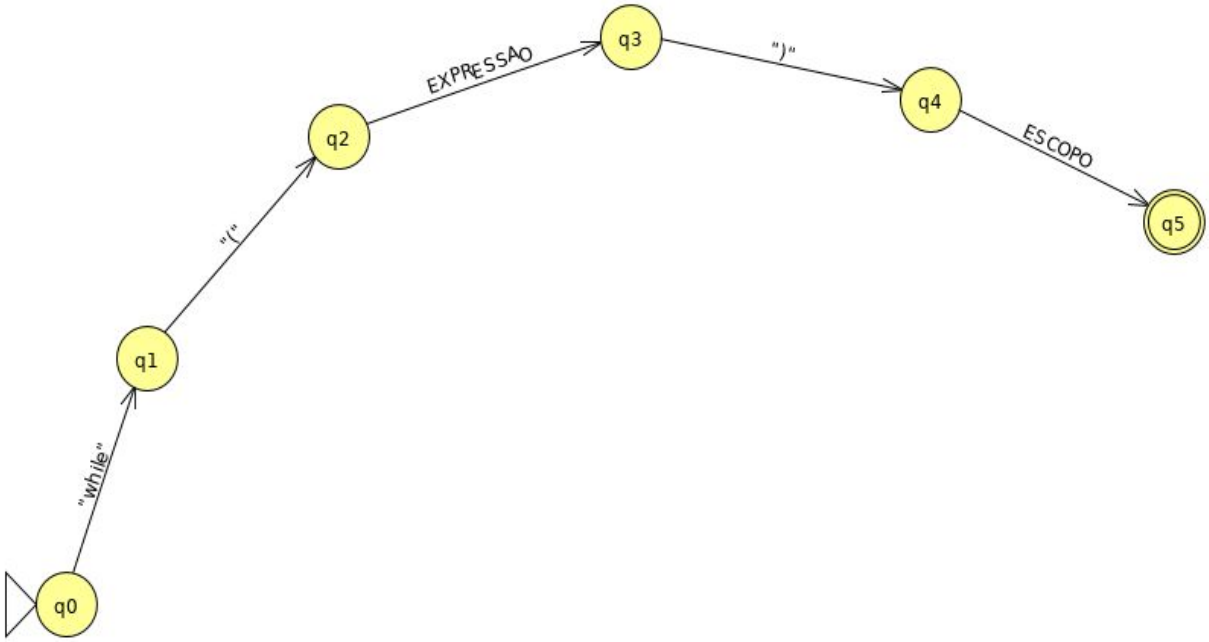
5. VARIAVEL



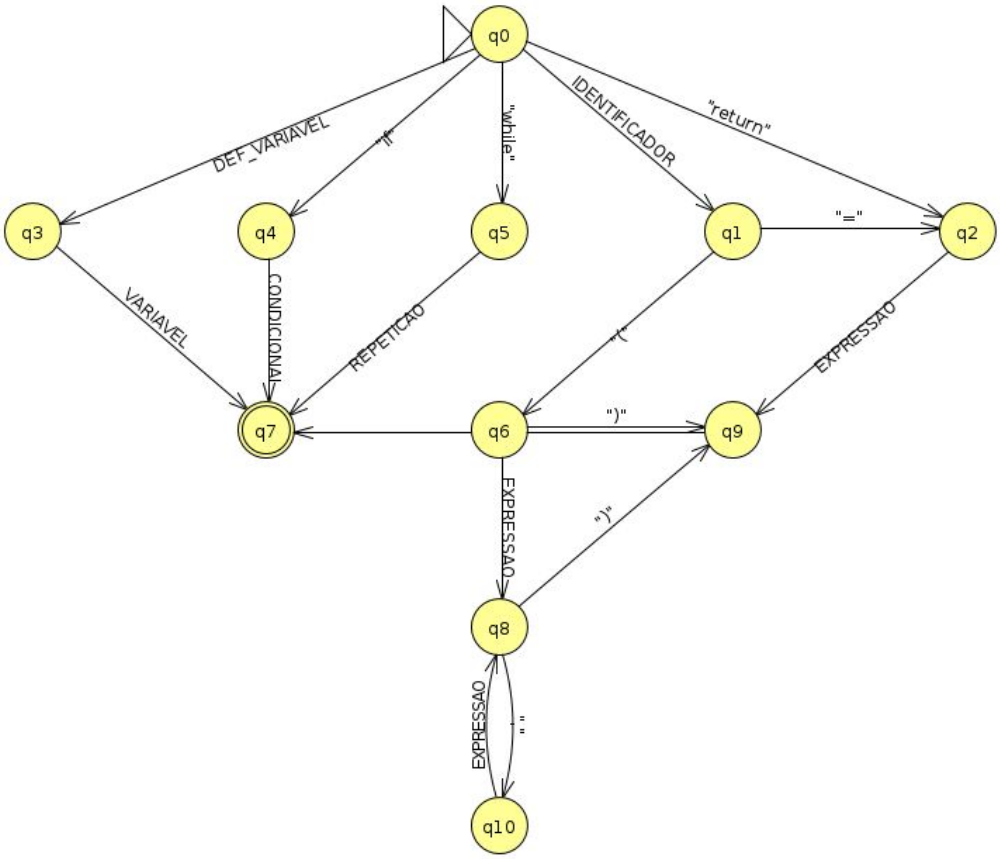
6. CONDICIONAL



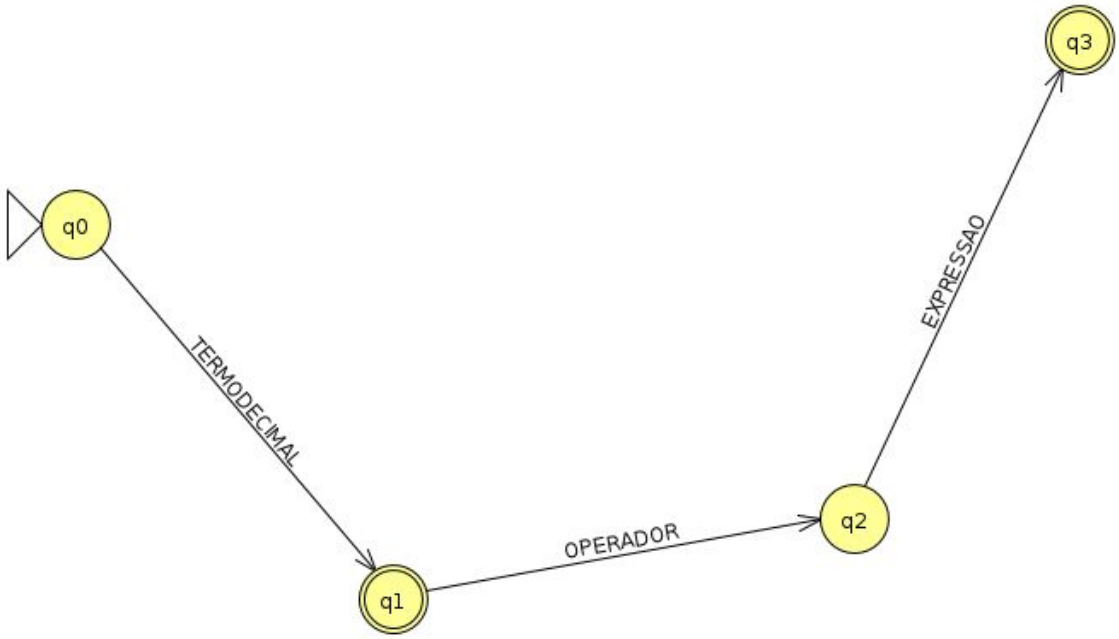
7. REPETICAO



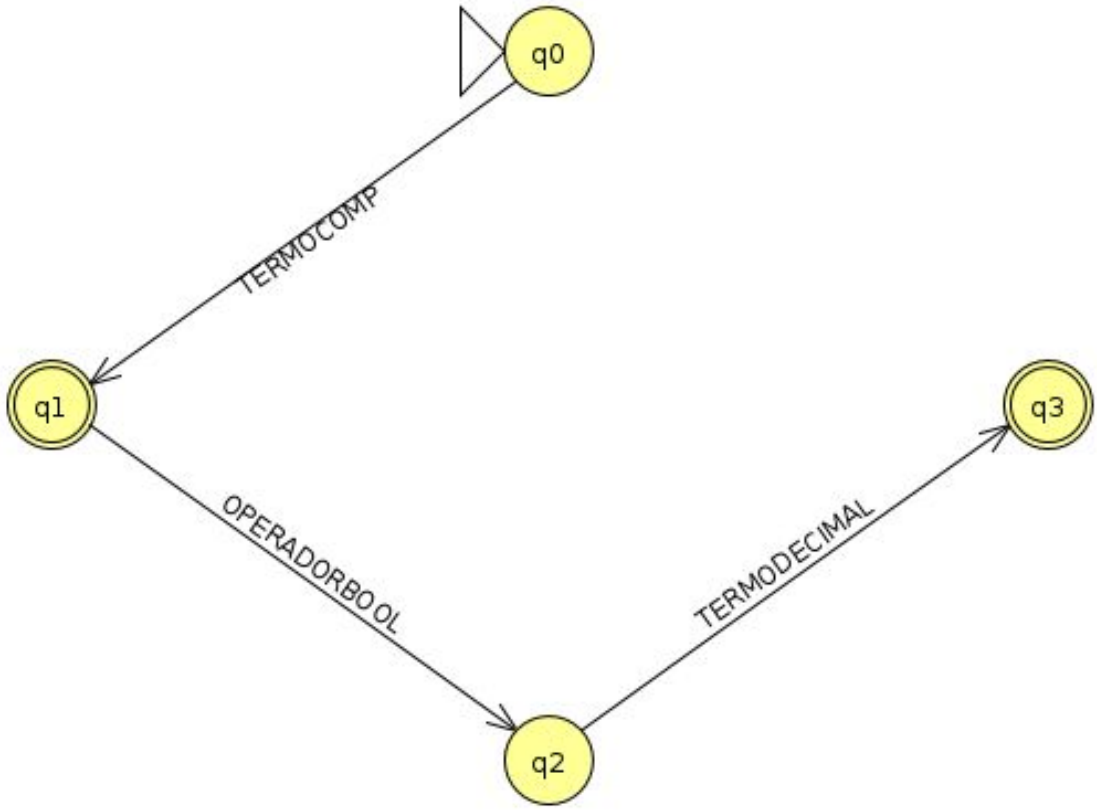
8. COMANDO



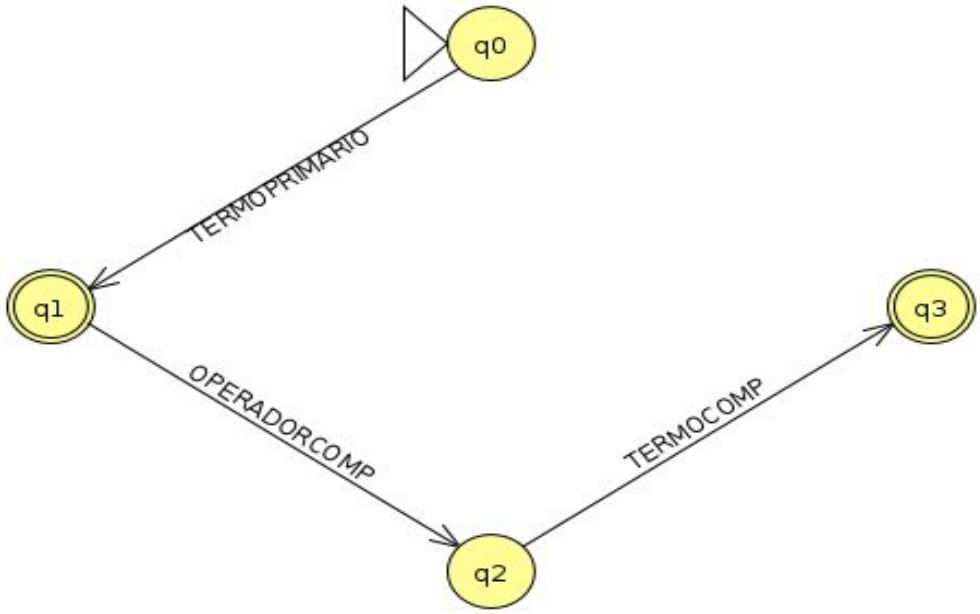
9. EXPRESSAO



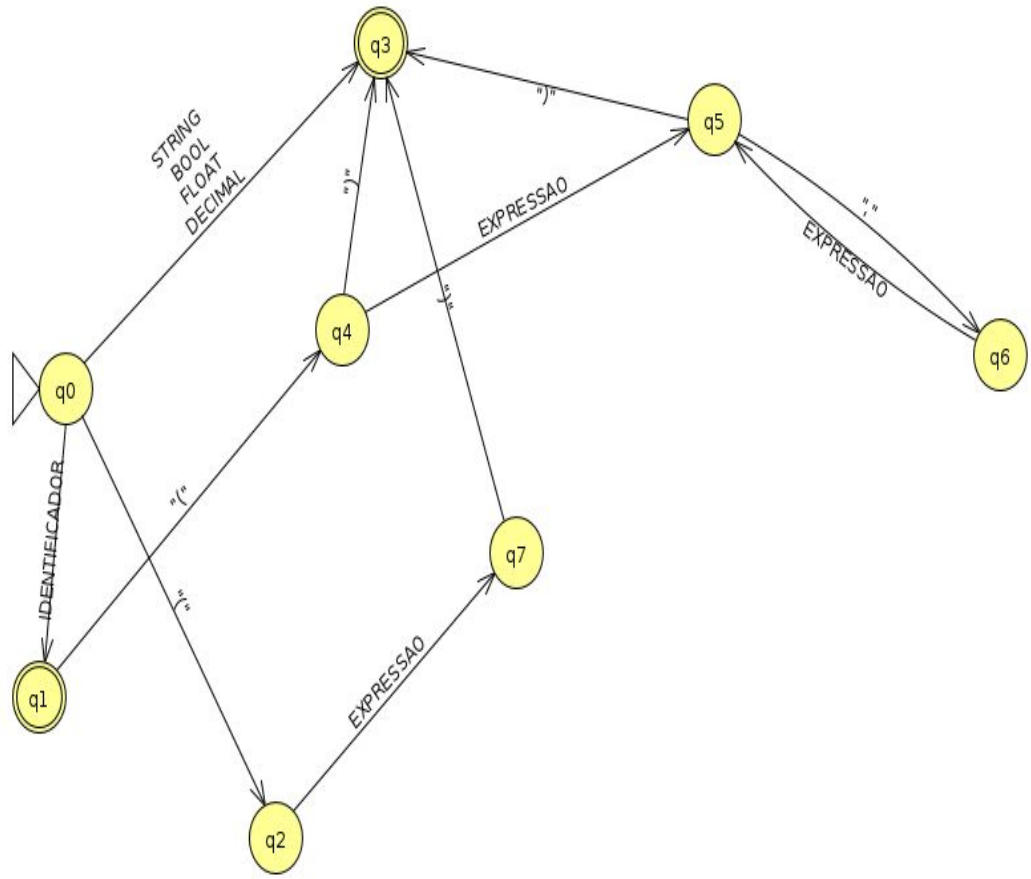
10. TERMODECIMAL



11. TERMOCOMP



12. TERMOPRIMARIO



3. Comentários sobre a implementação

