Linda

Walter Cazzola

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano

Outline

Linda
Nozioni Generali e Caratteristiche.

Linda è un linguaggio di coordinamento che estende i linguaggi tradizionali permettendone l'utilizzo nello sviluppo di applicazioni in ambiente distribuito.

Linda è indipendente dall'architettura sottostante (a memoria condivisa o rete di calcolatori) e dal linguaggio sequenziale usato (C, C++, etc ...).

Linda implementa una *memoria associativa* logicamente condivisa da tutti i processi dell'applicazione, detta Spazio delle Tuple.

Spazio delle Tuple
Nozioni Generali e Caratteristiche.

Lo spazio delle tuple è un'area di memoria associativa condivisa da tutti i processi che compongono il sistema.

Si parla di memoria associativa perché le tuple vengono identificate tramite matching su una chiave piuttosto che utilizzare un meccanismo tradizionale di indirizzamento.

Una tupla è una sequenza di campi con tipo:

```
("linda", 2, 3.14)
(sqrt(3.14), #FF3D0B,'t')
(0,'a')
```

Ogni campo può contenere un valore di un qualsiasi tipo (semplice o strutturato) ammesso dal linguaggio sequenziale utilizzato.

Spazio delle Tuple
Operazioni sulle Tuple.

In linda sono definite 6 operazioni che lavorano nello spazio delle tuple:

`out`($\bar{t}$)	`eval`($\bar{t}$)	Generano nuove tuple
`in`($\bar{s}$)	`inp`($\bar{s}$)	Eliminano tuple dallo spazio delle tuple
`rd`($\bar{s}$)	`rdp`($\bar{s}$)	Leggono tuple dallo spazio delle tuple

Notare che:

- il parametro $\bar{t}$ di `out`($\bar{t}$) e di `eval`($\bar{t}$) è una tupla; mentre
- il parametro $\bar{s}$ di `in`($\bar{s}$), `inp`($\bar{s}$), `rd`($\bar{s}$) e `rdp`($\bar{s}$) è un'anti-tupla (o template).

Un'anti-tupla è una sequenza di campi tipati che possono essere o dei campi valore (parametro attuale) o un campo indefinito (parametro formale) introdotto da ?.

```
(?radix, a string'', ?j, 100)
```


Spazio delle Tuple
Operazioni sulle Tuple: Regole di Matching.

Si ha un matching tra una tupla $\bar{t}$ ed un'anti-tupla $\bar{s}$ se:

- il numero di campi di $\bar{t}$ e di $\bar{s}$ è lo stesso;
- si ha una corrispondenza tra i tipi di ogni campo di $\bar{s}$ e di $\bar{t}$; e
- i campi valore di $\bar{s}$ sono uguali ai campi valore di $\bar{t}$

Ad esempio, le seguenti tuple:

```
t̄ ≡ (?radix, "a string", ?j)
s̄ ≡ (sqrt(2), "a string", 100)
```

danno origine ad un matching, perché

- il numero di elementi $\bar{t}$ e $\bar{s}$ è lo stesso,
- l'unico campo valore corrisponde in entrambe le tuple, e
- le variabili dell'antitupla $\bar{s}$ hanno tipo compatibile con corrispondenti valori della tupla $\bar{t}$.

Spazio delle Tuple
Operazioni sulle Tuple: Generazione delle Tuple.

`out`($\bar{t}$):
la tupla $\bar{t}$ viene valutata all'interno del processo invocante e aggiunta allo spazio delle tuple; il processo che la ha generata continua la propria esecuzione.

`eval`($\bar{t}$):
crea implicitamente un nuovo processo (tupla attiva) per valutare ogni campo di $\bar{t}$; terminata la valutazione di tutti i campi di $\bar{t}$, $\bar{t}$ viene depositata nello spazio delle tuple.

Spazio delle Tuple
Operazioni sulle Tuple: Eliminazione delle Tuple.

`in`($\bar{s}$):
una tupla $\bar{t}$, presente nello spazio delle tuple, che si accoppia con l'anti-tupla $\bar{s}$ viene eliminata.

Ai campi formali dell'anti-tupla $\bar{s}$ vengono associati i valori attuali della tupla $\bar{t}$.

Se al momento della chiamata nello spazio delle tuple non esiste nessuna tupla che possa essere accoppiata all'anti-tupla $\bar{s}$, il processo viene sospeso. Se esistono più tuple ne viene scelta una in modo arbitrario.

