Espressioni e Comandi

Luca Abeni

March 24, 2017

Elementi di Base dei Programmi

- Ricordate? Macchina Astratta:
  - Insieme di algoritmi e strutture dati che permettono di memorizzare ed eseguire programmi
- Abbiamo parlato di gestione della memoria (memorizzazione programmi e dati su cui operano)...

- ...Vediamo ora come sono composti i programmi!
- Elementi di base: espressioni e comandi
  - Differenza talvolta subdola...
  - Espressioni: ritornano un valore
  - Comandi: possono ritornare un valore, ma la cosa importante sono gli effetti collaterali!

Espressioni

- Espressione: *entità sintattica la cui valutazione produce un valore oppure non termina (espressione indefinita)*
  - Può avere o meno effetti collaterali...
  - ...Ma questi non sono l'aspetto fondamentale!
- Effetto collaterale: azione che influenza i risultati della computazione, ma senza restituire valori
- Siamo abituati ad espressioni matematiche...
  - Che però non hanno effetti collaterali!

Espressioni - Sintassi

- Espressione: composta da valori su cui operano degli operatori
  - Valori: espressi come costanti, variabili, etc...
  - Operatori: ha sotto-espressioni (anche valori semplici) come argomenti
- Come si combinano gli operatori ed i loro argomenti?
- Varie possibili sintassi
  - Infissa (Expr. matematiche cui siamo abituati)
  - Prefissa (anche “polacca di Cambridge”)
  - Postfissa (RPL)
- Assegnare un “significato” alle espressioni può essere più complesso (specie per notazione infissa)

Notazione Infissa

- Esempi: $a + b$, $a + b * c$, $a + b * c * *d * *e/f$, ...
  - Intuitiva per noi, ma può essere ambigua...
  - Precedenza fra operatori?
- Talvolta, regole di precedenza non intuitive:
  if A $<$ B **and** C $<$ D **then**
- Ogni linguaggio deve definire **chiaramente** le regole di precedenza
  - In caso di incertezza, parentesi!
- Pascal: regole di precedenza poco intuitive per operatori logici
- Smalltalk, APL: no regole di precedenza (valutazione da dx a sx)

Problemi con Notazione Infissa

- $15 - 4 - 3$: ci aspettiamo $(15 - 4) - 3$...
  - Ma alcuni linguaggi fanno $15 - (4 - 3)$!
- 5^{2^3}: ci aspettiamo $5^{(2^3)}$...
  - Ma ancora questo dipende dal linguaggio!!!
- Morale: scrivere sempre $(15 - 4) - 3$, $5 * *(2 * *3)$, etc...
  - Assumendo "**" come elevamento a potenza
- Ci sembra notazione semplice perché siamo abituati...
- ...Ma è tutt'altro che semplice ed intuitiva!
  - Molto difficile anche da valutare...

Notazione Postfissa

- Esempi: $a\ b\ +$, $a\ b\ +\ c\ d\ +\ *$, ...
- Prima operandi poi operatore
  - Sembra meno intuitiva, ma alla fine è più semplice!
  - Non servono parentesi
  - Non servono regole di associatività
  - Non servono regole di precedenza
- Valutazione mooooooolto più semplice
  - Basata su stack
  - Operando $\rightarrow$ pusha su stack
  - Operatore $\rightarrow$ preleva operandi da stack, esegui operazione e pusha risultato su stack
    - No operandi su stack? Errore sintattico (espressione non corretta)!

Semantica delle Espressioni

- Una volta definita la sintassi, va definita la semantica
  - Sappiamo come combinare valori ed operatori...
  - ...Ma, come si valuta l'espressione?
- Notazione infissa: vanno definite precedenza fra operatori, regole di associatività, etc...
- Indipendentemente da questo: ordine di valutazione?
  - Esempio: $(a + f(b)) * (c + f(b))$
  - Prima $a + f(b)$, poi $c + f(b)$ e poi si moltiplica?
  - Prima $c + f(b)$, poi $a + f(b)$ e poi si moltiplica?
  - $a + f(b)$ e $c + f(b)$ in parallelo e poi si moltiplica?

Ordine di Valutazione

$$(a + f(b)) * (c + f(b))$$

- L'ordine di valutazione cambia il risultato?
- Dal punto di vista matematico, no...
- Ma dal punto di vista pratico / informatico / implementativo?
- Bisogna tenere conto di:
  - Effetti collaterali
  - Aritmetica finita
  - Operandi non definiti
  - Ottimizzazioni del compilatore / interprete

Effetti Collaterali - 1

- Effetto collaterale: azione non valutabile ad un valore, ma che influenza i risultati della computazione / del programma
  - Non presente in funzioni matematiche “pure”...
  - ...Ma possibile in subroutine / funzioni / espressioni di un programma!
- Esempio:

$$(a + f(b)) * (c + f(b))$$

  - Si valuta prima $a + f(b)$ o $c + f(b)$?
  - Se $f(b)$ è valutata sempre allo stesso valore, ordine indifferente...

