Lezione 18-19

March 26-28, 2012

Tipi di Dati e Tipi del Linguaggio

- Terminologia: Tipi (o classi) di dato e Tipi del linguaggio
- Tipi di Dato: Proprietà Generali
- Scalari
- Strutturati
- Sistemi di deallocazione di memoria dinamica
- Tipi del linguaggio
- Sistemi di Tipi e Type Safety
- Coercion e Cast
- Polimorfismo: ad Hoc, Generico e di Sottotipo

Terminologia: Tipi (o classi) di dato e Tipi del linguaggio

- Dati: Le strutture più semplici con cui esprimere valori
- Tipi di Dato: Collezioni di valori **omogenei per operazioni a loro applicabili**
- Tipi del Linguaggio: Strutture che **classificano** (in modo univoco tutte) le strutture presenti in un prgramma
  - evidenziando l'uso di ciascuna struttura da un punto di vista semantico
  - forniscono una semantica statica del programma

Example

Non solo ai dati è associato un tipo.

Ad ogni dato può essere associato un tipo ma non il contrario.

Dati: Proprietà Generali/1

- Dati: Le strutture più semplici con cui esprimere valori
  - Anche se non essenziali per la calcolabilità (il lambda calcolo tipato è Turing Completo): Aumentano l'espressività del linguaggio
- Literal: Possono avere una presentazione esterna oppure no:
  - espressi direttamente come Literal (ad es.: interi in tutti i linguaggi)
  - solo calcolati (ad es.: variabili del linguaggio a 3 indirizzi, puntatori in Pascal, array di valori struct in C)

Dati: Proprietà Generali/2

- In un linguaggio sono distinti (in accordo all'uso fatto) in:
  - valori Denotabili (dichiarati come cotanti)
  - valori Esprimibili (calcolati da espressioni non variabile)
  - valori Memorizzabili (associati a locazione di memoria [oggetti di Java] o a valori modificabili [C e gli imperativi, in generale])
- Un'altra distinzione:
  - scalari: sono atomici: solo operazioni per calcolare altre valori
  - strutturati: hanno anche operazioni per visitare e/o modificare i componenti

Dati: Proprietà Generali/3

- Distinzione rispetto alla Modificabilità
  - Gli scalari non sono modificabili
    - sebbene una variabile di un tipo di dato scalare possa avere il valore associato modicato più volte
  - gli strutturati in genere sono modificabili nei componenti
    - in aggiunta, una variabile di un tipo di dato strutturato può vedere il valore associato sostituito da altri valori.

Dati: Proprietà Generali/4

- Distinzione rispetto alla Allocazione di
  - Memoria Statica a compile/loading time: {...int A[100];...}
  - Memoria Statica a run time: {...int A[x*y+10];....}
  - Memoria Dinamica: {...u=malloc((x*y+10)*sizeof(int));...}

- Ma dove sta questa memoria e quando è deallocata?
- In principio, abbiamo visto che il *Run Time Support* o la *Macchina Virtuale* o quella *Astratta* prevedono:
  - Dinamica. Allocata nello Heap e deallocata in vari modi
  - Statica. Allocata nella Memoria Statica e deallocata a fine programma

Dati: Proprietà Generali/5

- Memoria Dinamica è allocata nello Heap
  - deallocata sotto il controllo del programma (dispose, free)
  - deallocata automaticamente dal garbage collector
- Memoria statica
  - Alcune implementazioni. Allocata nello AR nello stack di AR e recuperata quando lo AR è popped.
    - Solo se il linguaggio ha ciclo di vita dei valori memorizzabili è uguale a quello dei bindings.
  - Oggi nessun linguaggio adotta tale sistema per la memoria statica.
  - Utilizzato per la memoria temporanea: valori intermedi - stack

