Elementi di semantica denotazionale

Contenuti

- definizioni semantiche “eseguibili”
  - specificate in ML invece che nel tradizionale metalinguaggio
- semplice linguaggio funzionale con scoping statico
  - simile ad ML senza tipi
- sintassi astratta e domini sintattici
- semantica denotazionale
  - domini semantici: valori e stato
  - funzioni di valutazione semantica
- dalla semantica denotazionale a quella operazionale
  - modifiche nei domini e nelle funzioni di valutazione semantica

Sintassi e semantica

- un linguaggio di programmazione, come ogni sistema formale, possiede
  - una sintassi
    - le formule ben formate del linguaggio (o programmi sintatticamente corretti)
  - una semantica
    - che assegna un significato alle formule ben formate
    - dando un'interpretazione ai simboli
      - in termini di entità (matematiche) note
- la teoria dei linguaggi formali fornisce i formalismi di specifica (e le tecniche di analisi) per trattare gli aspetti sintattici
- per la semantica esistono diverse teorie
  - le più importanti sono la *semantica denotazionale* e la *semantica operazionale*
- la semantica formale viene di solito definita su una rappresentazione dei programmi in *sintassi astratta*
  - invece che sulla reale rappresentazione sintattica concreta

Sintassi astratta

- in sintassi concreta, i costrutti del linguaggio sono rappresentati come stringhe
  - è necessaria l'analisi sintattica per mettere in evidenza la struttura importante dal punto di vista semantico, risolvendo problemi legati solo alla comodità di notazione
    - proprietà degli operatori concreti (associatività, distributività, precedenze)
    - ambiguità apparenti (per esempio, x := x + 1)
- in sintassi astratta, ogni costrutto è un'espressione (o albero) descritto in termini di
  - applicazione di un operatore astratto (dotato di un tipo e di una arietà n) ad n operandi, che sono a loro volta espressioni (del tipo richiesto)
  - la rappresentazione di un programma in sintassi astratta è "isomorfa" all'albero sintattico costruito dall'analisi sintattica

Domini sintattici

- la sintassi astratta è definita specificando i *domini sintattici*
  - in ML definendo un insieme di tipi

```
# type ide = string;;
type ide = string
# type expr =   | Den of ide   |  Fun of ide * expr
  | Plus of expr * expr   | Apply of expr * expr;;
type expr =
  | Den of ide
  | Fun of ide * expr
  | Plus of expr * expr
  | Apply of expr * expr
```

Semantica denotazionale: domini

- la semantica denotazionale associa ad ogni costrutto sintattico la sua *denotazione*
  - una funzione che ha come dominio e codominio opportuni *domini semantici*
- i domini semantici vengono definiti da *equazioni di dominio*

  nome-dominio = espressione-di-dominio
- le espressioni di dominio sono composte utilizzando i *costruttori di dominio*
  - enumerazione di valori

    bool = { True, False } int = { 0, 1, 2,..... } festivo = {Sabato, Domenica}
  - somma di domini

    val = [int + bool] (un elemento appartiene a int oppure a bool)
  - iterazione finita di un dominio

    listval = val^* (un elemento è una sequenza finita o lista di elementi di val)
  - funzioni tra domini

    env = ide → val (un elemento è una funzione che mappa un elemento di ide in uno di val)
  - prodotto cartesiano di domini

    closure = expr * env (un elemento è una coppia formata da un elemento di expr ed uno di env)
- per ogni costruttore di dominio, esiste un metodo per definire l'ordinamento parziale del dominio, a partire da quelle dei domini utilizzati
  - garantendo che tutti i domini siano effettivamente *reticoli completi*

Domini semantici in ML

- i costruttori di dominio sono operazioni sui tipi
- enumerazione di valori
  - bool = { True, False } int = { 0, 1, 2,..... } festivo = {Sabato, Domenica}
  - bool e int sono tipi primitivi
  - `# type festivo = Sabato | Domenica`
- somma di domini
  - val = [int + bool] (un elemento appartiene a int oppure a bool)
  - `# type val = Int of int | Bool of bool | Unbound`
- iterazione finita di un dominio
  - listval = val^* (un elemento è una sequenza finita o lista di elementi di val)
  - `# type listval = val list`
- funzioni tra domini
  - env = ide → val (un elemento è una funzione che mappa un elemento di ide in uno di val)
  - `# type env = ide -> val`
- prodotto cartesiano di domini
  - closure = expr * env (un elemento è una coppia formata da un elemento di expr ed uno di env)
  - `# type closure = expr * env`

Domini semantici: lo stato

- in un qualunque linguaggio ad alto livello, lo stato deve comprendere un dominio chiamato *ambiente* (environment)
  - per modellare l'associazione tra gli identificatori ed i valori che questi possono denotare

```
type env = string -> val
let emptyenv = function y -> Unbound
let applyenv(x,y) = x y
let bind(r, l, e) =
          function lu -> if lu = l then e else applyenv(r,lu)
let rec bindlist(r, il, el) = match (il,el) with
         | ([],[]) -> r
         | i::il1, e::el1 -> bindlist (bind(r, i, e), il1, el1)
         | _ -> raise WrongBindlist
```

- per trattare linguaggi imperativi o orientati ad oggetti servono anche i domini store e heap

