Capitolo 6:
Modelli di calcolo per linguaggi imperativi e funzionali

Modelli imperativi: le RAM (Random Access Machine)

I modelli di calcolo imperativi sono direttamente collegati all'architettura dei primi calcolatori elettronici (modello di von Neumann) e caratterizzano il linguaggio macchina, i linguaggi assemblativi e gran parte dei linguaggi di programmazione ad alto livello dei calcolatori reali.

Nel modello di calcolo imperativo un algoritmo è descritto tramite una sequenza finita di istruzioni (*programma*), che agisce sui dati contenuti in un insieme finito di celle (*memoria*).

Il calcolo è realizzato da una macchina che esegue *sequenzialmente* le istruzioni, modificando il contenuto della memoria. Tipiche istruzioni sono la *lettura* dei dati, il *trasferimento*, le *operazioni aritmetico-logiche* elementari, la *visualizzazione* di risultati.

In questo modello è fondamentale il concetto di *stato*, cioè il contenuto della memoria in un determinato istante. L'esecuzione di una istruzione comporta il passaggio da uno stato al successivo, e l'intera computazione si rappresenta attraverso una sequenza di stati.

LA RAM è un tipo particolare di modello di calcolo imperativo introdotta da Shepherdson e Sturgis nel 1963 per sviluppare la teoria della calcolabilità per un modello astratto di un reale calcolatore.

È caratterizzato da una *memoria* consistente in un numero arbitrario di celle (*registri*), R[1], R[2], ... ognuna delle quali contiene un intero positivo grande a piacere. I registri sono accessibili *direttamente* attraverso il loro numero d'ordine.

L'*accumulatore* (registro R[0]) contiene, durante la computazione, uno degli operandi su cui agiscono le istruzioni della macchina.

Il *contatore delle istruzioni* (CI) contiene il numero d'ordine della prossima istruzione da eseguire.

Lo scambio di informazioni con il mondo esterno avviene mediante due nastri, di *ingresso* e di *uscita*, consistenti in una sequenza di celle, ciascuna capace di contenere un intero grande a piacere.

L'*unità centrale* esegue le istruzioni del linguaggio di programmazione della macchina.

Il *programma* per RAM consiste in un insieme di *istruzioni* di vario tipo, che agiscono su *operandi* contenuti nei registri. Si adottano le seguenti convenzioni:

- 3 indica il contenuto del registro 3.
- (3) indica il contenuto del registro indirizzato dal registro 3 (*indirizzamento indiretto*).
- ♯3 indica che l'operando è l'intero 3.

<programma>	::=	<istruzione>	{<istruzione>}
<istruzione>	::=	**LOAD**	[♯]<operando> | (<operando>)
		STORE	<operando> | (<operando>)
		ADD	[♯]<operando> | (<operando>)
		SUB	[♯]<operando> | (<operando>)
		MULT	[♯]<operando> | (<operando>)
		DIV	[♯]<operando> | (<operando>)
		READ	<operando> | (<operando>)
		WRITE	[♯]<operando> | (<operando>)
		JUMP	<etichetta>
		JGTZ	<etichetta>
		JZERO	<etichetta>
		HALT	
<operando>	::=	<intero>	
<etichetta>	::=	<intero>	

Le *istruzioni di trasferimento* (**LOAD** e **STORE**) trasferiscono il contenuto di un registro nell'accumulatore, e viceversa.

Le *istruzioni aritmetiche* (**ADD**, **SUB**, **MULT**, **DIV**) eseguono le operazioni tra il contenuto dell'accumulatore e quello di un registro.

Le istruzioni sono eseguite sequenzialmente tranne nei casi in cui si presenti una instruzione di controllo (**HALT** o salto).
Il calcolo *termina* quando si incontra una istruzione di **HALT**.

Le *istruzioni di salto* sono salti incondizionati (**JUMP**) o condizionati dal contenuto dell'accumulatore (**JGTZ**, **JZERO**);
l'operando di una istruzione di salto è il numero d'ordine dell'istruzione alla quale si effettua il salto.
Il calcolo *non è definito* quando si salta a un'istruzione inesistente.
Il programma si arresta comunque.

Le *istruzioni di I/O* (**READ** e **WRITE**) leggono un numero intero dal nastro di input e lo trasferiscono in un registro o scrivono il contenuto di un registro sul nastro di uscita, rispettivamente.

La principale differenza fra una RAM e un calcolatore reale è che il programma della RAM non è contenuto in memoria, ma è *cablato*.

