Semantica denotazionale

Un programma p denota una funzione f_p che applicata ai dati di ingresso produce i risultati; tale funzione ne definisce il significato o denotazione:

f_p: Dati $\rightarrow$ Risultati

Si definisce funzione di denotazione una funzione ∂ tale che, per ogni programma p, $\partial[p] = f_p$:

∂: Progr $\rightarrow$ {Dati $\rightarrow$ Risultati}

$\partial[p](d) = r$, con $d \in$ Dati, $r \in$ Risultati

La denotazione di un programma può essere costruita componendo funzionalmente le denotazioni delle singole istruzioni o strutture sintattiche.

Esempio 1

Sia data la grammatica G:

- Program ::= Program Frase | Frase
- Frase ::= read id | id := const | write id

Domini sintattici:

P = **L**(G)

F = $\mathbf{L}_{\text{Frase}}$(G)

R: insieme delle frasi "read id"

A: insieme delle frasi "id := const"

W: insieme delle frasi "write id"

Domini semantici:

$\Im$: insieme dei numeri interi

Id : insieme degli identificatori

Mem = {**Id** $\rightarrow \Im$} : ogni elemento $m \in$ **Mem** rappresenta una mappa di memoria: m(id) = i, id $\in$ **Id** , i $\in \Im$ (si ignori il caso m(id) = $\perp$)

In = $\Im^*$: sequenza di ingresso

Out = $\Im^*$: sequenza di uscita

Stati = **In** $\times$ **Mem** $\times$ **Out**

Denotazione di $p \in P$

$\partial[p]$: **Stati** $\rightarrow$ **Stati**

Funzioni ausiliarie:

- head(x): **In** $\rightarrow \Im$; estrae il primo elemento della sequenza x
- tail(x): **In** $\rightarrow$ **In**; estrae la coda della sequenza x
- update : **Id** $\times \Im \times$ **Mem** $\rightarrow$ **Mem**;

update(v,i, m) = m' , m'(a) = m(a) $\forall a \neq v$

m'(v) = i

Denotazione di f $\in$ R; f = "read id"

$\partial[f]$: **In** $\times$ **Mem** $\rightarrow$ **In** $\times$ **Mem**

∂[read id](x, m) = (x',m') ,

x' = tail(x)

m' = update(id, head(x),m)

Denotazione di $f \in A$; f = "id := const"

$\partial[f] : \mathbf{Mem} \rightarrow \mathbf{Mem}$

∂[id := const](m) = m', m' = update(id, const,m)

Denotazione di $f \in W$; f = "write id"

$\partial[f] : \mathbf{Mem} \times \mathbf{Out} \rightarrow \mathbf{Out}$

∂[write id](m, x) = x', x' = x $\circ$ m(id)

Denotazione di $f \in F$

$\partial[f] : \mathbf{Stati} \rightarrow \mathbf{Stati}$

$\partial[f](x,m,y) =$ (∂[read id](x,m), y) | (x, ∂[id := const](m), y) | (x, m, ∂[write id])

Denotazione $\partial[p_1; p_2] = \partial[p_2] \circ \partial[p_1]$

x = (3 13 11 12)
m = {(a,0),(b,0)}
y = ε
p = read a; read b; write a; a := 2; write a;

∂[p](x,m,y) = ∂[write a](∂[a:=2](.. ∂[read a](x,m,y) ..))
= (x',m',y')

x' = (11 12)
m' = {(a,2),(b,13)}
y' = (3 2)

Controllo degli errori semantici

Gli errori semantici di classificano in:

- statici: non dipendenti dallo stato dell'esecuzione;
- dinamici: dipendenti dallo stato dell'esecuzione.

Nelle definizioni semantiche, gli errori possono essere descritti mediante appositi predicati.

Esempio 2 : errori statici

Sia data la grammatica G:

- Program ::= Program Frase | Frase
- Frase ::= id := Expr | if Expr then Program else Program
- Expr ::= Expr + Expr | Expr = Expr | id | intconst

Domini semantici:

$\mathbf{B}$ = {*true* , *false*}
$\mathfrak{I}$: insieme dei numeri interi
$\mathbf{Id}$: insieme degli identificatori
$\mathbf{V} = \mathbf{B} \cup \mathfrak{I}$
$\mathbf{Mem} = \{\mathbf{Id} \rightarrow (\mathbf{V} \cup \{\perp\})\}$
$\mathbf{Ris} = \mathbf{V} \cup$ {*errore* }
$\mathbf{Stati} = \mathbf{Mem} \cup$ {*errore* }

Denotazione di $p \in P$, $P = \mathbf{L}(G)$

$\partial[p] : \mathbf{Stati} \rightarrow \mathbf{Stati}$

NOTA: ogni situazione di errore deve essere classificata e descritta; ciascuna di esse deve essere identificata e propagata attraverso le successive denotazioni.

