Universidade de Coimbra

Compiladores

LEI - DEI - 2021/2022

Projeto de Compiladores

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Miguel Pedroso - 2019218176
Ricardo Simões - 2019231869

December 18, 2021

1 Gramática reescrita

Foi feita a conversão da gramática EBNF para BNF, ficando a gramática não ambígua e permite ao *yacc* lidar com os casos de elementos repetidos ou opcionais. Para tal, foram usadas as seguintes regras:

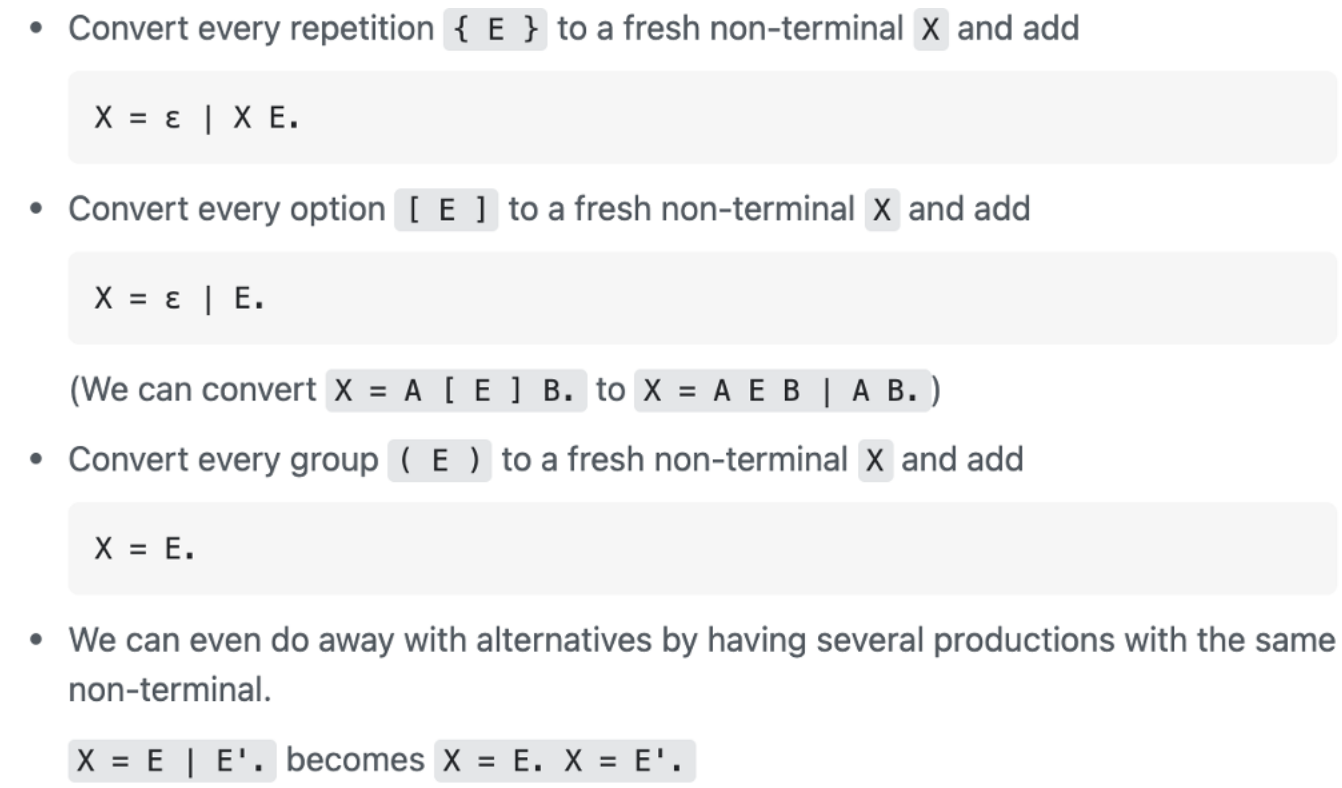

Figure 1: Regras de passagem de uma gramática EBNF para BNF

Também foram definidas prioridades para os vários operadores, de acordo com as regras da linguagem DeiGo, em que os operadores de baixo têm prioridade sobre os de cima:

%left OR
%left AND
%left GE GT LT LE EQ NE
%left PLUS MINUS
%left STAR DIV MOD
%left UNARY
%left ID

Finalmente, para evitar qualquer conflito de parêntesis, foi utilizado o nonassoc.

2 Algoritmos e estruturas de dados da AST e da tabela de símbolos

Os diferentes algoritmos e estruturas de dados estão divididos entre três principais ficheiros: functions.c, semantics.c e symbol_table.c.

No ficheiro functions.c encontram-se todas as funções dedicadas a operações com nós na AST, tais como adicionar um nó geral e criar um token (insert_node), inserir nós do tipo var_declaration (insert_vardec_node),inserir um nó auxiliar (insert_var_aux),e imprimir os nós da árvore corretamente (print_node).

As estruturas utilizadas para guardar dados na AST encontram-se no ficheiro structures.h e são as seguintes:

```
typedef struct token{
    class cl;
    char* symbol;
    char* identificador;
    char *type;
    int line;
    int column;
    int flag;
}token;

typedef struct node{
    token* t;
    struct node* next;
    struct node* first_son;
}node;

typedef struct var_aux{
    char* id;
    struct var_aux* next;
}var_aux;
```

Figure 2: Estruturas da AST

No ficheiro symbol_table.c estão todas as funções relacionadas com adicionar elementos às tabelas respetivas, bem como a sua busca e verificações adicionais. A função insert_el insere um novo elemento na cauda da lista ligada. A função show_table imprime os conteúdos da tabela. A função search_el é utilizada para procurar um elemento na tabela, e devolve 0 caso este não exista.

A função check_return utiliza a tabela para confirmar se o tipo de return de uma função coincide com o tipo do elemento que está a ser devolvido.

Para implementar a tabela foi utilizada uma lista ligada, e a estruturas de dados utilizadas encontram-se no ficheiro symbol_table.h e são as seguintes:

```
typedef struct _t0 {
    char* nome;
    struct _t0 *next;
} arg;

typedef struct _t1 {
  char name[32];   //nome da variavel ou funcao
  basic_type type; //tipo da variavel ou retorno da funcao (pode ser none)
  struct _t0 *first_son;  //tem os tipos dos parametros da funçao
  struct _t1 *next;
  int genero;
} table_element;

typedef struct _t2 {
  char name[50];
  struct _t0 *first_param;  //tem os tipos dos parametros da funçao
  struct _t2 *next;
  table_element *first_child;
} table;
```

Figure 3: Estruturas das tabelas de símbolos

No ficheiro semantics.c encontram-se todas as funções relacionadas com a análise semântica. O ficheiro consiste em duas funções principais: check_type, e compara tipo.

A função comparaTipo é utilizada para averiguar se expressions são válidas ou não, tendo em conta o tipo dos operandos e o operador.

A função check_type averigua que operações vão ser necessárias tendo em conta o tipo do nó recebido. Ambas as funções em casos específicos lidam com mensagens de erro no caso de existirem erros semânticos no ficheiro de input.