



ELEMENTI DI SEMANTICA OPERAZIONALE

1

Contenuti



- 👁️ sintassi astratta e domini sintattici
 - un frammento di linguaggio imperativo
- 👁️ semantica operativa
 - 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 possiede
 - **una sintassi**
 - ✓ le formule ben formate del linguaggio (o programmi sintatticamente corretti)
 - **una semantica**
 - ✓ che assegna un significato alle formule ben formate
 - ✓ 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
 - *Semantica denotazionale, Semantica operativa, Semantica assiomatica*
- ✦ la semantica formale viene di solito definita su una rappresentazione dei programmi in *sintassi astratta*
 - invece che sulla reale rappresentazione sintattica concreta

3

Semantica dei PL



- ✦ Tre metodi di analisi semantica:
 - Semantica operativa: descrive il significato di un programma in termini dell'evoluzione del programma mediante una nozione di macchina astratta;
 - Semantica denotazionale: il significato di un programma è definito in termini di funzioni e spazi funzionali;
 - Semantica operativa descrive il significato di un programma in termini delle proprietà che sono soddisfatte prima e dopo l'esecuzione del programma
- ✦ Ciascuno di questi metodi ha diversi vantaggi e svantaggi in termini degli aspetti matematici, della facilità di uso nelle dimostrazioni, o quanto è facile implementare un interprete o compilatore.

4

Il nostro corso



- ✎ Definizione della strutture a run-time
 - Semantica operativa
- ✎ Object orientation
 - Semantica assiomatica

5

Semantica Operazionale



- ✎ Macchina astratta= Sistema di transizione
 - Un insieme Config configurazioni (stati),
 - Relazione binaria $\rightarrow \subseteq \text{Config} \times \text{Config}$.
- ✎ Notazione: $c \rightarrow c'$ significa che c e c' sono nella relazione \rightarrow

6

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)

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

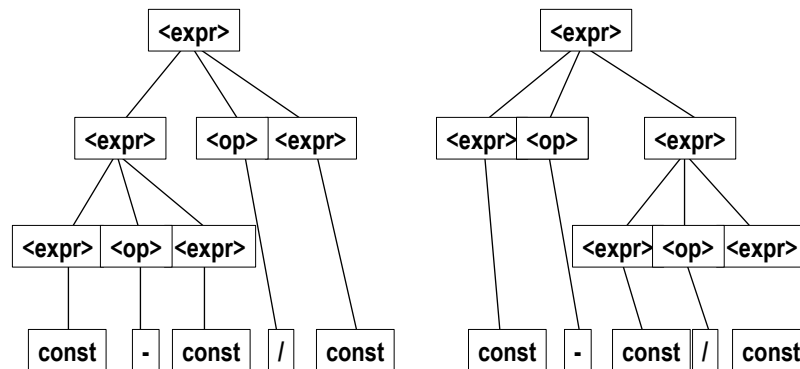
7

Ambiguità



$\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle \mid \text{const}$

$\langle \text{op} \rangle \rightarrow / \mid -$



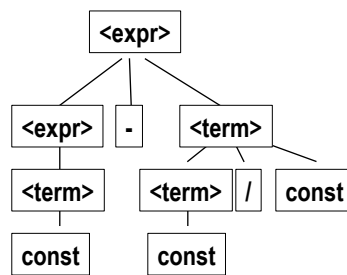
1-8

Grammatiche non ambigue



Precedenza tra gli operatori nella sintassi

$\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle - \langle \text{term} \rangle \mid \langle \text{term} \rangle$
 $\langle \text{term} \rangle \rightarrow \langle \text{term} \rangle / \text{const} \mid \text{const}$

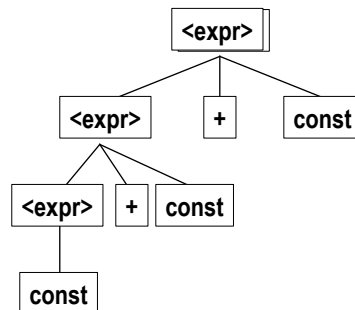


1-9

Associativita' operatori



$\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle + \langle \text{expr} \rangle \mid \text{const}$ (ambiguous)
 $\langle \text{expr} \rangle \rightarrow \langle \text{expr} \rangle + \text{const} \mid \text{const}$ (unambiguous)



