

Elementi di semantica operazionale

1

Contenuti

- sintassi astratta e domini sintattici
 - un frammento di linguaggio imperativo
- semantica operazionale
 - domini semantici: valori e stato
 - relazioni di transizione e funzioni di valutazione semantica
- semantica e paradigmi
- semantica e supporto a tempo di esecuzione
- verso definizioni semantiche “eseguibili”

2

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 operativa*
- ☛ la semantica formale viene di solito definita su una rappresentazione dei programmi in *sintassi astratta*
 - invece che sulla reale rappresentazione sintattica concreta

3

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 un'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

4

Domini sintattici

- la sintassi astratta è definita specificando i *domini sintattici*
 - nomi di dominio, con metavariable relative
 - definizioni sintattiche
- assumiamo l'esistenza del dominio sintattico IDE (e poi anche semantico) degli *identificatori*, con metavariable I, I₁, I₂, ...

5

Un esempio (frammento di linguaggio imperativo)

- domini
 - EXPR (espressioni), con metavariable E, E₁, E₂, ...
 - COM (comandi), con metavariable C, C₁, C₂, ...
 - DEC (dichiarazioni), con metavariable D, D₁, D₂, ...
 - PROG (programma), con metavariable P
- definizioni sintattiche
 - $E ::= I \mid \text{val}(I) \mid \text{lambda}(I, E_1) \mid \text{plus}(E_1, E_2) \mid \text{apply}(E_1, E_2)$
 - $C ::= \text{ifthenelse}(E, C_1, C_2) \mid \text{while}(E, C_1) \mid \text{assign}(I, E) \mid \text{cseq}(C_1, C_2)$
 - $D ::= \text{var}(I, E) \mid \text{dseq}(D_1, D_2)$
 - $P ::= \text{prog}(D, C)$
- non è un frammento completo, poiché mancano
 - costanti (per esempio, interi e booleani)
 - altre necessarie operazioni (per esempio, su interi e booleani)

6

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,..... }
 - 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
stato = env * store (un elemento è una coppia formata da un elemento di env ed uno di store)

7

Semantica operativa: i domini semantici

- lo stile di definizione tradizionale è quello dei sistemi di transizione
- insieme di regole che definiscono
 - attraverso un insieme di relazioni di transizione
come lo stato cambia per effetto dell'esecuzione dei vari costrutti
- una tipica regola descrive la semantica di un costrutto sintattico c eseguito nello stato σ , specificando il nuovo stato σ' e/o il valore calcolato v
- dobbiamo prima di tutto specificare i domini semantici con cui rappresentiamo lo stato ed i valori

8

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
- ☛ $env = IDE \rightarrow dval$, con metavariables $\rho, \rho_1, \rho_2, \dots$
 - $[\rho / I \leftarrow d]$ indica l'ambiente $\rho' = \lambda\xi. \text{if } x = I \text{ then } d \text{ else } \rho \ x$
- ☛ il nostro frammento è imperativo ed ha la nozione di entità modificabile, cioè le variabili (create con le dichiarazioni e modificate con l'assegnamento)
 - non è possibile modellare lo stato con un'unica funzione, perché è possibile che identificatori diversi denotino la stessa locazione (aliasing)
- ☛ è quindi necessario un secondo componente dello stato, chiamato *memoria* (store)
 - per modellare l'associazione fra locazioni e valori che possono essere memorizzati
- ☛ $store = loc \rightarrow mval$, con metavariables $\sigma, \sigma_1, \sigma_2, \dots$
 - $[\sigma / l \leftarrow m]$ indica la memoria $\sigma' = \lambda\xi. \text{if } x = l \text{ then } m \text{ else } \sigma \ x$ ⁹

Domini semantici: i valori

- ☛ $env = IDE \rightarrow dval$, con metavariables $\rho, \rho_1, \rho_2, \dots$
- ☛ $store = loc \rightarrow mval$, con metavariables $\sigma, \sigma_1, \sigma_2, \dots$
- ☛ abbiamo già utilizzato (anche se non definito) due domini semantici di valori
 - $dval$ (valori denotabili), dominio dei valori che possono essere denotati da un identificatore nell'ambiente
 - $mval$ (valori memorizzabili), dominio dei valori che possono essere contenuti in una locazione nella memoria
- ☛ è necessario introdurre un terzo dominio semantico di valori
 - $eval$ (valori esprimibili), dominio dei valori che possono essere ottenuti come semantica di una espressione
- ☛ i tre domini sono in generale diversi anche se possono avere delle sovrapposizioni
 - in questi casi useremo delle funzioni di "trasformazione di tipo"
- ☛ per capire come definire i tre domini dobbiamo analizzare il linguaggio
- ☛ assumiamo predefiniti i domini int e $bool$
 - con le relative "operazioni primitive"
- ☛ ed il dominio loc con una "funzione"
 - $newloc : \rightarrow loc$ (che, ogni volta che viene applicata, restituisce una nuova locazione)