A implementação do analisador léxico possui o arquivo *syntatic.c* e seu header *syntatic.h*. Ele possui uma função principal que está exposta chamada *compile* que possui como parâmetro o texto-fonte de entrada. Essa função:

1. Inicializa o estado do analisador sintático
2. Verifica se deve pegar outro token
3. Se sim, chama o analisador léxico e pega o nextToken. Caso ele seja nulo, interrompe-se a execução da análise e retorna.
4. Em seguida, atualiza-se o estado do autômato atual em função do estado do autômato atual e do token.
5. Caso seja necessário chamar uma sub-máquina, empilha-se a máquina atual e define-se o estado de retorno, altera-se o estado interno da analisador para utilizar a nova sub-máquina a partir do estado inicial.
6. Ao chegar em um estado final, desempilha-se a pilha de sub-máquinas e atualiza-se o estado interno do analisador.
7. Verifica-se se não houve erro na análise, e repete-se a partir de 1 caso não exista erros.

Para implementar essa função, foi necessário definir o estado interno do analisador sintático, com o estado atual do autômato, token atual e autômato atual e também uma pilha de sub-máquinas bem como funções de empilhar e desempilhar que além de alterar a pilha, atualizam o estado do analisador. Foi necessário também implementar funções que representassem os autômatos listados em 2.2.

Para integrar ao compilador, substituiu-se o loop do analisador léxico por uma chamada à função principal do analisador sintático.

Referências

1. Neto J. J. Introdução à Compilação. 1987.