Semantica di linguaggi di programmazione

Ne esistono differenti stili a seconda di

- paradigma di programmazione
- uso (validazione, prototyping, verifica di complessità...)
- gusto personale dell'inventore

Ne vedremo due, già introdotti a LP

Operazionale	Denotazionale
Ogni termine del linguaggio si semplifica fino ad un termine che rappresenta un valore	Ogni termine del linguaggio denota un valore in un opportuno dominio
Servono informazioni aggiuntive (contestuali)	Le informazioni contestuali sono rappresentate nel dominio, usando valori funzionali
Astrazione di una macchina	Buon supporto per interpretazioni di logiche

Dominio della semantica =?

(materiale propedeutico: dispense di LP, sezione 1 e appendici A, B,C)

Programma = termine su una qualche grammatica CF

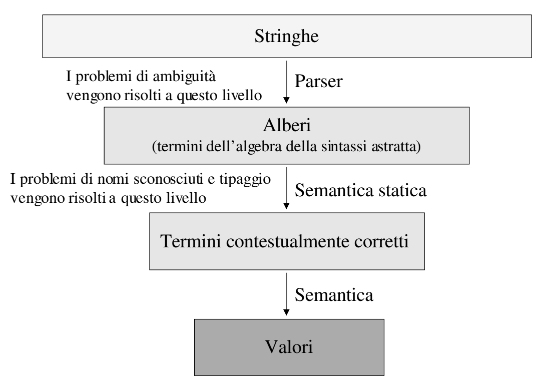
Vogliamo eliminare i termini *sbagliati* non appena possibile

- Per semplificare la definizione di semantica, fattorizzando i problemi che si possono incontrare (=le motivazioni per cui un programma è sbagliato) ed affrontandoli separatamente
- Per facilitare la comprensione da parte dell'utente degli errori commessi, presentando le nozioni minime alla comprensione

Un programma può essere sbagliato a ~~3~~ 4 livelli

- Non corrisponde alla sintassi descritta dalla grammatica (formalmente: non appartiene a L(G))
- Non soddisfa i vincoli contestuali (formalmente la semantica statica dà errore)
- La sua semantica non è definita (eg non terminazione) (formalmente la semantica dinamica non produce risultato accettabile)
- È logicamente sbagliato, cioè non fa quello che dovrebbe. Questo sarà l'argomento del la parte di specifiche

Percorso



Semantica statica

Controlla sostanzialmente due cose: uso di identificatori (costanti, funzioni, metodi..) e tipaggio

Tipaggio

- Con un insiemi di tipi finito e in assenza di identificatori (o con identificatori lessicalmente divisi per tipi) può essere catturato dalla grammatica. Ma non sono casi interessanti.
- Impossibile in particolare catturarlo a livello della grammatica se il linguaggio permette la definizione di tipi da parte dell'utente

Identificatori

- Presenti in tutti i linguaggi di alto livello, perché permettono di astrarre, semplificando anche l'esecuzione (su questo punto ci torneremo).
- Devono essere usati coerentemente dal punto di vista del tipo e della natura (con la loro dichiarazione/definizione, se c'è)

Exp

Num ::= …..
ExpNum ::= Num | ExpNum + ExpNum | ExpNum × ExpNum | Id
Id ::= ….
Dec ::= Id = ExpNum;
Prog ::= Dec* **eval** ExpNum

~~Tutte le stringhe sono contestualmente corrette~~

Vincolo contestuale: una costante deve essere dichiarata prima di poter essere usata

Idea intuitiva: introduco la nozione di ambiente che mi memorizza le costanti dichiarate.
All'inizio del programma l'ambiente è vuoto (non ho ancora esaminato nessuna dichiarazione)
Ogni dichiarazione (corretta) amplia l'ambiente.
Un'espressione è corretta se usa solo costanti già nell'ambiente.

