Semantica denotazionale algebrica di LW

Idea intuitiva: i valori che vogliamo "denotare" sono:

- gli statements sono funzioni di trasformazioni di *stato* (interno)
- espressioni etc. producono valori (di base) a partire dalle informazioni sui valori degli identificatori
- Dichiarazioni etc. determinano quali sono gli identificatori usabili nel seguito (e per le sole funzioni quali sono i rispettivi valori), cioè creano/modificano gli *ambienti*

Quindi i carrier dell'algebra semantica dovranno essere qualche tipo di funzioni aventi come argomenti le *informazioni sui valori degli identificatori*

Le informazioni sui valori degli identificatori sono di due tipi:

- Ci sono identificatori associati a variabili, il cui valore cambia nel tempo durante l'esecuzione dei programmi.
- Altri associati a funzioni, il cui valore è costante per tutta la durata dell'esecuzione.

Modello ambiente/stato

Locazioni: astrazione del concetto di cella di memoria, per semplicità sono tipate, in modo che le locazioni per il tipo T siano abbastanza "grandi" da contenere un valore di tipo T

Valori costanti. Siccome nel nostro linguaggio ci sono dichiarazioni di variabili di tipo base e di costante di tipo funzionale, avremo nei *valori costanti* funzioni e nell'immagine dello stato solo valori di tipo intero e booleano

Ci dovranno essere dei vincoli di coerenza, cioè locazioni di tipo T dovranno essere associate a valori di tipo T (il suo "contenuto")

Ambiente

Un ambiente associa ad un identificatore un "valore" che resta costante per tutta la durata dell'esecuzione.

Per gli identificatori di funzione si tratta della funzione descritta dal body.

Per le variabili si tratta della locazione.

$$Env = [L_{Id}(LW) \to_p Fun \cup Loc]_{Fin}$$

Le locazioni sono tipate: $Loc = \bigcup_{T \in BTypes} Loc_T$
Non ci interessa dettagliare come siano fatte le locazioni di un dato tipo, ma assumiamo di averne a disposizione un numero illimitato

$Fun = [Arg \to_p Res]$
$Arg = [\mathbb{Z} | \mathbb{B}]^+ \times States$
$Res = [\mathbb{Z} | \mathbb{B} | \{*\}] \times States$

Stato al momento della chiamata, da usare per valutare variabili globali

Stato modificato dalla chiamata

void

Stato

Uno stato rappresenta una fotografia della memoria del programma ad un dato istante dell'esecuzione.

Quindi associa ad ogni locazione (=cella) un valore.

$$States = [Loc \rightharpoonup_p Value]_{Fin}$$

Solo alle locazioni inizializzate (che sono un numero finito) è associato un valore

$Value = \bigcup_{T \in BTypes} [\![T]\!]$

$[\![T]\!]$ è l'insieme dei valori di tipo T: $[\![\textbf{int}]\!] = \mathbb{Z}$, $[\![\textbf{bool}]\!] = \mathbb{B}$

Siccome le locazioni sono tipate, l'associazione deve rispettare i tipi:

Per ogni stato s, $l \in Loc_T$ e s(l) definito implica $s(l) \in [\![T]\!]$.

Alternativamente Loc e Value possono essere visti come famiglie $Loc = \{Loc_T\}_{T \in Btypes}$, $Value = \{[\![T]\!]\}_{T \in Btypes}$, ed s: $Loc \rightharpoonup_p Value$ come una famiglia di funzioni $\{s_T: Loc_T \rightharpoonup_p [\![T]\!]\}_{T \in Btypes}$

Nomenclatura. Dato uno stato s

Dom(s). Il *dominio* di s consiste di tutte le locazioni "riservate", cioè associate ad un identificatore (ad un qualche livello più o meno locale)

DDef(s). Il *dominio di definizione* di s consiste di tutte le locazioni inizializzate.

Quindi $DDef(s) \subseteq Dom(s)$

Locazioni “nuove”

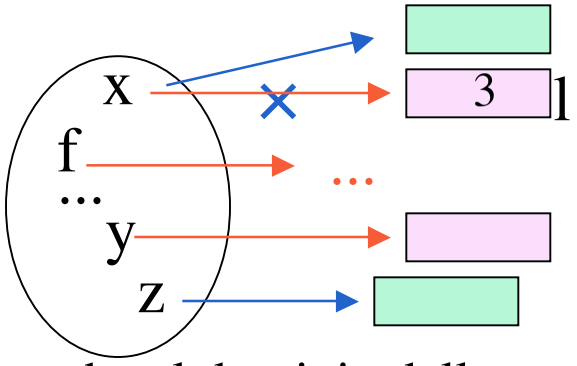
Nel seguito ci servirà allocare delle locazioni distinte da tutte quelle in uso (per evitare aliasing e cancellazione di dati).