Il dominio semantico fun

$E ::= I \mid \text{val}(I) \mid \text{lambda}(I, E_1) \mid \text{plus}(E_1, E_2) \mid \text{apply}(E_1, E_2)$

- le espressioni contengono l'astrazione
 - $\text{lambda}(I, E_1)$ rappresenta una funzione
 - il cui corpo è l'espressione E_1
 - con parametro formale I
- in una semantica operativa (ed in una implementazione) quando si incontra una astrazione si può solo raccogliere l'informazione (sintattica ed eventualmente semantica) che verrà poi usata nella applicazione
- $\text{fun} = \text{expr}$
 - la semantica di una espressione di tipo lambda sarà l'espressione stessa
 - ad alcuni non piace (vedi semantica denotazionale)
 - c'è anche nascosta una decisione sulla regola di scoping (vedi dopo)

11

eval, dval, mval

$E ::= I \mid \text{val}(I) \mid \text{lambda}(I, E_1) \mid \text{plus}(E_1, E_2) \mid \text{apply}(E_1, E_2)$

$D ::= \text{var}(I, E) \mid \text{dseq}(D_1, D_2)$

- eval
 - deve contenere int, bool e fun
 - decidiamo che non contiene loc (decisione semantica, la sintassi lo permetterebbe)
- dval
 - deve contenere loc
 - decidiamo che contenga anche int, bool e fun
- mval
 - deve contenere int e bool
 - decidiamo che non contiene loc e fun (decisione semantica, la sintassi lo permetterebbe)

eval = [int + bool + fun]

dval = [loc + int + bool + fun]

mval = [int + bool]

12

La semantica operativa

- lo stile di definizione tradizionale è quello dei sistemi di transizione
 - insieme di regole che definiscono
 - attraverso un insieme di relazioni di transizione
 - come lo stato cambia per effetto dell'esecuzione dei vari costrutti
- un esempio di regola nel caso dei comandi
 - configurazioni: triple $\langle \text{COM}, \text{env}, \text{store} \rangle$
 - relazione di transizione: configurazione $\rightarrow_{\text{com}} \text{store}$

- un esempio di regola di transizione:

$$\frac{\langle C_1, \rho, \sigma \rangle \rightarrow_{\text{com}} \sigma_2 \quad \langle C_2, \rho, \sigma_2 \rangle \rightarrow_{\text{com}} \sigma_1}{\langle \text{cseq}(C_1, C_2), \rho, \sigma \rangle \rightarrow_{\text{com}} \sigma_1}$$

13

Relazioni o funzioni di transizione?

- le transizioni si rappresentano più naturalmente con le relazioni
 - se il linguaggio permette transizioni nondeterministiche
 - programmazione logica e linguaggi concorrenti
- negli altri casi è possibile rappresentare le transizioni attraverso funzioni
- l'esempio nella composizione di comandi
 - funzione di valutazione semantica
 - $C(\text{cseq}(C_1, C_2), \rho, \sigma) = C(C_2, \rho, C(C_1, \rho, \sigma))$
 - relazione di transizione

$$\frac{\langle C_1, \rho, \sigma \rangle \rightarrow_{\text{com}} \sigma_2 \quad \langle C_2, \rho, \sigma_2 \rangle \rightarrow_{\text{com}} \sigma_1}{\langle \text{cseq}(C_1, C_2), \rho, \sigma \rangle \rightarrow_{\text{com}} \sigma_1}$$

14

Le funzioni di valutazione semantica

$E ::= I \mid \text{val}(I) \mid \text{lambda}(I, E_1) \mid \text{plus}(E_1, E_2) \mid \text{apply}(E_1, E_2)$
 $C ::= \text{ifthenelse}(E, C_1, C_2) \mid \text{while}(E, C_1) \mid \text{assign}(I, E) \mid \text{cseq}(C_1, C_2)$
 $D ::= \text{var}(I, E) \mid \text{dseq}(D_1, D_2)$
 $P ::= \text{prog}(D, C)$