Dati Scalari e Derivati

- Sono Valori Atomici
- Dati Scalari: int, real, float, double, bit, char, boolean
  - Hanno presentazione esterna (I/O)
  - Hanno operazioni primitive associate per calcolare altri valori: Interi e le operazioni aritmetiche
  - Aiutano l'espressività nelle applicazioni che richiedono tali valori
- Dati derivati: enumerati [ad es. $type\ T = (\mathtt{I}_1, ..., \mathtt{I}_k)$], intervallo [$type\ T = \mathtt{V}_{inf}..\mathtt{V}_{sup}$]
  - hanno le operazioni del dato originale
  - Migliorano il controllo degli indici per la visita di dati strutturati indiciati (array, vector)

Dati Strutturati

- Record (con varianti o union), Array (statici e dinamici), List, Set, Pointer
- Hanno operazioni e caratteristiche specifiche
- Aiutano l'espressività nelle applicazioni che richiedono valori strutturati
- Dati Concreti e Astratti: Aiutano l'espressività nelle rappresentazione di valori del problema non disponibili nel linguaggio (ad es. String può essere rappresentato con un array di char.

Dati Strutturati: Record (con varianti e union)

- struttura di valori non omogenei modificabili, accessibile attraverso field, campi, selettori
- sono prodotto cartesiano dei dati componenti
- campi sono(funzioni proiezione) proiezioni attraverso cui accedere i componenti
- record con varianti o union pericolosi meccanismi di basso livello

Example

```
type enum = (a,b);
var ex : record c1:integer;
                    case u:enum of a:(a1:array[1..5] of integer);
                                   b:(b1:array[1..10] of char);
             end;
... ex.u:= a; ...ex.b1[5]....
```

Record: Formalizzazione

- Ambiente delle sue proiezioni

Table15.1 – Variabile Record

Funzioni Semantiche

$$\mathcal{D}[\![\mathtt{D}]\!]_\rho : (\mathtt{Env} \times \mathtt{Store}) \to (\mathtt{Env} \times \mathtt{Store})_\perp$$

$$\mathcal{D}[\![\mathtt{I}\ \mathtt{:Rec}\ \mathtt{I_1}\mathtt{:}\ \mathtt{T_1} \ldots \mathtt{I}_n\mathtt{:}\ \mathtt{T_n}\ \mathtt{End}]\!]_\rho(\mathtt{s}) =$$

$$\mathtt{Let}\{\mathtt{s_0} = \mathtt{s}, \forall \mathtt{T_i}\,.(\mathtt{l_i}, \mathtt{s_i}) = \mathit{allocate}(\mathtt{T_i}, \mathtt{s_{i-1}}\}$$

$$\{\forall \mathtt{I_i}\,.\gamma(\mathtt{I_i}) = \mathtt{l_i}\}$$

$$(\mathtt{bind}(\mathtt{I}, \mathtt{E}(\gamma), \rho), \mathtt{s_n})$$

$$\mathcal{E}[\![\mathtt{E}]\!]_\rho : \mathtt{Env} \to \mathtt{State} \to \mathtt{State}_\perp$$

$$\mathcal{E}[\![\mathtt{Den}(\mathtt{I}\,.\,\mathtt{I_1})]\!]_\rho = \mathtt{Let}\{\mathtt{E}(\gamma) = \rho(\mathtt{I})\}\gamma(\mathtt{I_1})$$

$$\mathcal{E}[\![\mathtt{Val}(\mathtt{I}\,.\,\mathtt{I_1})]\!]_\rho = \ldots$$

Funzioni Ausiliarie e Domini Ausiliari

$$\mathtt{E} : \mathtt{EnvL} \to \mathtt{Den}$$

$$\mathtt{Den} ::= \ldots + \mathtt{EnvL}$$

$$\mathtt{EnvL} ::= \mathtt{I} \to \mathtt{Den}$$

Record: Implementazione

- Diverse tra loro
- Dipendono:
  - dal linguaggio
  - dal contesto di implemetazione: RTS, M. Virtuale, M. Astratta
- Tutte realizzate con un codice per l'accesso in memoria dei componenti.