Formalizzazione

Voglio definire una funzione a valori booleani (che interpreterò come corretto/scorretto) sul linguaggio di tipo Prog $[\![\_]\!]^s_P : L_{Prog}(Exp) \to \mathbb{B}$

Per poterlo fare ho bisogno di funzioni ausiliarie che verificano la correttezza dei sottoprogrammi.
Queste hanno bisogno di e/o costruiscono l'ambiente

$[\![\_]\!]^s_E : L_{ExpNum}(Exp) \times Env^s \to \mathbb{B}$
$[\![\_]\!]^s_D : L_{Dec}(Exp) \times Env^s \to Env^s$

(Ci sarebbero anche funzioni per Num e Id ma sono sempre vere)

La informazioni da memorizzare nell'ambiente in questo caso così facile sono solo i nomi delle costanti già introdotte

$Env^s = \wp_{Fin}(L_{Id}(Exp))$

Definizione delle funzioni

Le funzioni di semantica statica si definiscono per (mutua) induzione. Le presentiamo in stile top-down (solo presentazione, perché stiamo dando un *insieme* di metaregole).

Un programma è corretto se la sua parte dichiarazioni costruisce un ambiente in cui è corretta l'espressione da valutare:

Per esattezza qui ci andrebbe la funzione sulle sequenze di dichiarazioni

$$[\![dl\ \mathbf{eval}\ \ e]\!]^s_P = [\![e]\!]^s_E\,(\,[\![dl]\!]^s_D(\varnothing))$$

Per valutare le dichiarazioni ho bisogno di un ambiente, all'inizio sarà quello vuoto

Notazione equivalente:

$$\frac{[\![dl]\!]^s_D(\varnothing) = \rho \qquad\qquad [\![e]\!]^s_E\,(\rho) = b}{[\![dl\ \mathbf{eval}\ e]\!]^s_P = b}$$

Dichiarazioni

Una dichiarazione incrementa l'ambiente

Se $[\![e]\!]^s_E\,(\rho)$ = true, allora $[\![id = e]\!]^s_D\,(\rho) = \rho\cup\{id\}$ altrimenti err

Questo va bene solo se l'espressione è corretta

Non abbiamo previsto la possibilità che la valutazione delle dichiarazioni generasse un errore.
Per tenerne conto aggiungiamo un elemento speciale agli ambienti statici.

$$Env^s = \wp_{Fin}(L_{Id}(Exp)\,) \cup\{err\}$$

L'errore andrà, naturalmente propagato

$$[\![d]\!]^s_D\,(err\,) = err \qquad [\![e]\!]^s_E\,(err) = false$$

Esercizio proposto (facile) che cosa va modificato per impedire doppie dichiarazioni della stessa costante?

Espressioni

Per le espressioni abbiamo vari casi, seguiremo la struttura induttiva del linguaggio nel definire la semantica statica

I numeri sono sempre corretti

$[\![n]\!]^s_E\,(\rho)$ = true, se $\rho\neq err$ e $n\in L_{Num}(Exp)$

Somma e prodotto propagano correttezza ed errori

$[\![e + e']\!]^s_E\,(\rho) = [\![e]\!]^s_E\,(\rho) \wedge [\![e']\!]^s_E\,(\rho)$

$[\![e \times e']\!]^s_E\,(\rho) = [\![e]\!]^s_E\,(\rho) \wedge [\![e']\!]^s_E\,(\rho)$

Una costante è corretta se e solo se è nota nell'ambiente

$[\![id]\!]^s_E\,(\rho) = id \in \rho$

Esercizio

Verificare che il seguente programma è staticamente corretto

X =7; Y=X+8; **eval** X+Y

Calcolo l'ambiente statico creato dalle dichiarazioni

$[\![X =7;\ Y=X+8;]\!]^s_D\,(\varnothing) =$ Per le regole sulle sequenze (che non abbiamo mai dato)