`inp`($\bar{s}$):
versione non bloccante di `in`(). Se esiste una tupla $\bar{t}$ che si accoppia con $\bar{s}$, `inp`() si comporta con `in`() e ritorna 1. Altrimenti termina immediatamente e ritorna 0.

Spazio delle Tuple

Operazioni sulle Tuple: Lettura delle Tuple.

`rd(s̄)`:
ha lo stesso comportamento di `in()` con la differenza che la tupla $\bar{t}$ selezionata non viene eliminata dallo spazio delle tuple.

`rdp(s̄)`: versione non bloccante di `rd()`.

Spazio delle Tuple

Tuple Attive e Passive.

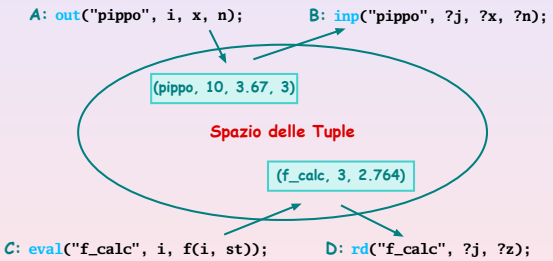
Lo spazio delle tuple può contenere: tuple attive e passive.

Tuple attive
Vengono valutate (eseguite) concorrentemente e si scambiano dati generando, leggendo e consumando tuple passive. Una tupla attiva, quando termina la sua esecuzione diventa una tupla passiva.

Tuple passive
Dati riferiti associativamente dalle tuple attive.

Le tuple possono essere generate, lette o consumate, ma mai modificate. Questa proprietà semplifica l'implementazione dello spazio delle tuple.

Spazio delle Tuple



Cooperazione in Linda

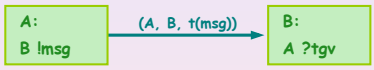
Utilizzando le sei operazioni primitive di linda è possibile emulare, attraverso lo spazio delle tuple, meccanismi di cooperazione tipici sia del modello a memoria distribuita che condivisa.

- Semplicità di sviluppo e porting delle applicazioni;
- portabilità su piattaforme hardware differenti.

Programmazione in Linda

Scambio di Messaggi.

Comunicazione Simmetrica Sincrona.
Il mittente del messaggio rimane bloccato fintanto che non viene effettivamente ricevuto dal destinatario.



Soluzione:

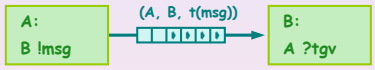
```
A:  out(msg, "A", "B");
    in("ack");

B:  in(?vtg, "A", "B");
    out("ack");
```


Programmazione in Linda

Scambio di Messaggi.

Comunicazione Simmetrica Asincrona.
Il mittente manda il messaggio al destinatario senza attendere conferma che sia stato ricevuto.



Soluzione:

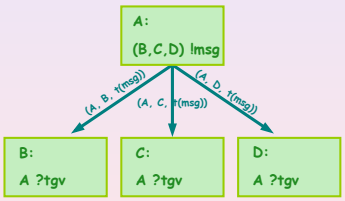
```
A:  out(msg, "A", "B");

B:  in(?vtg, "A", "B");
```


Programmazione in Linda

Scambio di Messaggi.

Comunicazione Asimmetrica Sincrona 1 a n.
Il mittente manda il messaggio a più destinatari ed attende conferma che tutti i destinatari lo abbiano ricevuto.



Soluzione:

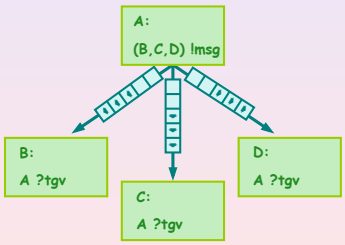
```
A:  out(msg, "A", n);
    in("ack");
    in(msg, "A", 0);

B:  in(?vtg, "A", ?m);
    out(vtg, "A", m-1);
    rd(?vtg, "A", 0);
    if (m == 1) out("ack");
```


Programmazione in Linda

Scambio di Messaggi.

Comunicazione Asimmetrica Asincrona 1 a n.
Il mittente manda il messaggio a più destinatari senza attendere conferma che tutti i destinatari lo hanno ricevuto.



Soluzione:

```
A:    out(msg, "A");

B,C,D: rd(?vtg, "A");
```

Programmazione in Linda
Sincronizzazione.