- ☛ per ogni dominio sintattico esiste una funzione di valutazione semantica
- ☛ $\mathcal{E} : \text{EXPR} * \text{env} * \text{store} \rightarrow \text{eval}$
- ☛ $C : \text{COM} * \text{env} * \text{store} \rightarrow \text{store}$
- ☛ $\mathcal{D} : \text{DEC} * \text{env} * \text{store} \rightarrow (\text{env} * \text{store})$
- ☛ $\mathcal{P} : \text{PROG} * \text{env} * \text{store} \rightarrow \text{store}$
- ☛ le funzioni di valutazione semantica assegnano un significato ai vari costrutti, con una definizione data sui casi della sintassi astratta

15

Semantica delle espressioni

$E ::= I \mid \text{val}(I) \mid \text{lambda}(I, E_1) \mid \text{plus}(E_1, E_2) \mid \text{apply}(E_1, E_2)$
 $\text{env} = \text{IDE} \rightarrow \text{dval}$
 $\text{store} = \text{loc} \rightarrow \text{mval}$
 $\text{fun} = \text{EXPR}$
 $\text{eval} = [\text{int} + \text{bool} + \text{fun}]$
 $\text{dval} = [\text{loc} + \text{int} + \text{bool} + \text{fun}]$
 $\text{mval} = [\text{int} + \text{bool}]$
 $\mathcal{E} : \text{EXPR} * \text{env} * \text{store} \rightarrow \text{eval}$

$\mathcal{E}(I, \rho, \sigma) = \text{dvaltoeval}(\rho(I))$
 $\mathcal{E}(\text{val}(I), \rho, \sigma) = \text{mvaltoeval}(\sigma(\rho(I)))$
 $\mathcal{E}(\text{plus}(E_1, E_2), \rho, \sigma) = \mathcal{E}(E_1, \rho, \sigma) + \mathcal{E}(E_2, \rho, \sigma)$

$\mathcal{E}(\text{lambda}(I, E_1), \rho, \sigma) = \text{lambda}(I, E_1)$

$\mathcal{E}(\text{apply}(E_1, E_2), \rho, \sigma) = \text{applyfun}(\mathcal{E}(E_1, \rho, \sigma), \text{evaltodval}(\mathcal{E}(E_2, \rho, \sigma)), \rho, \sigma)$
 $\text{applyfun}(\text{lambda}(I, E), d, \rho, \sigma) = \mathcal{E}(E) [\rho / I \leftarrow d] \sigma$

16

Semantica dei comandi

$C ::= \text{ifthenelse}(E, C_1, C_2) \mid \text{while}(E, C_1) \mid \text{assign}(I, E) \mid \text{cseq}(C_1, C_2)$
 $\text{env} = \text{IDE} \rightarrow \text{dval}$
 $\text{store} = \text{loc} \rightarrow \text{mval}$
 $\text{fun} = \text{EXPR}$
 $\text{eval} = [\text{int} + \text{bool} + \text{fun}]$
 $\text{dval} = [\text{loc} + \text{int} + \text{bool} + \text{fun}]$
 $\text{mval} = [\text{int} + \text{bool}]$
 $C : \text{COM} * \text{env} * \text{store} \rightarrow \text{store}$

$C(\text{assign}(I, E), \rho, \sigma) = [\sigma / \rho(I) \leftarrow \text{evaltomval}(\mathcal{E}(E, \rho, \sigma))]$

$C(\text{cseq}(C_1, C_2), \rho, \sigma) = C(C_2, \rho, C(C_1, \rho, \sigma))$

$C(\text{ifthenelse}(E, C_1, C_2), \rho, \sigma) =$

if $\mathcal{E}(E, \rho, \sigma)$ then $C(C_1, \rho, \sigma)$ else $C(C_2, \rho, \sigma)$

$C(\text{while}(E, C_1), \rho, \sigma) =$

if $\mathcal{E}(E, \rho, \sigma)$ then $C(\text{cseq}(C_1, \text{while}(E, C_1)), \rho, \sigma)$ else σ

17

Semantica delle dichiarazioni

$D ::= \text{var}(I, E) \mid \text{dseq}(D_1, D_2)$
 $\text{env} = \text{IDE} \rightarrow \text{dval}$
 $\text{store} = \text{loc} \rightarrow \text{mval}$
 $\text{fun} = \text{EXPR}$
 $\text{eval} = [\text{int} + \text{bool} + \text{fun}]$
 $\text{dval} = [\text{loc} + \text{int} + \text{bool} + \text{fun}]$
 $\text{mval} = [\text{int} + \text{bool}]$
 $\mathcal{D} : \text{DEC} * \text{env} * \text{store} \rightarrow (\text{env} * \text{store})$

