Universidade de Évora
Departamento de Informática
Compiladores
Ano letivo 2019 - 2020

Compilador da Linguagem YA!

Alunos:
Marcelo Feliz - 38073
Rúben Teimas - 39868

Docente:
Pedro Patinho

10 de Junho de 2020

Indice

1 Introdução

O objetivo deste trabalho é que implementemos um compilador para a linguagem *YA!*.

A linguagem *YA!* apresenta uma sintaxe parecida à linguagem de programação *C*, contudo esta é bastante mais simples e restrita. Uma das restrições da linguagem *YA!* é o facto da mesma só suportar 2 *scopes* em simultaneo, ou seja, não permite definir funções dentro de funções.

Para que o compilador esteja totalmente funcional é necessário que todas as etapes funcionem sem problemas: a análise lexical, análise sintática, análise semântica e geração de código.

Para o compilador em questão a representação intermédia não foi pedida, o que significa que vamos gerar código dependente de uma arquitetura. A arquitetura aconselhada foi a arquitetura *MIPS*.

Para a implementação do trabalho foi utilizada a linguagem de programação *C* juntamente com ferramentas de análica lexical e sintática compativeis (*Lex* e *Bison* respetivamente).

2 Desenvolvimento

O compilador começa, lógicamente, na análise lexical que irá receber do ficheiro de input uma *stream* de *chars* e transforma-la numa coleção de *tokens*, que estão definidos no ficheiro *ya.lex*.

Após estar completa a análise lexical o output da mesma (coleção de *tokens*) será enviado ao analisador sintático (*ya.y*) onde estão definidas as "regras da nossa gramática", ou seja, a sintaxe aceite pela nossa linguagem.

Durante a análise sintática o compilador verifica se existem ou não erros de sintaxe. Se no final existirem erros o compilador termina a sua execução, pois não serve de nada continuar.

Caso não existam erros o analisador sintático irá gerar a *APT* com a qual iremos trabalhar até ao fim do programa, pois não existe representação intermédia.

Para análise semântica precisamos da *APT*, gerada na análise sintática, e de uma *Symbol Table*.

Como a linguagem *YA!* permite apenas 2 *scopes* em simultâneo a nossa *Symbol Table* é nada mais do que 2 *hashtables*. Caso permitisse mais do que 2 *scopes*, uma alternativa para a *Symbol Table* poderia ser uma *Linked-List* de *Hashtables*.

Tendo a *APT* gerada e a *Symbol Table* pronto a utilizar podemos começar a análise semântica na qual ser verifica, como o nome indica, se existem erros semânticos. As regras da análise semântica estão definidas no ficheiro apt.c.

A nossa análise semântica ficou quase completa, ficando a faltar colocar os argumentos na Symbol Table.

Tendo erros na análise semântica, não conseguimos assim avançar para a fase da geração de código. No conteudo da nossa Symbol Table deveriamos também ter uma espaço para guardar o offset das variáveis e dos argumentos para que este offset pudesse posteriormente ser usado pela máquina de pilha.

3 Conclusão

Embora o resultado do trabalho tenha ficado um pouco aquém das expectativas, foi muito importante para nos ajudar a perceber o desenho de um compilador.

Para além dos conceitos teóricos necessários, a concepção do trabalho foi também bastante interessante pois obrigou-nos a usar conceitos de EDA, de ASCI e permitiu-nos ver a aplicação real de alguns conceitos de LFA.

Conseguimos também usar conceitos desta cadeira noutras cadeiras, como é o caso de LP.