Ivo Henrique Pedrini, Lorenzo Salume Zimerer, Vinicius de Abreu Bozzi

Prof. Eduardo Zambon

Compiladores

22 de Fevereiro de 2021

Checkpoint 2

O seguinte relatório de progresso no trabalho da disciplina de Compiladores, lecionada pelo professor Eduardo Zambon na Universidade Federal do Espírito Santo, tem como objetivo principal dar um retorno sobre o andamento do projeto até o momento. O projeto consiste no desenvolvimento de um compilador simples. O grupo foi contemplado com a linguagem fonte **Pascal** e a arquitetura alvo **MIPS.**

Para que a leitura dos relatórios seja independente, um breve resumo do que foi elaborado no Checkpoint 1:

Analisador Léxico (Scanner)

Com o scanner, a entrada é dividida em pequenos tokens. Para linguagens case-insensitive, como Pascal, o analisador léxico também precisa fazer a conjunção de tokens iguais, com diferenças apenas na sequência de letras maiúsculas/minúsculas. Os tokens, nesta etapa, ainda não têm “significado” específico.

Analisador Sintático (Parser)

Na etapa de análise sintática, os tokens, adquiridos na etapa anterior, vão ser associados entre si por meio de uma gramática, e seus comportamentos serão definidos.

Analisador Semântico

A etapa atual do projeto, e que, como o nome implica, envolve a análise semântica dos tokens obtidos.

A análise semântica, em si, é um processo de análise mais **contextual** do que **técnico** (este seria mais similar à análise sintática), tendo como objetivo encontrar possíveis erros conceituais, ou de incompatibilidade de tipos e operações entre eles. Outro objetivo da análise semântica é fortalecer "regras" que dificilmente seriam representadas na análise sintática por si só, por ex.: a verificação da declaração prévia de uma variável antes de seu uso; compatibilidade entre os tipos requeridos e os tipos atribuídos aos parâmetros de uma função; verificação de escopos de variáveis, entre outros.

Uma analogia possível é que a análise semântica pode ser vista como a automatização do processo de "chamar um amigo programador" para dar uma olhada rápida no seu código e te dizer o que não faz sentido para ele. Mesmo não sabendo seus objetivos, caso saiba como funciona a linguagem, ele deveria poder te dizer algo do tipo: "essa variável não foi declarada anteriormente" ou "essa função pede um inteiro e está recebendo uma string", apenas passando uma olhada rápida no código, esse seria o objetivo do analisador semântico.

O processo conta com algumas fases, a primeira fase tratada no trabalho foi a verificação de tipos, para a qual foi construída uma tabela de variáveis: armazenando seus nomes, a linha de

suas declarações e suas tipagens associadas, dessa forma, seria trivial verificar se havia declaração prévia, se houve alteração do tipo, entre outros. Vale mencionar que, como a atribuição dos tipos e inserção na tabela não eram parte da etapa anterior, foi necessário alterar um pouco as regras de gramática do parser para acrescentar essas funcionalidades.

Outra etapa importante é a elaboração de uma Abstract Syntax Tree (ou AST), que é, basicamente, uma árvore constituída por Declarações e Expressões, baseadas em uma linguagem formal, sendo construídas em conjunto. Seus objetivos são: facilitar a verificação de interações entre tipos diferentes e seus resultados e, quando usada em conjunto com linguagens especificadas por gramáticas livres de contexto, evitar a ambiguidade.

Detalhes da implementação:

A implementação continuada do CP1, foi seguida de acordo com os laboratórios utilizando o flex e bison. Devido a alta dificuldade para a implementação e falta de planejamento, foram feitas implementações simplificadas para a linguagem, bem parecidas com a da linguagem EZlang. Entretanto, foram adicionadas as funcionalidades para todos operadores de comparação da linguagem pascal, além dos operadores lógicos ‘and”, “or” e “not” e operações de divisão inteira como “mod” e “div”, sendo os lógicos e de divisão inteira não implementados para a AST. Foram incluídas as funcionalidades de if-then-else e repeat como sendo a estrutura de repetição. Foi realizada a implementação para o tipo composto array (de inteiros), apenas para conferência de sua regra semântica e também não foi implementada para a AST, assim como demais funcionalidades como as declarações de funções e procedures.

Considerações finais da implementação: os testes feitos que se encontram na pasta “tests” tem o seguinte objetivo: os arquivos renomeados com 1 na frente, são funcionais para as regras semânticas, já os que tem 2 na frente, são funcionais também para AST.

CONCLUSÃO

Apesar das limitações na implementação do trabalho, até o momento, foi de enorme esclarecimento em relação ao funcionamento das etapas de análises léxica, sintática e semântica de compiladores simples.