Per definire completamente il comportamento di un programma per RAM, assumeremo che all'inizio le informazioni siano fornite ordinatamente sul nastro di ingresso e che CI sia inizializzato a 1.

Definizione formale di alcune istruzioni: indichiamo con $R[i]$ il contenuto del registro i-esimo, con $R[0]$ il contenuto dell'accumulatore, con CI il contenuto del contatore delle istruzioni e con n un intero non negativo.

LOAD $\sharp n$	$R[0] := n$ $CI := CI + 1$
ADD n	$R[0] := R[0] + R[n]$ $CI := CI + 1$
ADD (n)	$R[0] := R[0] + R[R[n]]$ $CI := CI + 1$
JZERO n	if $R[0] = 0$ then $CI := n$ else $CI := CI + 1$

Che funzione è calcolata dal seguente programma?

```
     READ   1
     READ   2
     LOAD   #1
     STORE  3
5    LOAD   2
     JZERO  14
     LOAD   1
     MULT   3
     STORE  3
     LOAD   2
     SUB    #1
     STORE  2
     JUMP   5
14   WRITE  3
     HALT
```

Esercizio: definire il programma che calcola le funzioni resto, fattoriale e parte intera di $\log_2 x$, con $x \geq 0$. Determinare (vedi paragrafo successivo) il costo di esecuzione dei programmi, con i due modelli di costo.

Modelli di costo per i programmi RAM: un metodo semplice

Prima di correlare il comportamento delle RAM con le macchine di Turing è necessario chiarire come si valuta il costo di esecuzione di un programma RAM.

Nel **modello a costi uniformi**, ogni istruzione ha un costo unitario, quindi basta contare quante volte le istruzioni sono eseguite (in funzione dell'input del programma).

Nell'esempio precedente, ci sono tre parti:
✓lettura dell'input e inizializzazione (costo 4)
✓esecuzione del ciclo (costo $9y$+2)
✓terminazione (costo 2).

In totale, il costo di computazione è $9y + 8 \in O(y)$.

Due ipotesi troppo semplificative nel modello a costi uniformi:

1. il costo di esecuzione è indipendente dalla lunghezza degli operandi;
   calcolare 10×10 *costa quanto calcolare* $1.000.000 \times 1.000.000$

2. l'accesso a un registro ha costo unitario a prescindere dal numero di registri accessibili.
   accedere ad un registro su 10 costa quanto accedere ad un registro su 1.000.000

Modelli di costo: un metodo piú raffinato

Realisticamente, il costo di esecuzione delle istruzioni deve dipendere dalla lunghezza degli operandi. Se si opera su un intero n (contenuto in un registro), si paga un costo dell'ordine di $\log n$. Inoltre, il costo di accesso alla memoria deve dipendere dalla quantitá di memoria indirizzabile.

Sia $l(n) =$ if $n = 0$ then 1 else $\lceil \log_2 n \rceil + 1$.

Nel **modello a costi logaritmici**, elaborare un operando n ha costo $l(n)$; accedere a un registro i ha costo $l(i)$.

I costi delle istruzioni sono in tabella:

LOAD	op	c(op)
STORE	op	c'(op) $+ l(R[0])$
ADD	op	c(op) $+ l(R[0])$
SUB	op	c(op) $+ l(R[0])$
MULT	op	c(op) $+ l(R[0])$
DIV	op	c(op) $+ l(R[0])$
READ	op	c'(op) $+ l$(input)
WRITE	op	c(op)
JGTZ	et	$l(R[0])$
JZERO	et	$l(R[0])$
JUMP	et	1
HALT		1

dove c(op) e c'(op) dipendono dall'operando op; infatti:

$c(\sharp n) = l(n)$,
$c(i) = l(i) + l(R[i])$,
$c((i)) = l(i) + l(R[i]) + l(R[R[i]])$;

$c'(i) = l(i)$,
$c'((i)) = l(i) + l(R[i])$.

Nota. Per la moltiplicazione l'assunzione di costo logaritmico è discutibile. Infatti, ad oggi, non è noto alcun metodo che consenta di eseguire la moltiplicazione di due interi di n bit con un costo computazionale $o(n \log n)$.