Dati un ambiente r ed uno stato s, quali sono le locazioni in uso?

Tutte le locazioni nell’immagine di r sono sicuramente in uso, ma non sono le sole.

Infatti, se abbiamo una dichiarazione locale di x che copre una dichiarazione globale di variabile per x, la locazione l associata ad x nell’ambiente globale non compare nell’immagine dell’ambiente aggiornato (risulta coperta dal nuovo valore associato a x), ma non per questo si può riassegnare.

```
int x = 3;
int y;
void f(bool x; int z;)
    {if (x) {y=1;} else {y=z;}}
```



Però in un caso come questo, l appartiene anche al dominio dello stato (non necessariamente al dominio di definizione).

Quindi, definiamo il predicato Nuova $\subseteq$ Loc $\times$ Env $\times$ States come

Nuova(l,r,s) sse l $\notin$ (Im(r) $\cup$ Dom(s))

Funzione(/i) Semantica(/he)

Come nel caso della semantica statica, la semantica è una famiglia di funzioni indiciata sui non terminali (un omomorfismo)
Dovremmo quindi definire ciascuna funzione.
Definiremo solo quelle per tipi, liste di dichiarazioni, di statements, di espressioni e quella per il programma. Le altre si desumono da queste, perché il loro dominio è incluso in uno dei precedenti.

Per i tipi l'abbiamo già vista (manca la regola $[\![\textbf{void}]\!]_{RType} = \{*\}$)
$[\![\_]\!]_{RType}: L_{RType}(LW) \rightarrow$ Set

Le dichiarazioni modificano l'ambiente.
$[\![\_]\!]_{Decs}: L_{Decs}(LW) \rightarrow Env \rightarrow States \rightarrow Env \times States$
Lo stato serve (e può venir modificato) a causa delle dichiarazioni con inizializzazione, oltre che per individuare le locazioni libere.

Gli statements modificano lo stato.
$[\![\_]\!]_{Stats}: L_{Stats}(LW) \rightarrow Env \rightarrow States \rightarrow States$

Le espressioni producono un valore e(d eventualmente) modificano lo stato (side-effects).
$[\![\_]\!]_{Exps}: L_{Exps}(LW) \rightarrow Env \rightarrow States \rightarrow Value^{+} \times States$

Nelseguito tralasceremo gli indici

Definizione delle funzioni semantiche

Definiamo le funzioni semantiche per (mutua) ricorsione

Possiamo indifferentemente usare la notazione "in linea"

F(exp) = …. F(exp')….

Oppure quella a regole $\dfrac{F(exp') = A}{F(exp) = ….A….}$

Vediamo quale è più conveniente.

Ad esempio consideriamo il caso della concatenazione di dichiarazioni.

L'idea intuitiva è di valutare la prima dichiarazione e usare il risultato come punto di partenza (= argomento) per valutare le restanti dichiarazioni:

Notazione "in linea":

$[\![d\ ds]\!]_{Decs}(r,s) = [\![ds]\!]_{Decs}([\![d]\!]_{Decs}(r,s))$

Notazione a regole: $\dfrac{[\![d]\!]_{Decs}(r,s) = (r',s') \qquad [\![ds]\!]_{Decs}(r',s') = (r'',s'')}{[\![d\ ds]\!]_{Decs}(r,s) = (r'',s'')}$

Risulta più leggibile quella a regole. Quindi useremo questa.

Espressioni

Exps ::= Exp Exps

$$\frac{[\![e]\!](r,s) = (v,s') \qquad [\![es]\!](r,s') = (lv,s'')}{[\![e\ es]\!](r,s) = (v\ lv,s'')}$$

Exp ::= Id

Il risultato è il valore della variabile (se inizializzata)

$$\frac{r(x) \in Loc}{[\![x]\!](r,s) = (s(r(x)),s)}$$

Questo formalizza anche l'intuizione che leggere il valore di una cella non altera la memoria

Exp ::= Id = Exp

$$\frac{[\![e]\!](r,s) = (v\ ,s')}{[\![x = e]\!](r,s) = (v,s'[v/r(x)])}$$

Exp ::= Id(Exps)

$$\frac{[\![es]\!](r,s) = (lv\ ,s')}{[\![f(es)]\!](r,s) = r(f)(lv,s')}$$

Questa è una funzione per via della correttezza statica

Scarichiamo le funzione base sull'interpretazione dei tipi base nell'algebra semantica