Semafori.

```
out("semaphore");          ; inizializzazione del semaforo
in("semaphore");           ; acquisizione del semaforo (P)
out("semaphore");          ; rilascio del semaforo (V)
```

es. processo che usa un semaforo

```
A:  in("semaphore");
      «sezione critica»
    out("semaphore");
```

Barriere.

Le barriere sono utilizzate per sincronizzare più processi in modo che riprendano la propria computazione simultaneamente.

```
A: out("barrier", n-1);          Other: in("barrier", ?val);
   rd("barrier",0);                     out("barrier", val-1);
                                        rd("barrier", 0);
```


Programmazione in Linda
Remote Procedure Call.

```
A:    eval("Server S", ServerS());

      ServerS() {
        int i, id;
        int s_calc(int j) { ... }
        while (true) {
          in("request", ?id, ?i);
          out("reply", id, s_calc(i));
        }

B:    int rpc_s(int i) {
        static int id = 0;
        out("request", id, i);
        in("reply", id++, ?i);
        return i;
      }
```


Programmazione in Linda
Programmazione Parallela.

Parallelizzazione di un loop con interazioni indipendenti:

```
for (int i=0; i<=n; i++)
  eval("this loop", something(i));

for (int i=0; i<=n;i++)
  in("this loop", ?v);
```

Array multidimensionali condivisi:

```
("Array Name", index1, ..., indexn, val)
```


Programmazione in Linda
Strutture Dati Dinamiche Condivise.

Liste.

```
("list", "head", i)              ; Puntatore alla testa della lista
("list", i, element(i))          ; Elemento i-esimo della lista
("list", "tail", n)              ; Puntatore alla coda della lista
```

Inserire un elemento in coda alla lista:

```
in("list", "tail", ?index)          ; recupera la tupla della coda
out("list", "tail", index+1)        ; def. idx+1 come nuova coda
out("list", index+1, new_element)   ; inserisce il nuovo elemento
```

Cancellare un elemento dalla testa della lista:

```
in("list", "head", ?index)          ; recupera la tupla della testa
out("list", "head", index+1)        ; definisce idx+1 come nuova testa
in("list", index, ?element)         ; rimuove la vecchia testa
```

Cercare un valore all'interno della lista:

```
inp("list", ?index, val)
```


Programmazione in Linda
Strutture Dati Vive.

Ogni elemento della struttura dati è incapsulato in un processo il cui compito consiste nel calcolare tale elemento.

```
eval("Object A", compute_object(arg));
```

Effettuato il calcolo il processo (tupla attiva) termina rendendo disponibile il valore calcolato nello spazio delle tuple (tupla passiva).

```
rd("Object A", ?var);
```


Programmazione in Linda
Es. Crivello di Eratostene.

```
main() {
  int ok, limit;
  for (int i=2;i<=limit;i++) eval("primes", i, is_prime(i));
  for (int i=2; i<=limit; i++) {
      rd("primes", i, ?ok);
      if (ok) printf("%d\n", i);
  }
}

int is_prime(int me) {
  int i, limiti, ok;
  limit = sqrt((double)me)+1;
  for (i=2;i<limit;i++) {
     rd("primes", i, ?ok);
     if (ok && (me%i == 0)) return 0;
  }
  return 1;
}
```


Linda
Note Implementative.

Le implementazioni richiedono efficienza nel pattern matching all'interno dello spazio delle tuple.

Preprocessing

A: out("x", i); B: in("x", ?i);

x è costante, il compilatore può ottimizzare collegando direttamente (variabile condivisa o scambio di messaggi) i 2 processi.

Restringere la ricerca su una chiave singola

Se le anti-tuple hanno una sola variabile al loro interno, si può adottare l'hashing per la loro memorizzazione, ottenendo l'accesso ad una tupla in tempo costante.

Spazi delle tuple multipli

lo spazio delle tuple non è centralizzato, la quantità di tuple da controllare diminuisce aumentando l'efficienza.

Riferimenti Bibliografici

- Nicholas Carriero and David Gelernter.
  Linda in Context.
  Communications of the ACM, 32(4):444–458, April 1989.
- Nicholas Carriero, David Gelernter, and Jerrold Leichter.
  Distributed Data Structures in Linda.
  In *Proceedings of the 13th ACM Symposium on Principles of Programming Languages*, pages 236–242, St. Petersburg Beach, Florida, USA, January 1986.
- David Gelernter, Nicholas Carriero, Sarat Chandran, and Silva Chang.
  Parallel Programming in Linda.
  In *Proceedings of the International Conference on Parallel Processing*, pages 255–263, St. Charles, August 1985. IEEE.