Valutiamo il programma dell'esempio (ultima colonna=numero di volte che l'istruzione è eseguita):

	READ	1	$l(1)+l(x)$	$O(\log x)$	1
	READ	2	$l(2)+l(y)$	$O(\log y)$	1
	LOAD	♯1	$l(1)$	$O(1)$	1
	STORE	3	$l(3)+l(1)$	$O(1)$	1
5	LOAD	2	$l(2)+l(y)$	$O(\log y)$	$y+1$
	JZERO	14	$l(y)$	$O(\log y)$	$y+1$
	LOAD	1	$l(1)+l(x)$	$O(\log x)$	y
	MULT	3	$l(3)+l(x)+l(x^y)$	$O(\log x + y \log x)$	y
	STORE	3	$l(3)+l(x^y)$	$O(y \log x)$	y
	LOAD	2	$l(2)+l(y)$	$O(\log y)$	y
	SUB	♯1	$l(1)+l(y)$	$O(\log y)$	y
	STORE	2	$l(2)+l(y)$	$O(\log y)$	y
	JUMP	5	1	$O(1)$	y
14	WRITE	3	$l(3)+l(x^y)$	$O(y \log x)$	1
	HALT	1	$O(1)$	1	

In totale, $O(y^2 \log x)$ è il costo del programma.

Macchine a registri e Macchine di Turing

Le macchine a registri possono calcolare funzioni numeriche?

Definizione 1 *Una funzione* $f : \mathcal{N}^n \longmapsto \mathcal{N}$ è calcolabile *con macchine a registri se esiste un programma* P *per* RAM *tale che, se* $f(x_1,\ldots,x_n) = y$ *allora il programma, inizializzato con* $x_1,\ldots,x_n$ *sul nastro di input, termina con* y *sul nastro di output; se* $f(x_1,\ldots,x_n)$ *non è definita allora il programma non termina.*

Quali sono le funzioni calcolabili da una macchina a registri, e che rapporto c'è con le funzioni calcolabili da una macchina di Turing?

Teorema 2 *Data un macchina di Turing $\mathcal{M}$ con nastro semiinfinito e con alfabeto di nastro $\Gamma = \{0,1\}$,*
esiste una RAM con programma P tale che
se $\mathcal{M}$ realizza la transizione dalla configurazione iniziale q_0x alla configurazione finale q_Fy e
se la RAM è inizializzata con la stringa x nei registri $2,\ldots,|x|+1$,
alla fine della computazione la RAM avrá nei registri $2,\ldots,|y|+1$ la stringa y.
Se la macchina $\mathcal{M}$ opera in tempo T, allora la RAM opera in tempo $O(T \log T)$ (nel modello a costi logaritmici).

Quindi le RAM possono calcolare (via simulazione) tutte le funzioni calcolabili secondo Turing.

Teorema 3 *Data una macchina a registri con programma P che calcola la funzione f,*
esiste una macchina di Turing $\mathcal{M}$ tale che
se $f(x_1,\ldots,x_n) = y$ e
sul nastro di input sono memorizzati in binario gli interi $x_1,\ldots,x_n$,
la macchina $\mathcal{M}$ termina con la raappresentazione binaria di y sul nastro di output; se $f(x_1,\ldots,x_n)$ non è definita la macchina $\mathcal{M}$ non termina in una configurazione finale.
Se la RAM ha costo T (nel modello a costi logaritmici), allora $\mathcal{M}$ opera in tempo $O(T^2)$.

Quindi le macchine di Turing possono simulare il comportamento delle RAM.

I due teoremi dimostrano che ogni funzioni calcolabile secondo Turing è anche calcolabile tramite macchine a registri, e viceversa

Questo è un ulteriore indizio a conferma della tesi di Church-Turing.

Un problema è ritenuto **trattabile** se è risolubile in tempo polinomiale. Abbiamo visto che MDT e RAM a costi logaritmici si possono reciprocamente simulare con costi polinomiali.

Allora i due modelli sono interscambiabili per determinare se un problema è trattabile.

Tesi di Cobham. Tutti i modelli di calcolo introdotti per definire la "trattabilità computazionale" (calcolabilità in tempo polinomiale) hanno al più lo stesso potere computazionale delle MT operanti in tempo polinomiale.

Cosa accade se usiamo RAM a costi uniformi?

Teorema 4 *Nel modello a costi uniformi una RAM con l'operazione di moltiplicazione può calcolare in tempo polinomiale una funzione che richiede tempo esponenziale per essere calcolata da una macchina di Turing.*

Per dimostrarlo sufficiente considerare il calcolo della funzione $f(n) = 2^{2^n}$ su una RAM a costi uniformi dotate di operatore di moltiplicazione e su macchine di Turing.

Quindi, le RAM a costi uniformi sono un modello di calcolo più potente delle Macchine di Turing, nel senso che esse sono in grado di eseguire in tempo polinomiale calcoli che richiederebbero tempo esponenziale con una macchina di Turing.