- update : $\mathbf{Id} \times \mathbf{Ris} \times \mathbf{Stati} \rightarrow \mathbf{Stati}$;

Denotazione di $f \in F$, $F = \mathbf{L}_{Frase}(G)$

$\partial[f] : \mathbf{Stati} \rightarrow \mathbf{Stati}$

- $\partial[f](s) =_{def}$

```
if s = errore
      then errore
    else
      case f of
"id := expr":   if ∂[expr](s) = errore
                    then errore
                    else update(id, ∂[expr](s), s)
                endif
"if expr then p1 else p2":
                if (∂[expr](s) ∈ B )
                    then if ∂[expr](s)
                         then ∂[p1](s)
                         else ∂[p2](s)
                    endif
                    else errore
                endif
              endcase
endif
```

Denotazione di $e \in E$, $E = \mathbf{L}_{Expr}(G)$

$\partial[e] : \mathbf{Stati} \rightarrow \mathbf{Ris}$

• $\partial[e](s) =_{def}$

if s = *errore*
then *errore*
else
case e **of**

"$e_1 + e_2$": **if** $(\partial[e_1](s) \notin \Im)$ **or** $(\partial[e_2](s) \notin \Im)$
then *errore*
else plus($\partial[e_1](s)$, $\partial[e_2](s)$)
endif

"$e_1 = e_2$": **if** $(\partial[e_1](s) \in \Im)$ **and** $(\partial[e_2](s) \in \Im)$
or
$(\partial[e_1](s) \in \mathbf{B})$ **and** $(\partial[e_2](s) \in \mathbf{B})$
then equal($\partial[e_1](s)$, $\partial[e_2](s)$)
else *errore*
endif

"id": **if** s(id) = $\perp$
then *errore*
else s(id)
endif

"intconst": intconst

endcase
endif

Definizioni ricorsiva dei domini sintattici

Una produzione ricorsiva nella forma $\langle A^1 ::= A^2\ B \rangle$ descrive oggetti sintattici aventi struttura:

$\alpha^1 = \alpha^2 \circ \beta$, con $\alpha^1 \in A^1$, $\alpha^2 \in A^2$, $\beta \in B$

la cui denotazione è nella forma: $\partial[\alpha^1] = \partial[\beta] \circ \partial[\alpha^2]$ (scritta anche $\partial[A^1] = \partial[B] \circ \partial[A^2]$)

Per ogni albero sintattico finito, il numero di applicazioni della ricorsione è finito.

Definizioni ricorsiva dei domini semantici

Se il linguaggio presenta dichiarazioni di procedure e/o funzioni, la definizione dei domini semantici può risultare ricorsiva.

Esempio

Sia data la grammatica G:

• Program ::= Program Frase | Frase
• Frase ::= id := Expr | if Expr then Program | call id
• Expr ::= - Expr | Expr < Expr | id | intconst |
procedure Program end

Domini semantici:

B = {*true* , *false*}
ℑ : insieme dei numeri interi
Id : insieme degli identificatori
Mem = {**Id** → **V**}
V = **B** ∪ ℑ ∪ **P**
P = {**Mem** → **Mem**}: ogni procedura è una
funzione che ridefinisce una mappa di memoria

Denotazione di p ∈ P, P = **L**(G)
∂[p] : **Mem** → **Mem**

• update : **Id** × **V** × **Mem** → **Mem**;

Denotazione di f ∈ F, F = $\mathbf{L}_{Frase}$(G)
∂[f] : **Mem** → **Mem**

• ∂[f](m) =$_{def}$

case f **of**
"id := expr": update(id, ∂[expr](m), m)
"if expr then p_1": **if** ∂[expr](m)
then ∂[p_1](m)
else m
endif
"call id": m[id](m)
endcase

Denotazione di e ∈ E, E = $\mathbf{L}_{Expr}$(G)
∂[e] : **Mem** → **V**

• ∂[e](m) =$_{def}$

case e **of**
"- e_1": minus(∂[e_1](m))
"e_1 < e_2": lth(∂[e_1](m), ∂[e_2](m))
"id": m(id)
"intconst": intconst
"procedure p end": ∂[p]
endcase

• p = r := procedure x := y < 7 end; y := 1; call r;
• e = procedure x := y < 7 end

