Trabalho de Compiladores - 2022/1

Edmar Caixeta Filho
edmar.caixeta@ufms.br

Julho de 2022

1 Introdução

A disciplina de Compiladores I tem como objetivo estudar o processamento textual por trás de compiladores modernos, divida em três diferentes partes, sendo elas: Análise Léxica, Sintática e Semântica. O entendimento dessa área é de suma importância para formação acadêmica e profissional de um cientista ou engenheiro de computação, pois observa o processo de compilação que várias linguagens de computação sofrem. De forma que é bem claro que devemos entender as causas e não só as consequências dessas ferramentas.

O presente trabalho pertence a todo o âmbito da disciplina e requer a implementação na linguagem C++ dos analisadores, sendo assim dividido também em três etapas, sendo: Etapa 1, Análise Léxica; Etapa 2, Análise Sintática; e por último, Etapa 3, Análise Semântica. Logo, o presente relatório segue a mesma ordem em seu relato, mostrando as decisões de implementação, descrição e dificuldades.

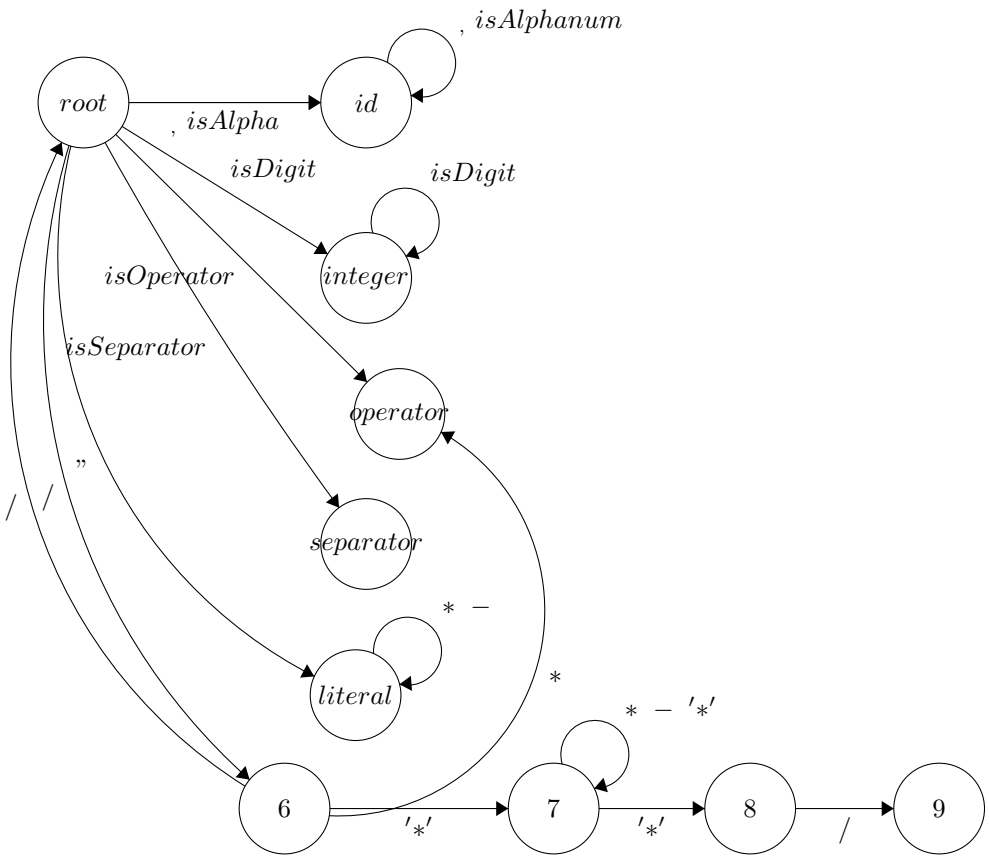
2 Análise Léxica

A análise léxica deve percorrer todo o texto e analisar tokens e seus lexemas e se pertencem a dada linguagem hipotética X++. Nessa etapa não deve-se realizar tratamento especiais para palavras reservadas da linguagem, logo deve-se apenas identificar se os seguintes tokens estão presentes:

- Identificadores: qualquer palavra começada com uma letra maiúscula ou minúscula ou por um _, podendo conter em seu infixo números.
- Operadores: símbolos aritméticos: +, -, /, *
- Separadores: os símbolos: (,), , , [,], , , ., ;
- Comentários: em linha ou em blocos, delimitados por // ou /* e fechados por */, respectivamente.

