Azzolini Riccardo 2019-03-04

Linguaggio AG

1 Limiti del modello RAM

I programmi RAM sono di difficile comprensione: poiché le istruzioni sono di tipo assembly, il codice è poco sintetico e intuitivo.

L'obiettivo è utilizzare un linguaggio che sia sintetico e intuitivo, ma comunque facile da ricondurre al linguaggio RAM, in modo che sia semplice valutare la complessità dei comandi.

2 Linguaggio AG

Il linguaggio procedurale AG è un linguaggio ad alto livello che:

- permette di analizzare la complessità senza la traduzione esplicita in linguaggio RAM;
- è basato su variabili e costrutti ad alto livello (selezione, iterazione, ecc.);
- evita dichiarazioni di tipo (quando è chiaro dal contesto).

3 Variabili

Una **variabile** è un identificatore X di una particolare zona di memoria.

A ogni variabile è associato un **tipo**: un insieme $\mathcal{U}$ dei possibili valori che tale variabile può assumere durante l'esecuzione del programma. $\mathcal{U}$ può essere costituito da numeri, parole o strutture dati complesse (vettori, liste, ecc.).

Nel modello RAM, una variabile è rappresentata da uno o più registri. Il loro stato costituisce il **valore corrente** della variabile, che può essere modificato tramite *istruzioni di assegnamento*.

3.1 Left value e right value

Una variabile X ha effettivamente due valori:

right value: il *valore corrente* contenuto nella locazione di memoria corrispondente, usato quando X appare "a destra" di un assegnamento (o, più in generale, in un'espressione);

left value: l'indirizzo della locazione di memoria, usato quando X si trova "a sinistra" di un assegnamento.

4 Espressioni

Un'**espressione** è un termine che denota l'applicazione di *simboli di operazioni* a *variabili* o a *valori costanti*.

In pratica, è qualsiasi combinazione di costanti, variabili e operatori che rispetta le regole del linguaggio.

Per i tipi complessi, è possibile definire specifiche operazioni tramite *procedure*.

Durante l'esecuzione, le espressioni hanno un *valore corrente* (*right value*), il risultato dell'applicazione delle operazioni sui valori correnti delle variabili, ma non hanno un *left value*. Non è quindi possibile assegnare un valore a un'espressione.

4.1 Esempi

- $(X + Y) * 2$
- $\textsc{Enqueue}(l, i)$

5 Condizioni

Una **condizione** è un *simbolo di predicato* applicato a una o più *espressioni*.

Il valore di una condizione in un certo stato, che corrisponde al risultato dell'applicazione del predicato ai valori correnti delle espressioni, è `Vero` (spesso denotato con 1) o `Falso` (0).

6 Sintassi

- **Assegnamento**: $V := E$
- **Selezione** (a due vie): **if** P **then** C_1 **else** C_2
- **Iterazione**
  - ciclo **for**: **for** $k = 1$ **to** n **do** C
  - ciclo **while**: **while** P **do** C
- **Sequenza**: **begin** C_1; C_2; ... C_n; **end**
- **Comando etichettato**: $e : C$
- **Salto**: **goto** e

dove:

- V è una variabile;
- E è un'espressione;
- P è una condizione;
- e è un'etichetta;
- $C, C_1, \ldots, C_n$ sono comandi.

7 Semantica e complessità

7.1 Assegnamento

$$V := E$$

assegna alla variabile V il valore corrente dell'espressione E.

Complessità: Somma di

- costo della valutazione di E (dipende dalle operazioni e vai valori correnti delle variabili coinvolte);
- costo di modifica dello stato di V (dipende dal numero di bit necessari a rappresentare i valori del tipo di V).

7.2 Selezione

$$\textbf{if } P \textbf{ then } C_1 \textbf{ else } C_2$$

esegue C_1 se la condizione P è vera, altrimenti esegue C_2.

Complessità: Somma di

- costo della valutazione di P;
- costo di C_1 se P è vera, altrimenti costo di C_2.

Di conseguenza,

$$[\![\textbf{if } P \textbf{ then } C_1 \textbf{ else } C_2]\!] \leq [\![P]\!] + \max([\![C_1]\!], [\![C_2]\!])$$

dove con $[\![C]\!]$ si indica il costo del comando C.

7.3 Ciclo for

$$\textbf{for } k = 1 \textbf{ to } n \textbf{ do } C$$

esegue in successione C, in cui la variabile k assume valore $1, 2, \ldots, n$.

Complessità: $\sum_{i=1}^{n}$ costo di C con $k = i$.

7.4 Ciclo while

$$\textbf{while } P \textbf{ do } C$$

se P è vera, esegue C, e ripete l'operazione finché P diventa falsa.

Complessità: $\sum$(costo di C + costo della valutazione di P) nei vari stati.

Osservazione: La condizione P viene sempre valutata una volta in più rispetto al comando C.