☛ $\mathcal{D}(\text{var}(I, E), \rho, \sigma) = \text{let } \text{loc} = \text{newloc}() \text{ in}$

$([\rho / I \leftarrow \text{loc}], [\sigma / \text{loc} \leftarrow \text{evaltomval}(\mathcal{E}(E, \rho, \sigma))])$

☛ $\mathcal{D}(\text{dseq}(D_1, D_2), \rho, \sigma) = \text{let } (\rho', \sigma') = \mathcal{D}(D_1, \rho, \sigma) \text{ in } \mathcal{D}(D_2, \rho', \sigma')$

18

Semantica dei programmi

$P ::= \text{prog}(D, C)$
 $\text{env} = \text{IDE} \rightarrow \text{dval}$
 $\text{store} = \text{loc} \rightarrow \text{mval}$
 $\text{fun} = \text{EXPR}$
 $\text{eval} = [\text{int} + \text{bool} + \text{fun}]$
 $\text{dval} = [\text{loc} + \text{int} + \text{bool} + \text{fun}]$
 $\text{mval} = [\text{int} + \text{bool}]$
 $\mathcal{P} : \text{PROG} * \text{env} * \text{store} \rightarrow \text{store}$

- $\mathcal{P}(\text{prog}(D, C), \rho, \sigma) = \text{let } (\rho', \sigma') = \mathcal{D}(D, \rho, \sigma) \text{ in } C(C, \rho', \sigma')$

19

Semantica e paradigmi 1

- linguaggi funzionali
- un solo dominio sintattico: EXPR
- un solo dominio semantico per lo stato: env
- $\text{dval} = \text{eval}$
 - i valori esprimibili contengono sempre fun
- una sola funzione di valutazione semantica

• $\mathcal{E} : \text{EXPR} * \text{env} \rightarrow \text{eval}$

20

Semantica e paradigmi 2

- linguaggi imperativi
- tre domini sintattici: *EXPR*, *COM* e *DEC*
- due domini semantici per lo stato: *env* e *store*
- *dval*, *eval* ed *mval* sono generalmente diversi
 - i valori su cui si interpretano le astrazioni funzionali (*fun*) sono di solito solo denotabili
 - le locazioni sono sempre denotabili
- tre funzioni di valutazione semantica
$$\mathcal{E} : \text{EXPR} * \text{env} * \text{store} \rightarrow \text{eval}$$
$$\mathcal{C} : \text{COM} * \text{env} * \text{store} \rightarrow \text{store}$$
$$\mathcal{D} : \text{DEC} * \text{env} * \text{store} \rightarrow (\text{env} * \text{store})$$

21

Semantica e paradigmi 3

- linguaggi orientati ad oggetti
- oltre ad: *EXPR*, *COM* e *DEC*, dichiarazioni di classe
- tre domini semantici per lo stato: oltre a *env* e *store*, un dominio nuovo (*heap*), per modellare i puntatori e gli oggetti
- *dval*, *eval* ed *mval* sono generalmente diversi
 - i valori su cui si interpretano le astrazioni funzionali (*fun*) sono di solito solo denotabili
 - le locazioni sono sempre denotabili
 - gli oggetti di solito appartengono a tutti e tre i domini
- le funzioni di valutazione semantica, prendono (e restituiscono) anche la *heap*
$$\mathcal{C} : \text{COM} * \text{env} * \text{store} * \text{heap} \rightarrow (\text{store} * \text{heap})$$

22

Semantica e supporto a run time

- le differenze fra i linguaggi dei diversi paradigmi si riflettono in differenze nelle corrispondenti implementazioni
- tutte le caratteristiche importanti per progettare un interprete o un supporto a tempo di esecuzione, si possono ricavare ispezionando i soli domini semantici della semantica operativa

23

Verso definizioni semantiche “eseguibili”

- quali caratteristiche sono importanti?
 - la λ -astrazione
 - con funzioni di ordine superiore
 - i tipi: il meccanismo naturale per definire domini sintattici e semantici
 - via enumerazione, somma, prodotto, iterazione, funzioni
 - la definizione di funzioni per casi
 - sui casi di un tipo (la sintassi astratta)
- il metalinguaggio è un frammento di un vero e proprio linguaggio di programmazione (funzionale, di ordine superiore, tipato, con pattern matching)
 - riesprimendo le nostre semantiche in tale linguaggio, otteniamo semantiche eseguibili

24