

Compiladores – Ano lectivo 2011/12 **Linguagem VSPL**

Informação de versão: Id: sintaxe-vspl.tex,v 1.3 2012/03/09 08:02:10 spa Exp

Especificação

Para o desenvolvimento dos trabalhos práticos, utiliza-se uma linguagem designada por VSPL (*Very Simple Programming Language*) na qual se encontram características diversas das linguagens de programação imperativas que exercitam um leque alargado de situações.

1 Elementos lexicais

As convenções lexicais são as habituais, nomeadamente:

1.1 Comentários

Um comentário começa com o caracter '#' e termina no fim da linha em que este ocorre.

1.2 Identificadores

Os identificadores têm a definição habitual, coincidindo com os da linguagem C. A linguagem VSPL é sensível às diferenças entre minúsculas e maiúsculas (i.e. diz ser “case-sensitive”).

1.3 Palavras Reservadas

Por regra geral, as palavras reservadas têm como definição a própria palavra, *em minúsculas*. Algumas palavras reservadas podem aceitar uma representação textual alternativa:

Terminal	Representação
AND	&
OR	
NOT	~
RETURN	^
COND	?
WHILE	*
ELSE	*

Atenção que estas variantes podem causar colisões com as restantes definições de símbolos terminais, pelo que o tratamento destas situações poderá ser efectuado com precaução, havendo várias abordagens possíveis nomeadamente a nível da análise sintáctica.

1.4 Constantes (Literais)

O analisador lexical deverá reconhecer constantes (literais) dos 3 tipos base apresentados, nomeadamente:

Constantes inteiras (INT_LIT). São constantes inteiras decimais, expressas pela definição habitual. Só são contempladas as este nível as constantes positivas, ie. sem sinal.

Constantes de vírgula flutuante (REAL_LIT). Tal como as anteriores, estas seguem as convenções habituais. No entanto, deverão ser reconhecidos, por exemplo, valores nas seguintes formas: “.8”, “0.005”, “123e+17”, “1.5e2”, “3e-5” e “.200284E3”.

Constantes booleanas (BOOL_LIT). As constantes booleanas, literalmente true e false.

2 Sintaxe

A linguagem VSPL é apresentada informalmente pela gramática das figuras 1 a 5 (ver páginas 2 a 5). Esta gramática já se encontra numa forma *fácilmente adaptável*¹ para especificar como input para um gerador de parsers LALR(1) como o CUP. Por uma questão de legibilidade (e tipografia) a gramática foi repartida em várias secções.

```
program -> decls                                /* Símbolo inicial */

decls -> /* VAZIO */                             /* Lista de declarações */
      | decls decl

decl ->                                         /* Declaração dum nome: */
      ids '=' type ';'                          /* Definição de tipo */
      | ids ':' type ';'                       /* Variável, tipo explícito */
      | ids ':' type ':=' exp ';'             /* Variável, tipo explícito, init */
      | ids ':' type '=' exp ';'             /* Constante, tipo explícito */
      | ids      '=' exp ';'                 /* Constante, tipo implícito */

formals -> /* VAZIO */                          /* Lista de parâmetros formais */
      | formal_decl formals

formal_decl ->                                 /* Parâmetro formal: */
      ids ';'                                  /* Tipo implícito */
      | ids ':' type ';'                      /* Tipo explícito */

ids -> id                                       /* lista de identificadores */
      | id ',' ids

id -> ID | OP op

op -> '+' | '-' | '*' | '/' | '%'             /* Os operadores */
      | AND | OR | NOT
      | '<' | '<=' | '=' | '>' | '>=' | '>'
```

Figura 1: EBNF para a linguagem VSPL – Declarações

Algumas observações sobre a gramática da linguagem VSPL:

- As instruções em VSPL devem ser terminadas explicitamente, daí aparecer o símbolo “;” como terminador, na linha das linguagens C ou Java.
- Um *programa* em VSPL consiste numa sequência de declarações.
- A linguagem tem inferência de tipos, pelo que as declarações de nomes poderão omitir o seu tipo.
- A identidade de tipos é estrutural, pelo que tipos anónimos ou com nomes diferentes podem ser considerados idênticos, desde que a sua estrutura coincida.
- O constructor de “tuplo anónimo” (o símbolo “,”) pode ser utilizado para construir *expressões primárias* ou *restritas* (ver figura 4). Na versão aumentada da linguagem, estas podem encontrar-se à esquerda dum símbolo de afectação (‘:=’). Na versão base tal não é permitido pelo que o não terminal “primary” parece ser inútil nesta gramática.
- As especificações de tipo compreendem agregados. Estes podem ser, nomeadamente, anónimos (*tuplos*) ou etiquetados. Esta última possibilidade é expressa pela última regra do símbolo não terminal `single_type`, como se pode ver na figura 2.

¹Não significa que esteja *directamente* utilizável, tão somente que a adaptação é um processo simples.