Dati Strutturati: Array (statici e dinamici)

- struttura dati di base dei Linguaggi Imperativi
- struttura di valori omogenei modificabili e indiciata
- sono funzioni finite su domini (con operatore successore)
- controllo dei bounds degli indici
- dimensione fissa: allocati a compile time o run time in Memoria Statica
- dimensione variabile: allocati a run time nello heap.
- può avere struttura determinata a run time (dope vector - controlla gli outbounds)

Dati Strutturati: Insiemi

- Presente in Pascal like (imperativo) e in SETL (applicativo)
- struttura non di valori omogenei non modificabili
- nessun accesso ai componenti
- sono insiemi finiti su tipi di dato

Example

type T = Set of T_1;
var T x, y, z;
T_1 u;
begin...$x := [e_1, ..., e_k]$; ...; $y := x + [e'_1, ..., e'_n]$; ...; $z := x - y$....
if u in x then...else...; ....

- operazioni su insiemi
- dimensione variabile: allocata a run time su heap

Dati Strutturati: Liste

- struttura dati di base dei Linguaggi Funzionali
- struttura ordinata di valori omogenei non modificabili
- operazioni per accesso ai componenti (car, cdr)
- costruzione di nuove liste (empty, cons, append)
- dimensione variabile: allocata a run time su heap

Dati Strutturati: Puntatori/1

- struttura dati modificabile ad un solo componente
- struttura dati per usare memoria in modo diretto
- struttura di basso livello: uso diretto di "strutture semantiche"
  - Uso controllato in Pascal like:
    - introdotti attraverso `new` (trasparenti)
    - operano in memoria separata da quella statica
    - deallocazione a programma `dispose` e automatica
  - Uso controllato in C like:
    - introdotti attraverso `malloc` (trasparenti)
    - ed anche: operatore & e "aritmetica" (non trasparanti)
    - operano in memoria non separata da quella statica (&)
    - deallocazione a programma `free`
- Un unica operazione: accesso al valore "puntato"
- dimensione variabile: allocata a run time su heap (C anche statica)

Dati Strutturati: Deallocazione di Memoria/1

- Automatica e Trasparente: Haskell, Caml
  - Le liste non accessibili dal reductum (del programma) sono deallocate (garbage collector)
- A programma con operatori
- usata con puntatori
- Dangling Reference

Example

$$\{...int * x, *y;$$
$$...x = (int*)malloc(sizeof(int));$$
$$...y = x;\ *x\ *= 3; printf(*x, *y);$$
$$...free(x); printf(*y); ...$$

- Tombstone: accesso al valore associato attraverso Tombstone che non sono rimossi ma marcati se elemento rimosso (variante: puntatori con chiave e valori associato con lucchetti)

Dati Strutturati: Deallocazione Automatica di Memoria/2

Tecnica a contatore: Un contatore, $C(V)$ è inizializzato a 1 alla creazione della struttura "puntata" V(e.g. malloc...)

- Ad ogni operazione di "copia" (assegnamento, trasmissione parametri) con sorgente p_s e destinazione p_t:
  - $C(V_{p_s})++$, dove V_{p_s} sia la struttura associata a p_s;
  - $C(V_{p_t})--$, dove V_{p_s} sia la struttura associata a p_s;
- Ad ogni pop di un AR (uscita da blocco inline o procedura)
  - Si considerano tutte le variabili locali e per ciascuna i puntatori accessibili come valori o come componenti di valori. Siano essi $\{p_{s_i}\}$.
  - $C(V_{p_{s_i}})--$
  - Se $C(V_{p_{s_i}}) == 0$, la struttura $V_{p_{s_i}}$ è deallocata.

Tipi del Linguaggio

Example

```
int main (int argc, char *argv[]){//cosa calcola?
  int x = 10;
  char a =' b';
  x+ = a;
  printf("stampa%2d\n", x + a);
  return0;
  }
```

Gnu CC compila un oggetto la cui esecuzione termina correttamente producendo :

```
stampa 206
```

- Potremmo dire: Vabbuó si sa che C fà accussì

Tipi del Linguaggio/2

- Potremmo dire: Vabbuó si sa che C fà accussì
- E se la variabile "a" fosse stato il nome dell'operatore di volo e "x" l'angolo della rotta da tenere per superare la cresta della montagna? - Abbiamo ancora voglia di rispondere così (se la rotta è quella dell' aereo su cui stiamo viaggiando)?