1-10

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, I_1, I_2, \dots

11

Un esempio (linguaggio imperativo)



- ✎ domini
 - EXPR (espressioni), con metavariable E, E_1, E_2, \dots
 - COM (comandi), con metavariable C, C_1, C_2, \dots
 - DEC (dichiarazioni), con metavariable D, D_1, D_2, \dots
 - 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)

12

Domini semantici



i domini semantici vengono definiti da *equazioni di dominio*

nome-dominio = espressione-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 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)

13

Semantica Operazionale



- 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

14

Domini semantici: lo stato



- ✎ lo stato comprende un dominio chiamato *ambiente* (environment)
 - per modellare l'associazione tra gli identificatori ed i valori che questi possono denotare
- ✎ $env = IDE \rightarrow dval$, con metavariable $\rho, \rho_1, \rho_2, \dots$
 - La notazione

$[\rho / l \leftarrow d]$

indica l'ambiente

$\rho' = \lambda\xi. \text{if } x = l \text{ then } d \text{ else } \rho \ x$

15

Lo stato



- ✎ il nostro linguaggio 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 metavariable $\sigma, \sigma_1, \sigma_2, \dots$
 - $[\sigma / l \leftarrow m]$ indica la memoria

$\sigma' = \lambda\xi. \text{if } x = l \text{ then } m \text{ else } \sigma \ x$

16

Lo stato



- ✎ env = IDE \rightarrow dval, con metavariable $\rho, \rho_1, \rho_2, \dots$
- ✎ store = loc \rightarrow mval, con metavariable $\sigma, \sigma_1, \sigma_2, \dots$

17

Domini semantici: i 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
- **eval (valori esprimibili)**, dominio dei valori che possono essere ottenuti come semantica di una espressione
- ✎ 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)

18

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
- fun = expr**
 - la semantica di una espressione di tipo lambda sarà l'espressione stessa

19

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]

20

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:

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

21

Equazioni e Regole



- 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

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

22

Large Step Semantics



- ✦ Quella che abbiamo visto e' nota in letteratura come "large step semantics"
- ✦ Restituisce direttamente il risultato finale della valutazione
- ✦ Nel nostro caso del linguaggio imperativo il risultato finale della valutazione e' la memoria.

23

Small steps semantics



- ✦ Tecnica differente per descrivere la semantica operativa
- ✦ Descrive tutti gli stati intermedi della valutazione

24

Small Steps: Composizione sequenziale



✎ $\langle c1, env, store \rangle \rightarrow \langle c', env, store' \rangle$

 $\langle c1 ; c2, env, store \rangle \rightarrow \langle c'; c2, env, store' \rangle$

✎ $\langle skip ; c2, env, store \rangle \rightarrow \langle c2, env, store \rangle$

25

Small steps vs Large Step



✎ La differenza principale emerge quando si vuole trattare situazioni in cui abbiamo

- Non determinismo
- Comunicazione

✎ In linguaggi semplici le sue semantiche coincidono.

26

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

27

Semantica delle espressioni

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

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

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

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

28

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

29

Semantica delle dichiarazioni

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



$\mathcal{D}(\text{var}(I, E), \rho, \sigma) = \text{let } \text{loc} = \text{newloc}() \text{ in}$
 $([\rho / I \leftarrow \text{loc}], [\sigma / \text{loc} \leftarrow \text{evaltomval}(\mathcal{E}(E, \rho, \sigma))])$

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

30

Semantica dei programmi

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



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

31

Semantica e paradigmi di programmazione

- 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



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



32

Semantica e paradigmi



- 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
 - $\mathcal{E} : EXPR * env * store \rightarrow eval$
 - $C : COM * env * store \rightarrow store$
 - $\mathcal{D} : DEC * env * store \rightarrow (env * store)$

33

Semantica e paradigmi



- 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 $heap$
- $C : COM * env * store * heap \rightarrow (store * heap)$

34

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

35

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

36