Costrutto di iterazione determinata

Enrico Sorbera

13 luglio 2020

Presentazione del costrutto

$$\textbf{for } id = exp_I \textbf{ to } exp_F \textbf{ do } cmd$$

Dove exp_I ed exp_F sono espressioni di tipo intero, id è una variabile di tipo intero, inizializzata a v_I, valore di exp_I, calcolato prima della prima iterazione. Il costrutto ripete il comando cmd, che non può modificare id, un numero di volte pari a $v_F - v_I$, dove v_F è il valore di exp_F. Ad ogni iterazione il valore di id viene incrementato di 1.

Considerazioni preliminari

Osserviamo che uno dei comandi di Small20 realizza già un ciclo for:
For of stm * exp * stm * cmd, il nuovo for che andremo ad aggiungere è un costrutto di iterazione determinata, si differenzia dunque dal precedente perché fissa prima della prima iterazione il numero di iterazioni da compiere e garantisce la terminazione.

Considerazioni preliminari

Mostriamo un esempio di programma per cui il for non determinato non garantisce la terminazione:

```
let k=N 0;;
let dc= StmD(Var(Int,"tot",N 0)) in
let dy=StmD(Var(Int,"y",N 4)) in
let di= Var(Int,"i",k) in
let d= Upd(Val("tot"),Plus(Val("tot"),N 1)) in
let c= For(StmD(di),LT(Val("i"),Upd(Val("y"),Plus(Val("y"),N 1))),
       StmC(Cmd(Upd(Val("i"),Plus(Val("i"),N 1)))),Cmd(d)) in
let p=Prog("ex",Seq[dc;dy;StmC(c)]) in
let q= printProg p in
 run p;;
```

```
Program ex{
   var int tot = 0;
   var int y = 4;
   for (var int i = 0;(i < (y = (y + 1)));i = (i + 1))
              tot = (tot + 1);
}
```

Considerazioni preliminari

A ogni iterazione il comando for rivaluterà l'espressione $y = y + 1$, incrementando così, ogni volta, il valore di y, che dunque non sarà mai raggiunta da i, portando ad una mancata terminazione.
Il for determinato, invece, calcolando l'espressione solo al primo passo, compirà 6 iterazioni, restituendo un valore di tot pari a 6, coerentemente a quello che ci aspettavamo.

Sintassi del costrutto

- **Sintassi astratta:**

```
Cmd::= ...
     | [for2] Ide Exp Exp Cmd
     | [iter] Loc Num Cmd
```

- **Sintassi concreta:**

```
Cmd::=  ...
     | for  Ide = Exp1 to Exp2 do Cmd
```

Strutture esprimibili

Con l'aggiunta di questo costrutto possiamo esprimere ogni programma che ripeta un comando (a meno che questo non modifichi l'identificatore del nostro costrutto) un numero di volte dato dalla differenza di due espressioni, purché di tipo intero.
Alcuni esempi possono essere il calcolo del fattoriale di una certa espressione, l'ordinamento di un vettore ed il programma riportato sopra, di cui in seguito vedremo il funzionamento.

Aggiornamento di $\mathtt{Sem}_{\mathtt{CMD}}$

- C.10:

$$\frac{\begin{array}{c}\langle \mathtt{e_I},(\rho,\mu)\rangle \rightarrow \lfloor [\mathtt{int}],\mathtt{v_I},(\rho_\mathtt{I},\mu_\mathtt{I})\rfloor \\ \langle \mathtt{e_F},(\rho_\mathtt{I},\mu_\mathtt{I})\rangle \rightarrow \lfloor [\mathtt{int}],\mathtt{v_F},(\rho_\mathtt{F},\mu_\mathtt{F})\rfloor \\ \triangleright(\mu_\mathtt{F},1)=(\mathtt{loc},\mu_\mathtt{s}) \\ [\mathtt{id}/([\mathtt{mut}][\mathtt{int}],\mathtt{loc})]\circ\rho_\mathtt{F} \;=\; \rho_\mathtt{id} \\ \mu_\mathtt{s}[\mathtt{loc}\leftarrow \mathtt{v_I}]=\mu_\mathtt{id} \\ \mathtt{v_F}-\mathtt{v_I}=\mathtt{n} \\ \langle [\mathtt{iter}]\ \mathtt{loc}\ \mathtt{n}\ \mathtt{c},\ (\rho_\mathtt{id},\mu_\mathtt{id}))\rangle \rightarrow ([\mathtt{void}],(\rho_\mathtt{f},\mu_\mathtt{f}))\end{array}}{\langle [\mathtt{for2}]\ \mathtt{id}\ \mathtt{e_I}\ \mathtt{e_F}\ \mathtt{c},(\rho,\mu)\rangle \rightarrow ([\mathtt{void}],(\rho_\mathtt{f},\mu_\mathtt{f}))}$$

- C.11 ...
- C.12 ...