Effetti Collaterali - 2

```
int acc;
int f(int n)
{
  acc += n;

  return acc;
}
```

- La prima volta che viene invocata, $f(b)$ ritorna b, la seconda volta ritorna $b + b$, la terza $b + b + b$, etc...
- Se si valuta prima $a + f(b)$, si ottiene $(a + b) * (c + 2b)$
- Se si valuta prima $c + f(b)$, si ottiene $(a + 2b) * (c + b)$

Aritmetica Finita

- I numeri sono rappresentati su un numero finito di bit...
  - Possibili overflow / underflow, etc...
  - Esempio: $\frac{a*b}{c}$
- a `*` b `/` c valutata come $(a * b)/c$, come $a * (b/c)$, ...
  - Se `a` e `b` contengono valori grandi, a `*` b può causare overflow, quindi facendo prima il prodotto si ottiene un risultato errato
  - Se `b` contiene un valore molto piccolo e `c` contiene un valore grande, b `/` c può causare underflow...
- In questi casi, l'ordine di valutazione cambia il risultato!

Operandi non Definiti

- Operatore con n operandi...
- n sotto-espressioni, di cui una può essere non calcolabile / indefinita
  - Esempio: divisione per 0
- a != 0 ? b / a : b $\rightarrow$ if aritmetico: se $a \neq 0$, l'espressione è valutata a b/a, altrimenti a b
  - Se si valuta prima a != 0, poi si valuta b/a solo se il predicato è vero... No errori!
  - Se si valutano sempre tutte e 3 le sottoespressioni, si può avere divisione per 0!
    - Valutazione lazy vs eager
- Altro esempio: valutazione predicati con corto circuito

Ottimizzazioni

- Le espressioni sono tradotte in istruzioni macchina...
- ...Ed il loro ordine di esecuzione può avere effetti sulle performance!

```
v = mem[i];
res = v * v + x * y;
```

- Valutando x * y prima di a * a, prestazioni migliori!
  - Latenza della memoria: il valore di `v` potrebbe non essere subito disponibile!
  - Generalmente i compilatore si occupa di stabilire l'ordine di valutazione in modo da evitare stalli del pipeline della CPU

Comandi

- Comando: *entità sintattica la cui esecuzione non necessariamente restituisce un valore, ma può avere effetti collaterali*
  - Può restituire o meno un valore...
  - ...Ma questo non è l'aspetto fondamentale!
- Effetto collaterale: influenza i risultati della computazione senza restituire valori
  - Comandi senza effetti collaterali sono abbastanza inutili...
  - Gli effetti colaterali sono il motivo per cui eseguiamo un comando!
- Differenze da espressioni: meno marcate che una volta

Effetti Collaterali dei Comandi

- Tutto ciò che non è legato al valore ritornato
- Modifica dello stato del programma
  - Modifica del valore memorizzato in una variabile...
  - Ma anche Input/Output!
  - ...
- Tipici del paradigma di computazione imperativo

Assegnamento e Variabili

- Tipico esempio di comando: assegnamento
  - Modifica il contenuto di una variabile
  - Presuppone il concetto di *variabili modificabili*
    - Ancora, tipiche del paradigma imperativo!
- Variabili “come le conosciamo”
  - Entità denotabili (a cui possiamo associare un nome)
  - Possono “contenere” un valore (sono locazioni di memoria)
- Ma esistono anche altri tipi di variabili

Variabili - 1

- Nei linguaggi imperativi
  - Variabili modificabili
  - Aree di memoria associate ad un nome, che possono contenere un valore memorizzabile
  - Doppio mapping: nome $\rightarrow$ variabile (entità denotabile) e variabile $\rightarrow$ valore memorizzato
- Nei linguaggi orientati agli oggetti
  - Variabili con *modello a riferimento*
  - Riferimenti (l-value) ad oggetti memorizzati nello heap
  - Assegnamento: non modifica l'oggetto riferito, ma cambia il riferimento
- Nei linguaggi funzionali, no variabili!

Variabili - 2

- “No variabili” (linguaggi funzionali) è uno statement forte...
- Meglio dire “no variabili modificabili”
- Solo legami (binding) fra nomi e valori denotabili (ambiente - environment)
- In pratica:
  - Paradigma imperativo: due funzioni *ambiente* e *memoria*
  - Paradigma funzionale: solo *ambiente*

Assegnamento - 1

- Sintassi: `<exp1> = <exp2>`
  - `<exp1>` valutata ad un *l-value*: riferimento che denota una variabile
  - `<exp2>` valutata ad un *r-value*: valore memorizzabile
- Esempio: `x = 1`
  - Cambia il contenuto della variabile identificata dal nome "`x`", memorizzandoci il valore `1`
- Esempio: `x = x + 1`
  - I simboli "`x`" ed "`x`" hanno significati diversi!
  - "`x`": variabile identificata dal nome "`x`"
  - "`x`": valore memorizzato nella variabile identificata dal nome "`x`"

Assegnamento - 2

`<exp1>` = `<exp2>`

1. Valuta `<exp1>`
   - Il risultato deve essere un l-value (riferimento)
   - Cose tipo "`x + 1 = 2`" non hanno senso!
2. Identifica la variabile riferita da tale l-value (usando l'ambiente)
3. Valuta `<exp2>`
   - Il risultato deve essere un r-value (valore memorizzabile)
   - Cose tipo "`x = double`" non hanno senso!
4. Memorizza tale r-value nella variabile identificata al punto 2 (modificando la funzione memoria)