∂[p](m) = m'[r](m')
m'= update(y, 1, update(r, ∂[x := y<7], m))
∂[x := y<7] = "update(x, lth(m[y],7), m)"

m_0 = (x = ⊥ , y = ⊥ , r = ⊥)
•update(r, ∂[e](m_0), m_0)
m_1 = (x = ⊥ , y = ⊥ , r = "update(x, lth(m[y],7), m)")
•update(y, ∂[1](m_1), m_1)
m_2 = (x = ⊥ , y = 1 , r = "update(x, lth(m[y],7), m)")
•m_2[r](m_2)
m_3 = (x = *true* , y = 1 , r = "update(x, lth(m[y],7), m)")

Funzioni denotate ricorsive

Una funzione denotata può essere definita ricorsivamente.

Esempio

Sia data la grammatica G:

- Program ::= Program Frase | Frase
- Frase ::= id := Expr | if Expr then Program endif | call id
- Expr ::= Expr > Expr | Expr - Expr | id | intconst | procedure Program end

Domini semantici:

B = {*true* , *false*}
$\mathfrak{I}$: insieme dei numeri interi
Id : insieme degli identificatori
Mem = {**Id** → **V**}
V = **B** ∪ $\mathfrak{I}$ ∪ **P**
P = {**Mem** → **Mem**}: ogni procedura è una funzione che ridefinisce una mappa di memoria

Denotazione di p ∈ P, P = **L**(G)
∂[p] : **Mem** → **Mem**

Denotazione di e ∈ E, E = $\mathbf{L}_{Expr}$(G)

∂[e] : **Mem** → **V**

- ∂[e](m) =$_{def}$

case e **of**
"$e_1 > e_2$": gth(∂[e_1](m), ∂[e_2](m))
"e_1 - e_2": minus(∂[e_1](m), ∂[e_2](m))
"id": m(id)
"intconst": intconst
"procedure p end": ∂[p]
endcase

definizione ricorsiva di un ciclo **while**:

p = while := procedure
if x>0 then x:= x-1; call while endif
end;
call while

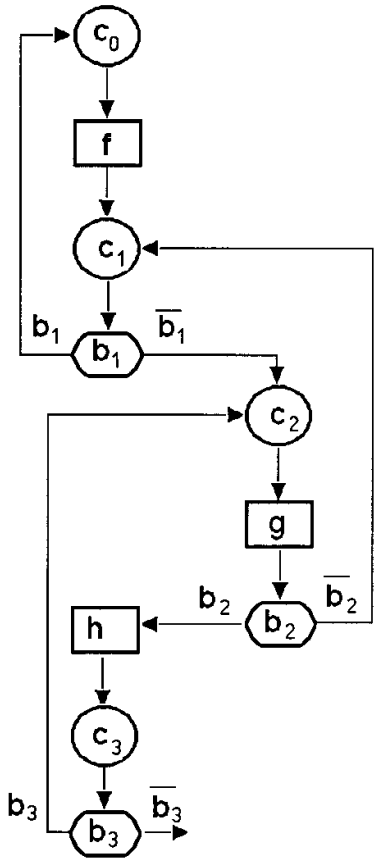
p'= if x>0 then x:= x-1; call while endif

∂[p](m) = m'[while](m')
m'= update(while, ∂[p'], m)
∂[p'] = "**if** ∂[x>0](m)
then {∂[x:= x-1](m)}[while]{∂[x:= x-1](m)}
else m
endif"

Continuazioni

La semantica di un programma contenente salti di controllo può essere definita mediante l'uso di continuazioni.

Esempio



- f, g, h: **Stati** → **Stati**

- b_i : **Stati** → (**Val** × **Stati**)
  $b_i(Q).v \in$ **Val** **Val** = {*true* , *false*}
  $b_i(Q).s \in$ **Stati**

- c_i : **Stati** → **Stati_finali** (Continuazioni)

$c_0(Q) = c_1(f(Q))$
$c_1(Q) =$ **if** $b_1(Q).v$
then $c_0(b_1(Q).s)$
else $c_2(b_1(Q).s)$
end if
$c_2(Q) =$ **if** $b_2(g(Q)).v$
then $c_3(h(b_2(g(Q)).s)$
else $c_1(b_2(g(Q)).s)$
end if
$c_3(Q) =$ **if** $b_3(Q).v$
then $c_2(b_3(Q).s)$
else $b_3(Q).s$
end if

Il significato del programma è l'effetto prodotto partendo dal punto di entrata, ossia la continuazione c_0