Domini semantici: i valori

- in un linguaggio funzionale, è quasi sempre sufficiente un solo dominio di valori (val, valori esprimibili), dominio dei valori che possono essere ottenuti come semantica di una espressione
  - i valori esprimibili contengono sempre le funzioni (che non abbiamo ancora visto)
  - tutti i valori esprimibili sono anche denotabili (da un nome nell'ambiente)

- nei linguaggi imperativi bisogna in generale distinguere tre tipi di valori
  - in aggiunta a queli esprimibili, ci sono quelli denotabili e memorizzabili

Il dominio semantico fun

```
# type expr =  | Den of ide  |  Fun of ide * expr
               | Plus of expr * expr  | Apply of expr * expr;;
```

- le espressioni contengono l'astrazione
  - `Fun(ide1, expr1)` rappresenta una funzione
    - il cui corpo è l'espressione `expr1`
    - con parametro formale `ide1`
- la semantica di tale costrutto è un valore del dominio `fun`
- `type fun = val -> val`
  - il primo valore di tipo `val` sarà il valore dell'argomento nella applicazione
    - il passaggio di parametri si effettua nell'ambiente
    - il linguaggio ha scoping statico
      - l'ambiente che ci sarà al momento della applicazione non influenza la valutazione

Il dominio semantico val

```
type val = Int of int | Bool of bool | Unbound
```

- dato che le funzioni sono valori esprimibili, dobbiamo aggiungerle al dominio `val`

- ```
  type fun = val -> val
  and val = Int of int | Bool of bool | Unbound
          |Efun of fun
  ```
  ```

La funzione di valutazione semantica

```
type expr =
  | Den of ide
  | Fun of ide * expr
  | Plus of expr * expr
  | Apply of expr * expr
```

- per ogni dominio sintattico esiste una funzione di valutazione semantica
  - nei linguaggi funzionali un solo dominio ed una sola funzione semantica
- `sem : expr -> env -> val`
- le funzioni di valutazione semantica assegnano un significato ai vari costrutti, con una definizione data sui casi della sintassi astratta

Semantica delle espressioni

```
type expr =
  | Den of ide
  | Fun of ide * expr
  | Plus of expr * expr
  | Apply of expr * expr
type val = | Int of int | Bool of bool | Efun of fun | Unbound
and fun = val -> val
type env = ide -> val
# let rec sem e rho = match e with
    | Den i -> rho i
    | Plus(e1, e2) -> plus(sem e1 rho, sem e2 rho)
    | Fun(i, e) -> Efun(function d -> sem e (bind(rho, i, d)))
    | Apply(e1, e2) -> match sem e1 rho with
          | Efun f -> f (sem e2 rho)
          | _ -> failwith("wrong application");;
val sem : expr -> env -> val = <fun>
```

- si vede lo scoping statico
  - valutazione del corpo della funzione in un ambiente che è quello dell'astrazione + passaggio parametri
- composizionalità
  - la semantica di una espressione è definita per composizione delle semantiche delle sue sottoespressioni

Caratteristiche della semantica denotazionale

- composizionalità
  - la semantica di un costrutto è definita per composizione delle semantiche dei suoi componenti
- assegna una denotazione al programma (funzione sui domini semantici)

```
sem : expr -> env -> val
```

  senza aver bisogno di conoscere lo "stato" iniziale
  - importante come base per analisi e trasformazioni che hanno a disposizione il solo programma
    - analisi statiche, ottimizzazioni, compilazione
- la costruzione della denotazione richiede un calcolo di minimo punto fisso, poichè le funzioni di valutazione semantica sono definite in modo ricorsivo
  - la semantica di un'espressione E è quindi `sem(E)`$\uparrow\omega$

La semantica operazionale

- lo stile di definizione tradizionale è quello dei sistemi di transizione
- se la esprimiamo in una notazione funzionale à la ML
  - la differenza fondamentale è nella natura della funzione di valutazione semantica, che da

    ```
    sem : expr -> env -> val
    ```

    diventa

    ```
    sem : expr * env -> val
    ```
- la semantica operazionale (come una qualunque implementazione) permette solo di eseguire un programma, una volta noto lo stato iniziale
- la semantica denotazionale costruisce valori di tipo funzionale (ordine superiore)
  - la semantica operazionale può essere vista come una sua versione al prim'ordine

Le funzioni in semantica operazionale

- bisogna cambiare anche il dominio semantico, da
  `fun = val -> val`, a
  `fun = expr * env`
- ancora una volta per eliminare un dominio funzionale (di ordine superiore)

```
# let rec sem (e, rho) = match e with
    | Den i -> rho i
    | Plus(e1, e2) -> plus(sem (e1, rho), sem (e2, rho))
    | Fun(i, e) -> Efun(e,rho)
    | Apply(e1, e2) -> match sem e1 rho with
          | Efun(Fun(i,e1),r1) -> sem(e1, bind(r1, i, sem (e2, rho)))
          | _ -> failwith("wrong application");;
val sem : expr * env -> val = <fun>
```

- il risultato si può ottenere solo utilizzando oggetti sintattici nei domini semantici e perdendo la composizionalità

Caratteristiche della semantica operazionale

- non è sempre composizionale
- non assegna una denotazione al solo programma, ma definisce come si raggiunge (per un certo programma) lo "stato finale" a partire dallo stato iniziale

- pur essendo le funzioni di valutazione semantica ancora definite in modo ricorsivo, non c'è bisogno di calcolare punti fissi
  - la semantica di un'espressione `E` in uno stato iniziale `rho` è `sem(E, rho)`, il valore ottenuto "eseguendo" le transizioni, finché possibile