$[\![Y=X+8;]\!]^s_D\,(\underbrace{[\![X =7;\ ]\!]^s_D\,(\varnothing)}_{\{X\}}\,) =$ Perché ($[\![7]\!]^s_E\,(\varnothing)$=true, essendo $\varnothing\neq$err e $7 \in L_{Num}(Exp)$

$[\![Y=X+8;]\!]^s_D\,(\{X\})=$ Perché ($[\![X+8]\!]^s_E\,(\{X\})$=
($[\![X]\!]^s_E\,(\{X\}) \wedge ([\![8]\!]^s_E\,(\{X\})$=
true $\wedge$ true

$\{X,Y\}$

Essendo $X \in \{X\}$

essendo $\{X\}\neq$err e $8 \in L_{Num}(Exp)$

Valuto l'espressione nell'ambiente statico creato dalle dichiarazioni

$[\![X + Y]\!]^s_E\,(\{X,Y\})= ([\![X]\!]^s_E\,(\{X,Y\}) \wedge ([\![Y]\!]^s_E\,(\{X,Y\})$ = true $\wedge$ true = true

Esercizi Proposti

Verificare che il seguente programma è staticamente corretto
X =7 × 2; Z=X+X; X = 2; **eval** Z+X

Verificare che il seguente programma **non** è staticamente corretto
X =2; Z=Y+X; **eval** Z

Espressioni tipate

Che cosa dobbiamo cambiare per gestire espressioni di vari tipi?

Num	::=	…..
Bool	::=	tt | ff
Exp	::=	Id | Num | Bool | Exp + Exp | Exp × Exp| Exp &Exp ….

Non vale la pena di distinguere l'applicazione degli operatori a livello di grammatica, perché gli stessi problemi ci sono in ogni caso per via degli identificatori che possono avere il tipo che vogliono

Idea intuitiva: dovremo tener conto del tipo delle espressioni per sapere quali operazioni sono lecite su di esse.

Vogliamo quindi che

- Un ambiente statico corretto associ agli identificatori noti (che sono un insieme finito) il loro tipo.
- La semantica delle espressioni dia come risultato il tipo (o un errore)

Formalmente

Funzioni parziali a dominio finito

$$Env^s = [L_{Id}(Exp) \to_P Types]_{Fin} \cup\{err\}$$

$$[\![\_]\!]^s_E{:}L_{Exp}(Exp) \times Env^s\to Types \cup\{err\}$$

Types = {int, bool}

Se $[\![e]\!]^s_E(\rho)=t \in$ Types, allora $[\![id = e]\!]^s_D(\rho) = \rho[t/id]$, altrimenti err

Notazione per le funzioni parziali a dominio finito:
[] funzione con dominio vuoto
$[a_1/x_1,\dots, a_n/x_n]$ con tutte le x_i distinte funzione f con dominio $\{x_1,\dots,x_n\}$ definita da $f(x_i)=(a_i)$
$\rho[t/id]$ funzione con dominio $dom(\rho) \cup \{id\}$ definita da $\rho[t/id](x)= \rho(x)$ se $x \neq id$ e $\rho[t/id](id)=t$

I numeri sono sempre corretti
$[\![n]\!]^s_E(\rho)$ = int, se $\rho \neq$ err e $n \in L_{Num}(Exp)$, altrimenti err
Idem per le costanti booleane
$[\![b]\!]^s_E(\rho)$ = bool, se $\rho \neq$ err e $b \in L_{Bool}(Exp)$, altrimenti err
Somma e prodotto producono interi se applicati ad interi, errori altrimenti
$[\![e + e']\!]^s_E(\rho)$ = int, se $[\![e]\!]^s_E(\rho)$ = int $\wedge$ $[\![e']\!]^s_E(\rho)$ = int, err altrimenti
Analogamente per gli operatori boleani
Una costante ha il tipo che risulta nell'ambiente oppure è ignota (err)
$[\![id]\!]^s_E(\rho) = \rho(id)$ se $id \in Dom(\rho)$, err altrimenti

