Relatório Compiladores

Licenciatura em Engenharia Informática
2018/2019
Compilador para a linguagem deiGo

Diogo Alexandre Santos Amores Nº 2015231975 damores@student.dei.uc.pt
Maria Inês António Roseiro Nº 2015233281 miroseiro@student.dei.uc.pt

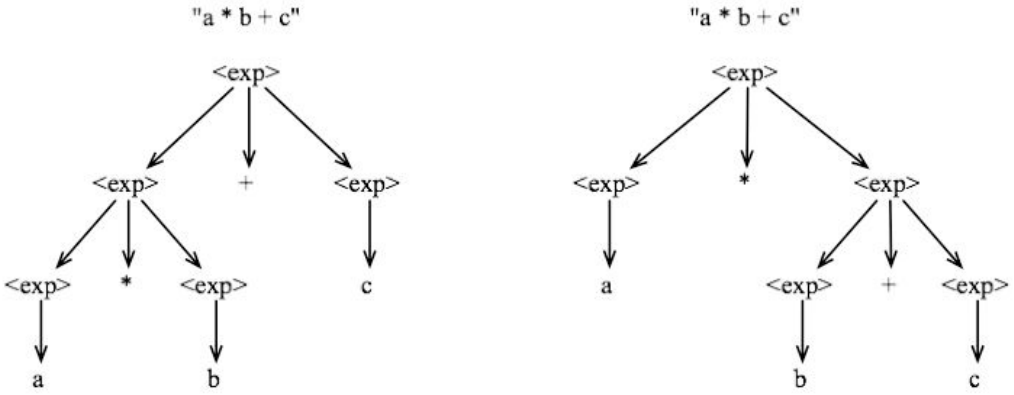
Gramática Re-escrita

Para a Meta 2, é fornecida a gramática inicial em notação EBNF, este gramática inicial não é suportada pelo *yacc* e é ambígua pois admite várias produções para o mesmo caminho (exemplificando, as produções **Expression** da gramática EBNF são ambíguas, pelas razões explicadas acima).
Assim, é necessário re-escrever esta gramática para permitir a análise sintática com a ferramenta *yacc*, para esta re-escrita, focamo-nos essencialmente na introdução de precedências nos operadores conforme a linguagem Go e na reestruturação da gramática. Para esta reestruturação, em geral, foram criadas produções auxiliares para tratar de ciclos (produções opcionais e de zero ou mais repetições).

Para a seguinte produção ambígua em notação EBNF:

Expr → Expr (PLUS | MINUS | STAR | DIV | MOD) Expr

Esta produção é ambígua pois podemos ter cadeias deste género:



Por este motivo, foram implementadas precedências nos operadores usando associatividade à esquerda, direita e nenhuma associatividade, solucionando este problema. Foi usado como referência a documentação de Go para implementação destas precedências.

Transformação da produção da linguagem EBNF em linguagem aceite pelo *yacc:*

Expr:
Expr PLUS Expr
| Expr MINUS Expr
| Expr STAR Expr
| Expr DIV Expr
| Expr MOD Expr

Para a seguinte produção com zero ou mais repetições em EBNF:

VarSpec → ID {COMMA ID} Type

Transformação da produção da linguagem EBNF em linguagem aceite pelo *yacc*:

VarSpec:
ID CommaAux Type

CommaAux:
| Comma ID CommaAux

Para a seguinte produção com elementos opcionais em EBNF:

Statement → Return [Expr]

Transformação da produção da linguagem EBNF em linguagem aceite pelo *yacc:*

Statement:
RETURN
| RETURN Expr

AST e Tabela de Símbolos

Para estruturas de dados da AST e Tabela de Símbolos foram usadas as seguintes estruturas:

```
typedef struct node{
    struct node* brother;
    struct node* child;
    char *name; //label of the node
    char *value; //value of the node, if it has any
    char *annotation;
    int line;
    int column;
} node;
```

A *struct* node é usada na representação de um nó da árvore de sintaxe abstrata. Os primeiros dois elementos desta estrutura correspondem aos nós irmão e filhos desse nó, *name* vai corresponder à "etiqueta" desse nó (ou seja, Strlit, Reallit, ect...), o seu *value* (valor), se tiver, *annotation* com a finalidade de anotar os tipos para a árvore anotada. *Line* e *column* tinham a finalidade de auxiliar no controlo dos erros, guardando a linha e a coluna destes. Esta estrutura é utilizada em conjunto essencialmente com as funções *create_node, add_child* e *add_brother* para construir a árvore.

```
typedef struct func *func_list;
typedef struct func{
    symb_table *table; //de que tabela é a funcao
    vars_list func_vars; //possiveis variaveis
    param_list func_param; //parametros da funcao
    func_list next;
}func;
```

Estrutura base da tabela de símbolos, esta estrutura é uma lista ligada que corresponde às funções da tabela de símbolos (ou varíaveis globais), esta estrutura têm como elementos *symb_table table* que corresponde à tabela de símbolos correspondente, a uma lista ligada de variáveis dentro da função (*func_vars,* se tiver) e uma lista ligada de parâmetros (*func_param,* se tiver), e um apontador para *next* para percorrer a lista. O

header da lista e tabela global são guardados globalmente com os nomes *func_header* e *global_table*.

```
typedef struct symb_table{
    char *table_name; //nome da table
    char *table_type; //tipo da tabela
    int func_check; //func ou global
}symb_table;

typedef struct params *param_list;
typedef struct params{
    char *param_name; //nome do param
    char *param_type; //tipo do param
    param_list next; //next da lista de parametros
}params;

typedef struct vars *vars_list;
typedef struct vars{
    char *var_name;     //nome da var
    char *var_type;     //tipo da variavel
    vars_list next;
}vars;
```

Estas estruturas correspondem a estruturas auxiliares a *func_list*, a estrutura *symb_table* guarda um nome da tabela (Global ou de alguma função), o tipo da tabela se tiver (*table_type*), e um int *func_check* que serve como *"check"* se a tabela é referente a uma função ou a uma variável global. Depois, a estrutura *params* corresponde a uma lista ligada de parâmetros de uma função (que pode ter ou não, como explicado em cima), esta estrutura contém o nome do parâmetro (*param_name),* o tipo do parâmetro (*param_type*) e um apontador para *next*. A última estrutura *vars* também corresponde a uma lista ligada de variáveis (mais uma vez, pode existir ou não), esta estrutura contém o nome da variável (*var_name),* o tipo da variável (*var_type*) e um apontador para *next*.