- Inteiros: qualquer dígito numérico, desde que não contenha separador para casas decimais.
- Literais do tipo String: sequência de caracteres diversos delimitados por "e ".

A parte de análise léxica foi implementada seguindo um modelo de autômato definido ilustrado abaixo:



Não foi ilustrado no autômato para se facilitar a visão, mas se o autômato está em um estado e não há uma transição definida para carácter atual, o autômato volta para o estado *root*, por exemplo, o autômato está no estado *integer* e espera ler dígitos e salvar isso no *buffer* até que receba um carácter não numérico. Dessa forma, o autômato entende que acabou o processamento do inteiro e assim aguarda o fim de arquivo ou o processamento do próximo *token*. Os estados 6, 7, 8 e 9 são para tratar comentários, pois o carácter "/"pode representar uma divisão e ser um operador ou pode ser um início de comentário em blocos ou comentário de linhas. O tratamento realizado foi armazenar "/"no

buffer e olhar para a próxima entrada, se é um asterisco o autômato ignora qualquer coisa até achar um "*"seguido de "/", caso seja um comentário em linha (//) o autômato cancela sua iteração e processa a próxima linha. Caso a barra seja sucedida de qualquer outro carácter é processada como operador de divisão.

Para carácteres não definidos o analisador retorna um aviso instantaneamente, porém continua sua execução.

O código pode ser compilado através do compilador Gnu g++, através do seguinte comando:

```
g++ application.cpp cpp_files/*.cpp -I header_files/  -o <nome_do_executável_desejado>
```

Além disso foi implementado também, a flag –*debug*, ao se executar o programa com a flag opcional o programa gera um arquivo de texto com as saídas do analisador léxico, logo é fácil visualizar o que será consumido pelo Parser. De forma que se segue:

```
./<nome_do_executável_gerado> <nome_do_arquivo_de_entrada> --debug
```

3 Análise Sintática

Após a realização da análise léxica, onde garantimos que não há a entrada de caracteres inválidos, partimos para a análise sintática, onde conferimos se o arquivo textual de entrada pertence a gramática definida para a linguagem X++. Nessa etapa, é necessário ser observado que algumas produções possuem recursão a esquerda diretamente e indiretamente. Dessa forma deve-se corrigir tais produções.

Após a correção das produções, foi se codificado de forma que todas as regras de derivação da gramática são métodos da classe Parser. Além do mais, com o intuito de componentizar a primeira etapa e a segunda etapa da forma mais independente possível foi-se implementada uma lista encadeada de tokens que é construída na primeira etapa e o construtor do Parser recebe essa lista e percorre ela avaliando a gramática. Essa lista é a saída gerada pela flag –debug citada acima.

4 Análise Semântica

A análise semântica não foi implementada.

5 Decisões de Projeto e Dificuldades

Inicialmente o compilador havia sido construído utilizando a ferramenta Gnu flex, porém foi se alertado pela professora que tal ferramenta não seria permitida. Logo, o código foi refatorado de forma que o processamento da linguagem seria codificado de forma "manual".

Como citado anteriormente, eu gostaria que cada etapa fosse executada da forma mais independente possível e por isto implementei uma lista encadeada que é gerada na primeira etapa e é utilizada como entrada para a segunda etapa, outra opção seria realizar a análise sintática em conjunto com a análise léxica.

Todo o código utilizado no projeto foi de minha autoria e está disponível publicamente no GitHub, não foi-se utilizado o esqueleto fornecido pela professora, embora haja partes semelhantes.