Notazione equivalente

$\_ \vdash \_ \to \_ \subseteq Env^s \times L_{Dec}(Exp) \times Env^s \cup \{err\}$
$\_ \vdash \_ : \_ \subseteq Env^s \times L_{Exp}(Exp) \times Types \cup \{err\}$
$[\![\_]\!]^s_E: \quad L_{Exp}(Exp) \times Env^s \to Types \cup \{err\}$

$$\frac{\rho \vdash e : t}{\rho \vdash id = e \to \rho[t/id]}\ t \in \text{Types} \qquad \frac{\rho \vdash e : err}{\rho \vdash id = e \to err}$$

$$\frac{}{\rho \vdash n:int}\ \rho\neq err, n\in L_{Num}(Exp) \qquad \frac{}{err \vdash e: err}$$

$$\frac{\rho \vdash e:int \quad \rho \vdash e':int}{\rho \vdash e + e':int} \qquad \frac{\rho \vdash e: err}{\rho \vdash e + e': err} \qquad \frac{\rho \vdash e': err}{\rho \vdash e + e': err}$$

$$\frac{}{\rho \vdash id : \rho(id)}\ id \in Dom(\rho),$$

Spesso le regole per trattare gli errori si omettono, e analogamente nell'altro stile si definiscono funzioni parziali senza err nel codominio

Ambienti locali

Che cosa dobbiamo cambiare per gestire costrutti con ambienti locali?
Exp ::= ….**let** dec* **in** Exp

Vogliamo che le costanti dichiarate localmente non siano più visibili dopo la fine del costrutto

$$\frac{\rho \vdash ds \to \rho' \quad \rho' \vdash e: t}{\rho \vdash \textbf{let}\ ds\ \textbf{in}\ e: t}$$

Questo si ottiene automaticamente, perché l'ambiente ρ' non compare nella conseguenza

Complicazione ulteriore: vogliamo permettere di ridichiarare nell'ambiente locale le costanti globali, ma non doppie dichiarazioni allo stesso livello.
Idea intuitiva: in tutte le regole avremo un ambiente locale ed uno globale, da usare per tenere traccia degli identificatori globali ma non modificare.
Problema: e nei casi tipo **let** d **in let** d' **in let** d" **in** e ci basteranno due ambienti? Sì, perché analizzeremo i costrutti uno alla volta

Due livelli di ambiente

Dove serve distinguere fra ambiente locale e globale?
Solo al momento di decidere se una dichiarazione è lecita.
Basta quindi cambiare
$[\![\_]\!]^s_D: L_{Dec}(Exp) \times \underset{\uparrow}{Env^s} \times Env^s \to Env^s$
Ambiente globale usato ma non modificato

Se $[\![e]\!]^s_E(\rho_g[\rho_l])=t \in$ Types, e $id \notin Dom(\rho_l)$, allora $[\![id = e]\!]^s_D(\rho_g,\rho_l) = \rho_l[t/id]$, altrimenti err

Notazione ulteriore per le funzioni parziali a dominio finito:
$\rho[\rho_l]$ funzione con dominio $dom(\rho) \cup dom(\rho_l)$ definita da
$\rho[\rho_l](x)= \rho_l(x)$ se $x \in dom(\rho_l)$, e $\rho[\rho_l](x)= \rho(x)$ altrimenti

$$\frac{[\![ds]\!]^s_D(\rho,[]) = \rho_l \quad [\![e]\!]^s_E(\rho[\rho_l])=t}{[\![\textbf{let}\ ds\ \textbf{in}\ e: t\,]\!]^s_E(\rho) = t}$$

La dichiarazione locale copre (temporaneamente) quella globale, perché l'identificatore viene cercato prima nell'ambiente locale e solo se non lo si trova in quello globale

