Universidade Federal da Fronteira Sul

Compiladores

Nome: Bárbara Pegoraro e Lucas Jaenisch

Chapecó, Março 2022.

Resumo

Neste relatório apresentaremos o desenvolvimento, bem como os resultados obtidos através dos processos de compilação. Tivemos como base, o desenvolvimento de um compilador que apresente as etapas léxicas, sintáticas e semânticas, com a ajuda de programas externos como o Gold Parser.

Introdução

Compilador é um programa que transforma um código escrito em alto nível para língua de máquina. Sendo assim, ele analisa cada caractere em busca de erros, bem como também verifica as estruturas formadas por esses caracteres.

Neste trabalho, utilizamos de uma ferramenta externa chamada Gold Parser para a criação da tabela LALR, utilizada para a verificação sintática, a partir da nossa linguagem base, a fim de economizar trabalho, e evitar erros.

A seguir explicaremos como foi feita a construção de cada fase, bem como a explicação da metodologia seguida para o desenvolvimento.

Referencial Teórico

Linguagem formal: também conhecida como linguagem de programação, procuram eliminar toda ambiguidade possível, garantindo assim que um comando e palavras reservadas tenham sempre o mesmo significado independentemente de onde apareçam no programa.

Estado final: um estado só é final se ele pode ser trocado por um símbolo terminal ou uma Epsilon produção.

Transições: transição é quando estando em um estado e a partir de uma produção atingir outro estado.

Autômato determinístico: separa cada símbolo de entrada existe exatamente uma transição de saída de cada estado.

Compilador: programa que transforma um código escrito em alto nível para língua de máquina.

Implementação e Resultado

Construímos o código com base na linguagem de programação python, devido a facilidade da manipulação de estruturas.

O nosso código foi implementado visando atender as exigências vistas em aula. Começamos portanto pela ideia de aproveitamento do código utilizado para o trabalho final da disciplina de LFA. Como o código utilizado foi escrito em linguagem C, e tinha limitações em relação ao tamanho da entrada, o projeto foi recomeçado do 0, com a construção de um novo autômato finito determinístico. Após, foi então criado um arquivo que representa o código escrito pelo programador, a fim de validação. Foi criada também a gramática livre de contexto utilizada para definir as estruturas da linguagem.

Temos como arquivos de entrada do programa, o arquivo de tokens, que é composto pela GLC e tokens utilizados, um arquivo para armazenar o “código” hipotético escrito pelo programador que será analisado, o arquivo da linguagem bnf, responsável pelo controle de estruturas permitidas na linguagem. E por último, um arquivo .xml, contendo as informações para a comparação sintática, gerado pelo programa Gold Parser.

O arquivo de entrada contendo o “código” escrito pelo programador será analisado e se não ocorrer nenhum erro, ele será compilado. Caso algum erro ocorra, o erro será apresentado. Os possíveis erros são:

A análise léxica varre todos os caracteres do arquivo de entrada, verificando se eles fazem parte da linguagem, e realizando uma determinada ação a cada caractere encontrado, adicionando-o a fita de saída e na tabela de símbolos.

Essa análise leva como base a ocorrência ou não na linguagem hipotética. Nossa linguagem hipotética apresenta 4 ações possíveis: atribuição de um valor a uma variável, declaração da variável, verificação de condição e uma estrutura de loop.

Sintática: recebe como entrada, o resultado da análise léxica. Procura por erros na estrutura sintática do programa, verificando se obedece às regras da gramática livre de contexto.

Semântica: Como semântica, definimos que seriam tratados os casos em que:

- Uma variável já foi declarada anteriormente;
- Uma variável está sendo usada fora do escopo em que foi declarada;
- Uma variável utilizada não foi declarada;

Com relação à geração de código intermediário, não conseguimos concluir a produção completa, portanto decidimos abandonar.

A otimização de código não foi realizada.

Com a supracitada metodologia de implementação, conseguimos resultados válidos, com margem de funcionamento satisfatória. Situações como escopos

vazios (representados por {}) não foram trabalhadas, e podem retornar erros sintáticos, já que o programa reconhece ‘}’ como sendo um carácter, não um operando, porém, dentro de uma margem tolerável.

Pode-se dizer portanto que o trabalho foi gerado satisfatoriamente.

Conclusão

Como descrito no presente artigo, realizamos as análises léxica, sintática e semântica, bem como tentamos a realização de geração do código intermediário, sem sucesso. Devido a utilização da base do autômato desenvolvido em LFA para a continuidade do trabalho, houveram algumas restrições quanto ao uso de alguns caracteres para a construção do conjunto de tokens que compõem a linguagem, porque não previmos a sua utilização e o autômato utiliza estes caracteres como sinalizadores ou marcadores, um exemplo são os caracteres '<' e '>' utilizados pelo autômato, e que não puderam ser utilizados como símbolo de ‘maior’ e ‘menor’, bem como do símbolo '*', definido como epsilon nas gramáticas regulares, e não pode ser utilizado como símbolo de multiplicação.

O trabalho, portanto, apresentou resultados satisfatórios na construção dos componentes requeridos pela disciplina de Construção de Compiladores.