Escola de Ciências e Tecnologias

Curso: Engenharia Informática

Disciplina: Compiladores

Professores: Salvador Abreu e Vasco Pedro

Compilador de VSPL

Trabalho Prático de Compiladores

Trabalho Elaborado Por:

Marlene Oliveira, Nº 25999

Pedro Mateus, Nº 26048

João Aiveca, Nº 26175

Ano Lectivo 2011/2012

Introdução

No âmbito do segundo trabalho prático da disciplina de compiladores foi-nos solicitada a elaboração de um compilador para a linguagem VSPL (*Very Simple Programming Language*).

A VSPL é uma linguagem de programação de fácil implementação que possui características das diversas linguagens de programação imperativas. Como recorre pouco a palavras reservadas, esta linguagem evita questões linguísticas. Para facilitar a implementação de um compilador para esta linguagem foram impostas as seguintes restrições à mesma:

- A execução de um programa consistirá numa activação da função *program*, que deve ser definida pelo programador e não possui argumentos nem valor de retorno (void em ambos os casos);
- A definição de um nome, caso esta exista, deve ser a primeira definição do mesmo (deve ocorrer antes de qualquer uso);
- As definições do não-terminal *program* são reconhecidas em todo o programa (são designadas como definições globais);
- A passagem de parâmetros é sempre feita por valor;
- Uma função só pode devolver valores inteiros e booleanos;
- Os parâmetros de uma dada função não podem ser outras funções;
- Um tuplo só deve conter elementos do tipo inteiro ou booleano;
- Um array só pode conter tuplos ou elementos do tipo inteiro ou booleano;
- Uma declaração só pode aparecer nos argumentos formais de uma função (desde que a declaração seja do tipo “id : [*] tipo”).

Para implementar a gramática utilizámos o bison. Nos locais apropriados fazemos prints que serão interpretados pelo ficheiro apt.pl. De um modo geral, os prints são no formato [Input novo|X]-[Ligação|X]. O input novo é retirado da árvore nos nós folha, e a ligação vai servir para o prolog unificar os nós. O valor X serve para guardar a árvore inteira do programa.

Devido às nossas dificuldades, o passo que se segue (tratar a árvore para a preparar para a geração de código) não foi implementado. Assim sendo, a apt.pl imprime no stdout o resultado da árvore lida da gramática.

A geração de código não foi implementada.

Analisador Lexical

O analisador lexical do nosso trabalho contém todos os terminais da linguagem VSPL, bem como as definições e regras que permitem reconhecer os padrões correspondentes aos identificadores, inteiros, booleanos, reais, espaços em branco e comentários.

As regras elaboradas foram as seguintes:

- {ID} {yylval.identif = strdup(yytext); return ID;} – Permite reconhecer o padrão correspondente aos identificadores da linguagem. Tal como é indicado na definição ID [a-zA-Z_][a-zA-Z0-9_]*, os identificadores são sequências de caracteres. O conjunto de caracteres disponíveis para as ditas sequências contém todas as letras maiúsculas e minúsculas, todos os números de 0 a 9 e o caractere *underscore*.
- {INT} {yylval.inteiro = atof(yytext); return INT_LIT;} – Permite reconhecer o padrão correspondente aos valores do tipo inteiro.
- {REAL} {yylval.real = atof(yytext); return REAL_LIT;} – Permite reconhecer o padrão correspondente aos valores do tipo real.
- {WTS} { } – Permite identificar os padrões correspondentes aos espaços, indicados na definição: WTS [\t\n]+. Os espaços em branco serão ignorados.
- {COM} { } – Permite identificar os padrões correspondentes aos comentários. A definição COM "#".*\n indica que, sempre que for encontrado um padrão que comece com '#' seguido de qualquer sequência de caracteres e seja terminado por um *newline*, estamos perante um comentário. À semelhança do que ocorre com os espaços em branco, também os comentários devem ser ignorados.
- Como o ELSE pode ser representado pelo caractere *, tal como acontece com a multiplicação e com o WHILE, foi criada a seguinte regra: "*"/{WTS}"->" {return ELSE;}. Assim, sempre que for encontrado um * seguido de um espaço e de um ->, sabemos que estamos perante um ELSE.
- Foi elaborada uma regra semelhante à do ELSE de modo a que se pudessem distinguir os *whiles*. A regra criada foi a seguinte: "*"/{WTS}"[" {return ELSE;}. Assim, sempre que for encontrado um * seguido de um espaço e de um [, sabemos que estamos perante um WHILE.
- "true" {yylval.boolean = 1; return BOOL_LIT;} – Permite identificar a constante booleana correspondente ao booleano “true”. Note-se que “true” corresponde ao valor 1.
- "false" {yylval.boolean = 0; return BOOL_LIT;} – Permite identificar a constante booleana correspondente ao booleano “false”. Note-se que “false” corresponde ao valor 0.

Nota: Todos os outros terminais podem ser consultados no ficheiro vspl.l.

Analisador Sintáctico

O analisador sintáctico do nosso trabalho permite que o compilador reconheça as várias produções que constituem a gramática da linguagem VSPL restrita.

Foi criado um *union* que permite especificar uma colecção de tipos de dados para os valores semânticos dos identificadores, inteiros, booleanos e reais.

Todos os *tokens* foram declarados na zona da gramatica apropriada. Os *tokens* declarados foram os seguintes:

%token <identif> ID

%token OP AND OR NOT

%token <inteiro> INT_LIT

%token <real> REAL_LIT

%token <boolean> BOOL_LIT

%token INT REAL BOOLEAN VOID

%token RETURN BREAK SKIP RETRY MAP

%token COND WHILE ELSE

%token LESSEQ MOREEQ DIF APT AFECT

Os operadores que associam à esquerda são os seguintes:

- '+'
- '-'
- '*'
- '/'
- AFECT (token que corresponde ao terminal ":=" – afectação)
- APT (token correspondente ao terminal "->")
- AND
- NOT
- '.'
- '%'
- OR
- ','

Os operadores que não associam à esquerda nem à direita são os seguintes:

- '('
- '['

- ']'
- ')'
- '<'
- '>'
- LESSEQ (token que corresponde ao terminal “<=”)
- MOREEQ (token que corresponde ao terminal “>=”)
- DIF (token corresponde ao terminal “<>”)
- '='

O símbolo inicial da gramática será o não-terminal program.

A gramática utilizada neste trabalho foi a disponibilizada no enunciado do mesmo.

Tabela de Símbolos

O ficheiro apt.pl recebe como *input* o output da gramática. Cada linha é terminada com um '.', pelo que a função *read* vai lendo uma linha de cada vez – os prints na gramática foram feitos especificamente para permitir este comportamento. Dada a disposição das variáveis na gramática, a unificação do prolog, quando obtém argumentos suficientes, trata de os associar; por exemplo, um *assign* com três argumentos vai inicialmente ler cada um dos argumentos para a lista que está a ser construída, e unifica após encontrar [A],[B],[C]|X no argumento antes do separador (-), e assign(A,B,C)|X no argumento seguinte.

A cada linha tratada, o *input* é impresso no *stdout*.