UNIVERSIDADE D
COIMBRA

Projeto de Compiladores 2021/22

Compilador para a linguagem deiGo

Trabalho realizado por:

- António Correia (N.º 2019216646)
- Mário Lemos (N.º 2019216792)

Índice

Opções Técnicas

Gramática

Na implementação da nossa gramática foram adicionadas regras auxiliares com o intuito de agilizar o processo de transformação da gramática inicial (a qual se encontra em notação **EBNF)** que nos foi providenciada no enunciado em uma gramática final capaz de ser analisada sintaticamente com o *yacc*. Ao analisar as especificações da linguagem **Go** (disponível na bibliografia e no enunciado do projeto) encontrámos as precedências da mesma, sendo essas as que usámos na nossa gramática.

Algumas das inconsistências na gramática final (por exemplo duas regras diferentes com uma notação **EBNF** semelhante que seriam "tratadas" da mesma maneira, não o são) devem-se ao tratamento dos *Shift Reduce*'s os quais são devolvidos quando uma sequência de *tokens* lida pelo *yacc* pode ser associada a mais do que uma regra, fazendo com que o compilador não saiba qual delas escolher.

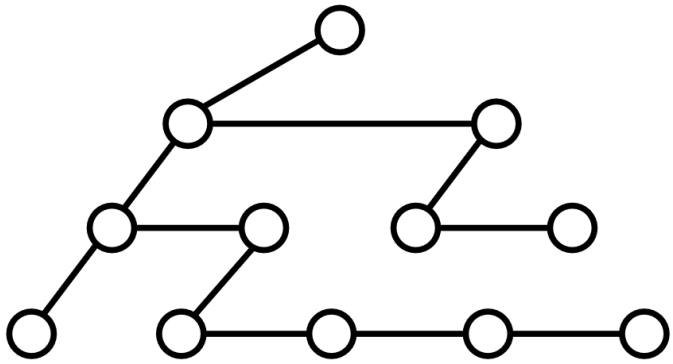
Algoritmos e estruturas de dados

A **árvore de sintaxe abstrata** usada no nosso projeto, em vez de uma estrutura de múltiplos filhos, como uma árvore normal, usa uma estrutura de um filho e irmãos. Em vez de um nó ter várias ligações a outros nós abaixo do mesmo, este tem apenas uma ligação ao seu filho e, caso exista, ao seu irmão. A estrutura utilizada tem este funcionamento pois, ao analisar o código do utilizador, há sempre um número diferente de relações entre os nós. Com esta estrutura, não só fica resolvido esse problema, como se torna mais fácil percorrer os filhos de um nó, não sendo necessário estar sempre a voltar ao pai quando se quer passar para um irmão.

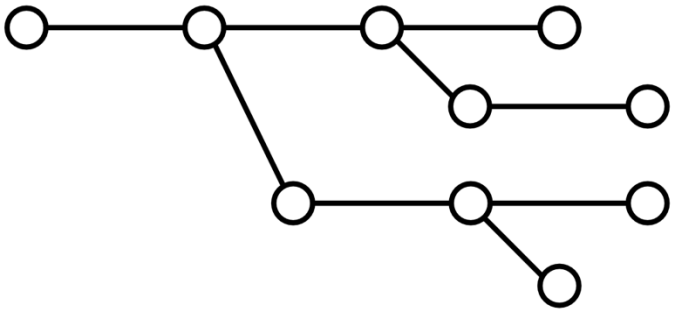
Na estrutura de cada nó, além dos ponteiros para o filho e para o irmão, há também três *strings* (**id**, **token** e **annotation**) que correspondem, respetivamente, ao identificador, valor e anotação do nó (para a árvore anotada), dois inteiros para a linha e a coluna (para localização de erros) e um *bool* para saber se, ao anotar a árvore, o nó já foi anotado ou não (para evitar a repetição de erros).

Em termos de funcionalidade, a árvore conta com funções para criar um novo nó, adicionar um irmão a um nó existente e imprimir recursivamente a árvore.

A estrutura da árvore (em forma visual) consiste no seguinte:



A **tabela de símbolos** foi implementada como uma lista ligada com ramificações. Esta é uma lista ligada normal, mas com um ponteiro extra para uma ramificação da mesma. Assim, ao analisar a árvore, torna-se mais simples distinguir *scopes*. Se uma lista ligada é uma ramificação, significa que todos os símbolos nessa lista vão ser locais, mas também poderão ser utilizados nas ramificações da mesma. Os símbolos na primeira lista ligada (a que não é uma ramificação) são os símbolos globais. A estrutura, em forma visual, consiste no seguinte:



Na estrutura de cada nó há três s*trings*, que correspondem a um nome, um parâmetro e um tipo, um *bool* **isParameter** e quatro inteiros, dois para linhas e colunas, um para o número de parâmetros e um último para saber se foi usado ou não.

No que toca a funções, existem as funções esperadas de criar um novo símbolo, uma nova ramificação e imprimir a árvore com as anotações. Além disso existem ainda diversas funções com o objetivo de preencher a tabela de símbolos e anotar a árvore de sintaxe.

Geração de código

A estrutura de dados usada nesta parte do projeto consiste numa lista duplamente ligada, de modo a ser possível navegar livremente pelas variáveis do programa para ser possível obter o tipo de um registo e qual o valor do próximo registo disponível.

O código começa a analisar a árvore gerada nos passos anteriores, tendo também ajuda da tabela de símbolos. Quando encontra uma variável, o programa "carrega" os registos necessários através da instrução **global** e continua a analisar a árvore. Caso encontre uma função, **alloca** e dá **store** dos argumentos da mesma antes de prosseguir com a anális. Dentro da função, caso encontre variáveis, tem o mesmo comportamento de quando encontra uma variável fora de uma função, mas, em vez de usar a instrução **global** usa a instrução **alloca**.

Por fim, se encontrar um *return* utiliza a instrução **ret** para definir um resultado da função na qual se encontra. Caso não encontro um *return* dentro da função, automaticamente coloca um **ret** com um valor *default* dependendo do tipo da função na qual se encontra.

Bibliografia

Especificações da linguagem Go

https://go.dev/ref/spec