La situazione peggiore nella quale puó incorrere l'esecuzione di un programma:

- non è che termini a causa di "floating point exception" o "illegal division by 0" o "array outbounds" (**errori trapped**)
- nemmeno che il programma non termini (il sistema operativo, e molti altri, lo devono fare)

La peggiore è che l'esecuzione raggiunga **stati** di calcolo del tutto **inattesi**

Tipi del Linguaggio/3

La peggiore è che l'esecuzione raggiunga **stati** di calcolo del tutto **inattesi**

- Lo scopo principale della presenza di una struttura dei tipi in un linguaggio è quella di riconoscere programmi che contengono codice che possa condurre a comportamenti inattesi
- ovvero, programmi che possono produrre **errori untrapped**, ovvero errori che non sono riconosciuti come tali e che per tale ragione non possono essere "intrappolati"
- codice che conduce a comportamenti inattesi è detto **codice stuck** o impantanato.

Sistemi di Tipi

Costituito da 3 strutture:

- il Dominio dei tipi includente i tipi base del linguaggio
- Le Espressioni di tipo anche per formare tipi nuovi
- Le Regole con le quali il linguaggio associa tipi ad ogni struttura dei programmi del linguaggio

Vediamo tutto questo nel sistema F1 per linguaggio Lambda Calcolo Tipato

- il sistema può essere utilizzato come base per i sistemi di tipi di linguaggi più complicati
- Ne vediamo l'estensione al sistema F2 per tipi polimorfi

Type Safety o Soundness

Un linguaggio può avere un sistema di tipi che garantisce che i programmi prodotti siano type safe (Haskell, Ocaml, Java)

- **Progress** L'esecuzione di un **programma well typed**, $[\vdash \mathtt{P} : \mathtt{T}]$, non è stuck:
  - o ha raggiunto uno stato finale e calcolato i valori dei tipi attesi, $[\mathtt{P} \in \mathtt{Val}]$;
  - o, raggiungere un nuovo stato, in accordo alle regole semantiche, $[\vdash \mathtt{P} \longrightarrow \mathtt{P}']$.
- **Preservation o Subject Reduction** Un **programma well typed**, $[\vdash \mathtt{P} : \mathtt{T}]$, mantiene i tipi delle sue strutture componenti per tutta l'esecuzione [se $\vdash \mathtt{P} \longrightarrow \mathtt{P}'$ allora, $\vdash \mathtt{P}' : \mathtt{T}$].

Sistema F1

Vediamo come è definito F1

Espressioni di tipo

Permettono di definire nuovi tipi per le strutture del programma in aggiunta ai tipi costruiti con prodottti, $\times$, somma, $+$, funzione, $\rightarrow$

- naming di tipi

Example

State = Env X Store
Env = Ide $\rightarrow$ Den
Store = Loc $\rightarrow$ Mem

- ricorsivi

Example

'a list = Cons 'a ('a list) | Nil

Relazioni tra tipi

- **equivalenza**
  - **nominale** Equivalenza solo con sè stesso
  - **strutturale** Stessa espressione di tipo, rimpiazzando nomi con definizioni
- **sottotipi**
  - esplicita (java) o
  - implicita (contra-covariance delle funzioni)
- **coercion**: Operazione di conversione di rappresentazione di valori e conseguente cambiamento di tipo
- **cast**: Vincolo di tipo che deve essere soddisfatto a run time
- **overloading** tipi e valori differrenti per uno stesso identificatore (di funzione)
- **polimorfismo di sottotipo** In combinazione con i sottotipi: un metodo si applica anche a valori di sottotipi dei tipi attesi.