Riassunto delle regole di dichiarazione

$$\frac{[\![id = e;]\!]^s_D(\rho, \rho_l) = \rho_{loc} \qquad [\![ds]\!]^s_D(\rho, \rho_{loc}) = \rho_0}{[\![id = e;ds]\!]^s_D(\rho, \rho_l) = \rho_0}$$

$$\frac{[\![e]\!]^s_E(\rho[\rho_l])=t}{[\![id = e;]\!]^s_D(\rho, \rho_l) = \rho_l[t/id]}\ id \notin Dom(\rho_l) \wedge t \in \text{Types}$$

$$\frac{[\![e]\!]^s_E(\rho[\rho_l]) = err}{[\![id = e;]\!]^s_D(\rho, \rho_l) = err}$$

$$\frac{}{[\![id = e;]\!]^s_D(\rho, \rho_l) = err}\ id \in Dom(\rho_l)$$

Esercizio

In quali ambienti statici è corretta la seguente espressione?
let x=2; y=z+1; **in let** x=x+y; **in** x*y
Partiamo con un generico ambiente ρ_0 e vediamo quali restrizioni si deducono per avere la correttezza (intuitivamente: $\rho_0(z)$ = int).

$$\frac{\dfrac{T_1}{[\![x=2;\ y=z+1;\ ]\!]^s_D(\rho_0,[]) = \rho_l} \qquad \dfrac{T_2}{[\![\textbf{let}\ x=x+y;\ \textbf{in}\ x*y\ ]\!]^s_E(\rho_0[\rho_l]) = t}}{[\![\textbf{let}\ x=2;\ y=z+1;\ \textbf{in let}\ x=x+y;\ \textbf{in}\ x*y\ ]\!]^s_E(\rho_0) = t}$$

$\rho_0 \neq$ err
$\rho_0(z)$ = int
ρ_l = [int/x,int/y]

$\llbracket x=2;\ y=z+1;\ \rrbracket^s_D(\rho_0,[]) = \rho_l$

$\llbracket x=2;\rrbracket^s_D(\rho_0,[]) = [int/x]$ $\quad$ $\llbracket y=z+1;\ \rrbracket^s_D(\rho_0,[int/x]) = [int/x, t/y\ ] = \rho_l$

$x \notin \emptyset = Dom([])$ $\quad$ $y \notin \{x\} = Dom([int/x])$

$\llbracket 2\rrbracket^s_E(\rho_0[[]]) = int$ $\quad$ $\llbracket z+1\rrbracket^s_E(\rho_0[[int/x]]) = t$

$\rho_0 \neq err$ $\quad$ $t=int$

$\llbracket z\rrbracket^s_E(\rho_0[[int/x]]) = int$ $\quad$ $\llbracket 1\rrbracket^s_E(\rho_0[[int/x]]) = int$

$\rho_0[[int/x]](z) = int$

$\rho_0(z) = int$

Perché $\rho_0[[int/x]](z) = \rho_0(z)$, essendo $z \neq x$

Regole di correttezza dei numeri

$\llbracket \textbf{let}\ x=x+y;\ \textbf{in}\ x*y\ \rrbracket^s_E(\rho_0[[int/x, int/y]]) = t$

$\llbracket x=x+y;\rrbracket^s_D(\rho_0[[int/x, int/y]], []) = [t_1/x]$ $\quad$ $\llbracket x*y\ \rrbracket^s_E(\rho_0[[int/x, int/y]][t_1/x]) = t$

$x \notin \emptyset = Dom([])$

$\llbracket x+y\rrbracket^s_E(\rho_0[[int/x, int/y]]) = t_1$

$t_1 = int$

$\llbracket x\rrbracket^s_E(\rho_0[[int/x, int/y]]) = int$ $\quad$ $\llbracket y\rrbracket^s_E(\rho_0[[int/x, int/y]]) = int$

$\rho_0[[int/x, int/y]](x) = int$ $\quad$ $\rho_0[[int/x, int/y]](y) = int$

stesso albero di sinistra nello stesso ambiente, perché $\rho_0[[int/x, int/y]][t_1/x]) = \rho_0[[t_1/x, int/y]] = \rho_0[[int/x, int/y]]$