Exp ::= Exp + Exp

$$\frac{[\![e]\!](r,s) = (v,s') \qquad [\![e']\!](r,s') = (v',s'')}{[\![e + e']\!](r,s) = (a,s'')}$$

$v + v' = a$

Esercizio

Valutare la semantica della seguente espressione in ambiente r e stato s, assumendone la correttezza statica

x = f(3,x) + 2

$$\frac{\dfrac{\dfrac{}{r(f)(3\ s(r(x)), s) = (u_1, s_1)}\quad \dfrac{}{[3,x](r,s) = (3\ s(r(x)),\ s)}\quad \dfrac{}{[f(3,x)](r,s) = (u_1, s_1)}\quad \dfrac{}{[2](r,s_1) = (2, s_1)}}{[f(3,x)+2](r,s) = (u_1 + 2,\ s_1)}\quad v = u_1 + 2}{[x = f(3,x)+2](r,s) = (v, s_1[v/r(x)])}\quad v = u_1 + 2$$

$$\frac{}{[3](r,s) = (3, s)}\quad \frac{}{[x](r, s'') = (s''(r(x)), s'')}$$

$v_1 = 3$, $s''_1 = s$

$lv = 3\ s(r(x))$

$u_2 = 2$

$s' = s_1$

$v_2 = s''(r(x))$

$s'_1 = s''_1$

Concludendo $[x = f(3,x)+2](r,s) = (u_1 + 2,\ s_1[u_1 + 2 / r(x)])$

dove $(u_1, s_1) = r(f)(3\ s(r(x)), s)$

Convenzione: usiamo questa font per indicare valori ed operazioni nell'algebra semantica

Dichiarazioni di variabili

Decs ::= Dec Decs

$$\frac{[\![d]\!](r,s) = (r',s') \quad [\![ds]\!](r',s') = (r'',s'')}{[\![d\ ds]\!](r,s) = (r'',s'')}$$

Dec ::= Type Id

L'effetto deve essere di associare alla variabile una nuova locazione non inizializzata

$$\text{Nuova}(l,r,s) \quad \frac{}{[\![t\ x]\!](r,s) = (r[l/x], s[\square/l])}$$

Convenzione:
Dom(s[□/l])= Dom(s)□ {l}
s[□/l](y)= s(y) se y□DDef(s)
DDef(s[□/l])= DDef(s) (opzionale)

Dec ::= Type Id = Exp;

$$\text{Nuova}(l,r,s') \quad \frac{[\![e]\!](r,s) = (v\ ,s')}{[\![t\ x = e]\!](r,s) = (r[l/x], s'[v/l])}$$

Dichiarazioni di funzioni 1

Consideriamo il caso più semplice possibile: procedurale, senza ricorsione e con un solo parametro.

$$[\![\textbf{void } f(T\ x)\ \{sts\}]\!]$$

L'idea è: usare il parametro x per estendere ambiente (parametro formale) e lo stato al momento della chiamata (parametro attuale) e valutarvi il corpo ottenendo una funzione da associare a f nell'ambiente

$$[\![\textbf{void } f(T\ x)\ \{sts\}]\!]\ (r, s) = (r[F/f],s)$$

Lo stato non viene modificato.
La locazione per il parametro viene allocata al momento di ogni chiamata

Parametro attuale

Stato al momento della chiamata

Stato prodotto dalla chiamata

Dove $F : [\![T]\!] \to States \to \{*\} \to States$ è definita da

Maschero f, caso mai ci fosse nell'ambiente.
Per il momento è inutile, perché sappiamo che f non è ricorsiva

$$\frac{[\![sts]\!]\ (r[\square/f,l/x],s_c[v/l]) = s'}{F(v,s_c) = (*,s')}$$

Nuova(l,r,s_c)
l è nuova rispetto al momento della chiamata

Dichiarazioni di funzioni ricorsive

Si parte come nel caso non ricorsivo

$$[\![\textbf{void } f(T\ x)\ \{sts\}]\!](r,s) = (r[F/f],s)$$

Dove F : $[\![T]\!] \to States \to \{*\} \to States$

Il problema è la definizione di F:

non so come fare le chiamate ricorsive perché f non compare in r[□/f]

$$\frac{[\![sts]\!]\ (r[\square/f,l/x],s_c[v/l]) = s'}{F(v,s_c) = (*,s')}$$

Cambiamo approccio. F è definita induttivamente da tutte le regole della semantica più le seguenti due:

Caso base: riesco a valutare il corpo(come nel caso non ricorsivo)

$$\frac{[\![sts]\!]\ (r[\square/f,l/x],s_c[v/l]) = s'}{F(v,s_c) = (*,s')}\ Nuova(l,r,s_c)$$

Regola ad hoc per la chiamata di f

$$\frac{[\![e]\!](r[\square/f],s_0) = (u,s') \quad F(u,s') = (*,s'')}{[\![f(e)]\!]\ (r[\square/f],s_0) = (*,s'')}$$

Dichiarazioni di funzioni ricorsive 2

Cosa cambia nel caso in cui il tipo di ritorno non è **void**?

$$[\![\text{RT f(T x) \{sts } \textbf{result} \text{ e}\}]\!](r,s) = (r[F/f],s)$$

Dove F : $[\![T]\!] \to$ States $\to$ $[\![RT]\!] \to$ States

La definizione di F si fa analogamente al caso **void**

F è definita induttivamente da tutte le regole della semantica più le seguenti due:

Caso base: riesco a valutare il corpo(come nel caso non ricorsivo)

$$\frac{[\![sts]\!](r[\square/f,l/x],s_c[v/l]) = s' \qquad [\![e]\!](r[\square/f,l/x], s') = (a, s'')}{F(v,s_c) = (a,s'')} \; \text{Nuova}(l,r,s_c)$$

Regola ad hoc per la chiamata di f

$$\frac{[\![e]\!](r[\square/f],s_0) = (u, s') \qquad F(u,s') = (a, s'')}{[\![f(e)]\!](r[\square/f],s_0) = (a, s'')}$$

Statements

Stats ::= Stat Stats

$$\frac{[\![st]\!](r,s) = s' \qquad [\![sts]\!](r,s') = s''}{[\![st\ sts\ ]\!](r,s) = s''}$$

Stat ::= Exp;

Vogliamo considerare solo i side-effects della valutazione di un'espressione

$$\frac{[\![e]\!](r,s) = (v\ ,s')}{[\![e;]\!](r,s) = s'}$$

Stat ::= **if** (Exp) {Stats} **else** {Stats}

Scelta condizionale usuale

$$\frac{[\![e]\!](r,s) = (tt\ ,s') \qquad [\![sts]\!](r,s') = s''}{[\![\textbf{if}\ (e)\ \{sts\}\ \textbf{else}\ \{sts'\}]\!](r,s) = s''}$$

$$\frac{[\![e]\!](r,s) = (ff\ ,s') \qquad [\![sts']\!](r,s') = s''}{[\![\textbf{if}\ (e)\ \{sts\}\ \textbf{else}\ \{sts'\}]\!](r,s) = s''}$$

Stat ::= **while** (Exp) {Stats}

$$\frac{[\![e]\!](r,s) = (ff\ ,s')}{[\![\textbf{while}\ (e)\ \{sts\}]\!](r,s) = s'}$$

$$\frac{[\![e]\!](r,s) = (tt\ ,s') \qquad [\![sts]\!](r,s') = s'' \qquad [\![\textbf{while}\ (e)\ \{sts\}]\!](r,s'') = s_0}{[\![\textbf{while}\ (e)\ \{sts\}]\!](r,s) = s_0}$$

Input e Output

Finora abbiamo considerato solo programmi che modificano lo stato interno.
Vogliamo permettere anche operazioni di input e output.

Anzitutto bisogna introdurre i costrutti opportuni

Stat ::= ... **?**(Id) | **!**(Exp)

Poi dovremo dare le ovvie regole di semantica statica (per esercizio)

Ultimo problema: la semantica.

Bisogna aggiungere allo stato le componenti per l'interazione col mondo esterno:

States = Input × Output × $[\mathrm{Loc} \to_p \mathrm{Value}]_{\mathrm{Fin}}$

↑

stato (vecchio) interno

Nel nostro caso si leggono valori da assegnare alle variabili e si scrivono valori (valutazioni delle espressioni), quindi

$$\mathrm{Input} = \mathrm{Output} = \mathrm{Value}^*$$

Perché le regole vecchie abbiano senso bisogna estendere la notazione s[v/l] ai nuovi stati

$$(i,o,m)[v/l]=(i,o,m[v/l])$$

$$\frac{[\![e]\!](r,s) = (v,(i,o,m))}{[\![\mathbf{!}(e)]\!](r,s) =(i,\ v\bullet o,m)}$$

$$\frac{}{[\![\mathbf{?}(x)]\!](r,(v\bullet\ i,o,m)) =(i,o,m[v/r(x)])}$$