7.5 Sequenza

$$\textbf{begin } C_1;\ C_2;\ \ldots\ C_n;\ \textbf{end}$$

esegue in successione $C_1, C_2, \ldots, C_n$.

Complessità: $\sum_{i=1}^{n}$ costo di C_i.

7.6 Comando etichettato

$$e : C$$

esegue C. Può essere la destinazione di un comando di salto.

Complessità: Costo di C.

7.7 Salto

$$\textbf{goto } e$$

rimanda l'esecuzione al comando con etichetta e.

Complessità: Costante.

8 Esempio di programma

Calcolo del prodotto di n numeri (per $n \in \mathbb{N}$ qualsiasi):

```
begin
    ACC := 1;
    Read(X);
    while X ≠ ♭ do
        begin
            ACC := ACC × X;
            Read(X);
        end;
    Write(ACC);
end
```

Si può analizzare direttamente la complessità:

- ogni iterazione esegue un numero costante di operazioni, ciascuna a sua volta di costo costante (in base al CCU);
- vengono eseguite n iterazioni.

La complessità è quindi $\Theta(n)$.

9 Sottoprogrammi

È possibile scrivere **sottoprogrammi** che possono essere richiamati dal programma principale.

Essi sono di due forme:

funzione: calcola e *restituisce un valore*, che viene poi esplicitamente utilizzato dal programma principale;

procedura: *modifica lo stato* del programma principale.

La sintassi per la definizione di una funzione o procedura è

$$\textbf{Procedura } \textsc{Nome}(\underline{\lambda})\ C;$$

dove:

- NOME è l'*identificatore* del sottoprogramma;
- $\underline{\lambda}$ è la lista dei **parametri formali**;
- C è un comando che
  - per le funzioni *deve* contenere **istruzioni di ritorno** (`return` E);
  - per le procedure può anche non contenere istruzioni di ritorno.

L'invocazione di una funzione si effettua mediante la sintassi

$$A := \textsc{Nome}(\underline{B});$$

dove $\underline{B}$ è la lista dei **parametri attuali**. Al momento dell'esecuzione:

1. i valori o gli indirizzi di $\underline{B}$ vengono copiati in $\underline{\lambda}$;
2. il controllo viene passato al sottoprogramma;
3. quando si esegue un comando "`return` E", il valore di E viene copiato in A.

L'invocazione delle procedure, corrispondente alla sintassi

$$\textsc{Nome}(\underline{B});$$

è simile a quella delle funzioni: l'unica differenza è che una procedura non restituisce nessun valore, quindi può solo modificare lo stato delle variabili del programma.

9.1 Passaggio di parametri

Passaggio per valore/copia: Nei parametri formali vengono copiati i *valori* dei parametri attuali. Di conseguenza, l'eventuale modifica di un parametro formale *non ha effetto* sul parametro attuale corrispondente.

Passaggio per riferimento/indirizzo: Nei parametri formali vengono copiati gli *indirizzi* dei parametri attuali. Ogni modifica di un parametro formale *si riflette* sula parametro attuale.

9.2 Complessità

La complessità di una funzione o procedura è la somma di:

- costo del sottoprogramma;
- lunghezza dei parametri, se passati *per valore* (gli indirizzi hanno invece lunghezza costante, che si può quindi trascurare);
- lunghezza del valore restituito, nel caso di una funzione (corrisponde al costo del comando **return** E).

9.3 Esempi

Procedura $\textsc{Max}(x, y)$
  if $x > y$ **then return** x
    else return y

Procedura $\textsc{Scambia}(x, y)$
  begin
    $t := x;$
    $x := y;$
    $y := t;$
  end

Se $V[I] := 4$ e $V[J] := 7$:

- l'invocazione di funzione $A := \textsc{Max}(V[I], V[J])$ assegna il valore 7 ad A;
- l'invocazione di procedura $\textsc{Scambia}(V[I], V[J])$ fa sì che $V[I] := 7$ e $V[J] := 4$, purché i parametri siano passati per riferimento.

10 Puntatori

Un **puntatore** è una variabile X che assume come valore corrente l'*indirizzo di un'altra variabile*.

- La variabile puntata da X si denota con $*X$.
- Durante l'esecuzione, X può non essere definita (*nil*):

$$\mathcal{U} = \{nil\} \cup \{i \mid i \text{ indirizzo RAM}\}$$

I puntatori sono una visione più ad alto livello dell'indirizzamento indiretto, e vengono spesso usati per memorizzare strutture dati complesse, che richiedono più registri.

Nota: I riferimenti del linguaggio Java sono di fatto puntatori (a oggetti).

10.1 Esempio di complessità

Se $*X$ e $*Y$ contengono matrici $n \times n$, allora:

- $*X := *Y$ ha costo $\Theta(n^2)$ perché assegna a $*X$ una copia della matrice;
- $X := Y$ ha costo $\Theta(1)$ perché copia in X l'indirizzo a cui punta Y, e gli indirizzi hanno dimensione costante, indipendente dall'oggetto a cui puntano.