Material de Apoio #69

Esquemas de tradução para arranjos

1 Formulação

A definição do endereço dentro de um arranjo depende da organização adotada na linguagem de programação: se é por linha, coluna ou por vetor de acesso. No que segue, adotamos a organização **por linha**. O cálculo do endereço de uma determinada célula de um arranjo multidimensional é feita de maneira recursiva. Por fins de otimização, podemos dividir o cálculo em duas partes: na declaração e no acesso.

Na declaração, todas as informações a respeito do endereçamento são conhecidas (quantas dimensões, tamanho e limite de cada dimensão), permitindo a definição de uma constante C_A que é registrada na entrada da tabela de símbolos para o identificador do arranjo. Seja $A[low_0..high_0][low_1..high_1]..[low_k..high_k]$ os limites das dimensões, *base* o endereço de base do arranjo (de acordo com a posição de sua declaração e dependendo do escopo), e w o tamanho do elemento deste vetor (relacionado ao tipo), o cálculo do valor constante C_A de um arranjo A é dado pelo seguinte:

$$C_A = base - r_k * w \tag{1}$$

$$r_k = \begin{cases} low_k & \text{se } k = 0 \\ r_{k-1} * |high_k - low_k| + low_k & \text{se } k \geq 1 \end{cases} \tag{2}$$

No acesso, as informações dos índices (identificando unicamente uma célula) estão disponíveis. Os índices podem eventualmente ser calculados a partir de expressões aritméticas. Seja $A[i_0][i_1]..[i_k]$ os índices, o cálculo do endereço final é dado pelo seguinte (w é o tamanho do elemento, C_A foi calculado na declaração):

$$endereco = C_A + d_k * w \tag{3}$$

$$d_k = \begin{cases} i_k & \text{se } k = 0 \\ d_{k-1} * |high_k - low_k| + i_k & \text{se } k \geq 1 \end{cases} \tag{4}$$

2 Esquemas de tradução

O conjunto de instruções ILOC é utilizado quando necessário.

Para a **declaração** do arranjo (e cálculo do C_A), temos o seguinte esquema:

D	→	N_1 .. N_2	{ `D.low = N1.val; D.high = N2.val; D.n = mod(D.high - D.low);` }
L	→	D	{ `L.r = D.low; L.dim = 0; def_dim_size(L.dim, D.high, D.low);` }
L	→	L_1 D	{ `L.r = L1.r * D.n + D.low; L.dim++; def_dim_size (L.dim, D.high, D.low);` }
DECL	→	T **ident [** L **]**	{ C_A `= base - L.r * T.w; decl(ident, T.type, T.w,` C_A`, get_dim_sizes());` }

Para o **acesso** de um arranjo (supondo declaração dentro de uma função – por isso `fp`), temos o seguinte esquema:

L	→	D	{ `L.dim = 0; L.code = D.code; L.d = D.temp;` }
L	→	L_1, D	{ `L.d = temp(); L.dim = L1.dim+1; n = get_dim_size (ident, L.dim); x = temp();` `L.code = L1.code || D.code ||` “`mult` L_1`.d, n => x” ||` “`add x, D.temp => L.d”;` }
ACES	→	**ident [** L **]**	{ `x=temp(); ACES.temp=temp();` `ACES.code = L.code ||` “`mult w, L.d => x” ||` “`add` C_A`, x => y” ||` “`loadAO fp, y => ACES.temp”;` }

3 Um exemplo de uso de declaração

Para uma declaração de arranjo:

```
int cubo[3..5][-2..3][-4..10];
```

obtemos um valor de C_A de 178, conformo Figura 1.

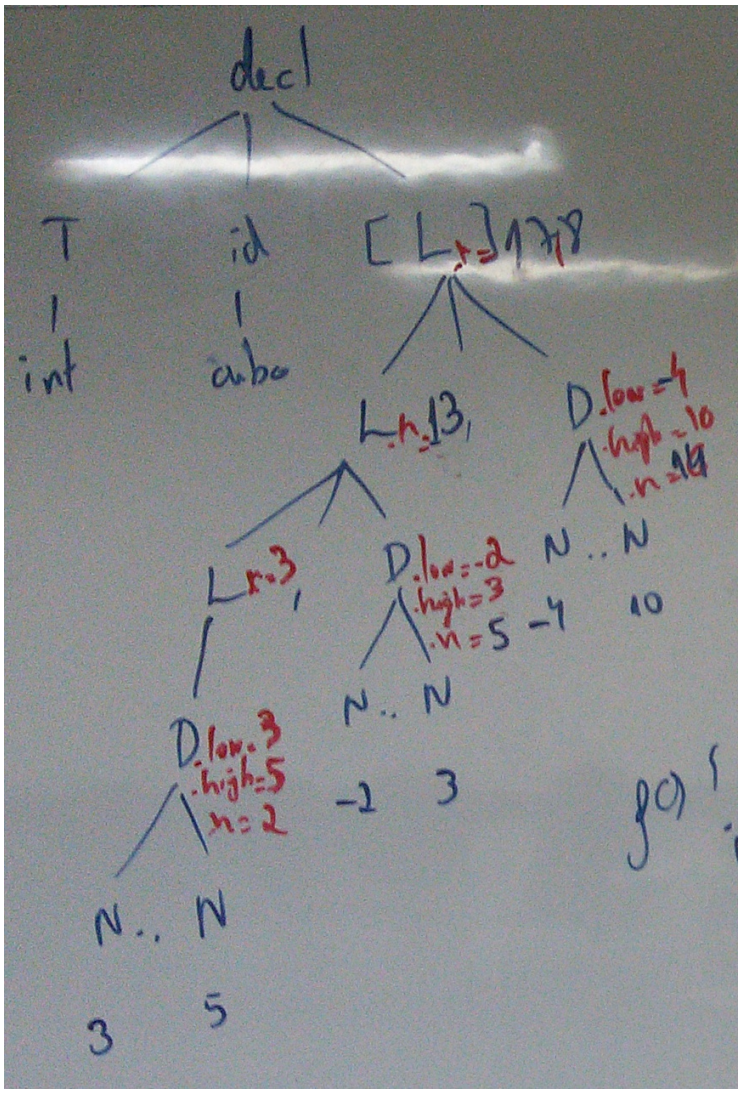

Figura 1: Solução do exemplo de uso.