- A regra do não-terminal “sexp” que começa por “CLASS” destina-se a expressar constantes de tipos agregados heterogêneos, p/ex. CLASS a: int, b: bool [a := 3; b := true].

```

type ->
    single_type          /* -- ASSINATURA DE TIPO -- */
    | '(' type ')'      /* Um só tipo (fim de lista) */
    | single_type ',' type /* Agrupamento sintactico */
                                /* Tuplo de tipos (lista) */

single_type ->
    ID                  /* -- EXPRESSÃO DE TIPO -- */
    | INT               /* Identificador de tipo */
    | REAL              /* Inteiro */
    | VOID              /* Vírgula flutuante */
    | BOOL              /* Booleano */
    | type '->' type    /* Void (ex. instruções de controle) */
    | '[' exp ']' type /* Tipo funcional */
    | '{' formals '}'  /* Tipo "Array" */
                                /* Tipo agregado (classe) */

```

Figura 2: EBNF para a linguagem VSPL – Declarações de tipo

3 Notas sobre Semântica

- A execução do programa consiste numa activação da função `program`, função esta que deverá ser definida pelo programador e não tem argumentos nem valor de retorno (tipo vazio em ambos os casos).
- Caso exista uma *definição* para um nome, **é necessário que esta ocorra antes de qualquer uso**, i.e. deve ser a primeira ocorrência do nome.
- As definições que constituem o não-terminal `program` dum programa serão designadas como definições *globais*, pelo que são reconhecidas em todo o programa.
- A passagem de parâmetros é sempre efectuada por *valor*.

4 Restrições à Linguagem

Para facilitar a implementação dum compilador para VSPL, são impostas algumas restrições à linguagem, que seguem:

1. Tipos de dados: só existem os tipos inteiro, booleano e os constructores de tipo array, tuplo e função. Em particular desaparecem os tipos classe e string.
2. Os literais de função só podem ocorrer no nível lexical mais alto dum programa (i.e. não há funções dentro de funções), e como valores para símbolos constantes.
3. Declaração implícita: se um nome for declarado implicitamente (por ter uma ocorrência de uso antes duma de definição), sê-lo-há necessariamente no bloco mais interior em que ocorre, independentemente de poder haver uma declaração posterior em blocos exteriores.

```

exp -> sexp                                     /* -- EXPRESSÃO -- */
    | sexp ',' exp
    | '(' exp ')'
```

```

sexp -> sexp OR sexp                           /* Operadores booleanos */
    | sexp AND sexp
    | NOT sexp

    | sexp '<' sexp                             /* Operadores de comparação */
    | sexp '<=' sexp
    | sexp '=' sexp
    | sexp '>' sexp
    | sexp '>=' sexp
    | sexp '>' sexp

    | sexp '+' sexp                             /* Operadores aritméticos */
    | sexp '-' sexp
    | sexp '*' sexp
    | sexp '/' sexp
    | sexp '%' sexp
    | '-' sexp

    | sexp '.' ID                               /* Nomes qualificados */
    | sexp '[' exp ']'                          /* Referências a arrays */
    | sexp '(' exp ')'                          /* Aplicação funcional */
    | '@' '(' exp ')'                          /* Aplicação recursiva directa */

    | ID                                         /* Nome simples */

    | INT_LIT                                   /* Constante inteira */
    | REAL_LIT                                  /* Constante em vírgula flutuante */
    | BOOL_LIT                                  /* Constante booleana */

    | '[' exp ']'                              /* Literal de array */
    | MAP '(' formals ')' '[' stats ']'        /* Literal funcional */
    | MAP '(' formals ')' '->' type           /* Idem, com tipo explícito */
    | '[' stats ']'
    | CLASS '(' formals ')' '[' stats ']'     /* Literal de classe */

```

Figura 3: EBNF para a linguagem VSPL – Expressões

```

primary -> prim
    | primary ',' prim
    | '(' primary ')'
```

```

prim -> ID
    | prim '.' ID
    | prim '[' exp ']'

```

Figura 4: EBNF para a linguagem VSPL – Expressões restritas

```

stats -> /* VAZIA */                               /* -- INSTRUÇÕES -- */
        | stat stats

stat -> decl                                       /* Declaração (já inclui o ";") */
        | prim ':=' exp ';'                       /* Afectação *** ATENÇÃO *** */
        | prim '(' exp ')' ';'                   /* Chamada de função */
        | RETURN exp ';'                          /* Retorno de função */
        | BREAK ';'                               /* Saída de ciclo */
        | SKIP ';'                                /* Salto para o fim do ciclo */
        | RETRY ';'                               /* Regresso para o início do ciclo */
        | COND '[' clauses ']'                   /* Instrução condicional */
        | WHILE '[' clauses ']'                 /* Instrução de ciclo condicional */
        | '[' stats ']'                           /* Agrupamento de instruções */

clauses -> exp '->' stats                          /* Instrução com guarda */
        | exp '->' stats '|' clauses
        | exp '->' stats '|' ELSE '->' stats

```

Figura 5: EBNF para a linguagem VSPL – Instruções