Aggiornamento di $\mathtt{Sem}_{\mathtt{CMD}}$

- C.10: ...
- C.11

$$\frac{\begin{array}{c}\mathtt{n}\geq 0,\ \sigma(\rho,\mu)\\ \langle \mathtt{c},(\rho,\mu)\rangle \rightarrow ([\mathtt{void}],(\rho_{\mathtt{c}},\mu_{\mathtt{c}}))\\ \mu_{\mathtt{c}}(\mathtt{loc})=\mu(\mathtt{loc}),\ \mathtt{n}-1=\mathtt{n}_1\\ \mu[\mathtt{loc}\leftarrow\mu(\mathtt{loc})+1]=\mu_1\\ \langle [\mathtt{iter}]\ \mathtt{loc}\ \mathtt{n}_1\ \mathtt{c},\ (\rho_{\mathtt{c}},\mu_1)\rangle \rightarrow ([\mathtt{void}],(\rho_{\mathtt{F}},\mu_{\mathtt{F}}))\end{array}}{\langle [\mathtt{iter}]\ \mathtt{loc}\ \mathtt{n}\ \mathtt{c},(\rho,\mu)\rangle \rightarrow ([\mathtt{void}],(\rho_{\mathtt{F}},\mu_{\mathtt{F}}))}$$

- C.12

$$\frac{\mathtt{n}<0}{\langle [\mathtt{iter}]\ \mathtt{loc}\ \mathtt{n}\ \mathtt{c},\ (\rho,\mu)\rangle \rightarrow ([\mathtt{void}],(\rho,\mu))}$$

Semantica e Sistema Y

Aggiornamento del sistema Y:

- Y : 22:

$$\frac{\begin{array}{c}\langle \mathtt{e_I}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{int}], \mathtt{Y}_\rho), \\ \langle \mathtt{e_F}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{int}], \mathtt{Y}_\rho), \\ \langle [\mathtt{iter}]\ \mathtt{loc\ n\ c}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} \langle [\mathtt{void}], \mathtt{Y}_\rho \rangle\end{array}}{\langle [\mathtt{for2}]\ \mathtt{id}\ \mathtt{e_I}\ \mathtt{e_F}\ \mathtt{c}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{void}], \mathtt{Y}_\rho)}$$

- Y : 23:

$$\frac{\langle \mathtt{c}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{void}], \mathtt{Y}_\rho)}{\langle [\mathtt{iter}]\ \mathtt{loc\ n\ c}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{void}], \mathtt{Y}_\rho)}$$

Semantica e Sistema Y

- Che portano agli errori di gestione di tipo $\mathtt{E}:37$:

$$\frac{\langle \mathtt{e_I}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}, \mathtt{Y}_\rho),\ \mathtt{t} \neq \mathtt{Int}}{\langle [\mathtt{for2}]\ \mathtt{id}\ \mathtt{e_I}\ \mathtt{e_F}\ \mathtt{c}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

- di $\mathtt{E}:38$:

$$\frac{\begin{array}{c}\langle \mathtt{e_I}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{int}], \mathtt{Y}_\rho) \\ \langle \mathtt{e_F}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}, \mathtt{Y}_\rho),\ \mathtt{t} \neq \mathtt{Int}\end{array}}{\langle [\mathtt{for2}]\ \mathtt{id}\ \mathtt{e_I}\ \mathtt{e_F}\ \mathtt{c}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

- e di $\mathtt{E}:39$:

$$\frac{\langle \mathtt{c}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}{\langle [\mathtt{iter}]\ \mathtt{loc}\ \mathtt{n}\ \mathtt{c}, \mathtt{Y}_\rho \rangle \rightarrow_{\mathtt{Y}} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

A questi dobbiamo aggiugere l'errore dovuto al caso in cui il comando all'interno del for modifichi id.
Caso trattato nell'implementazione, detto $\mathtt{E}:36$.

Implementazione

Sintassi astratta:

```
cmd = ...
      | For2 of ide * exp * exp * cmd
      | Iter of ide * num * cmd
```

Sintassi concreta:

```
toStringCmd tab = ...
        | For2(ide,exp,exp2,cmd) -> ((indent tab)
          ^ "for " ^(toStringI ide ) ^ " = "
              ^(toStringExp exp)^ " to "
              ^ (toStringExp exp2) ^ "do"
              ^ (toStringCmd (tab) cmd))
        | Iter(ide,num,cmd)-> " "
```

Implementazione

```
| For2(id,expi,expf,c)->
    (match expSem expi (rho,mu) with
                |(Int,Ival(vi),(rhoi,mui)) ->
                (match expSem expf (rhoi,mui) with
                     |(Int,Ival(vf),(rhof,muf)) ->
                        (let (loca,mua) = allocate muf 1 in
                      let den = DVar((Mut Int),loca) in
                       let rhoid = bind rhof id den in
                       let v= Ival(vi) in
                      let muid = upd mua loca (eTOm v) in
                      let n= vf - vi in
                      let c1= Iter(id,n,c) in
                         cmdSem c1 (rhoid,muid))
                     | _ -> (let msg = "E38: cmdSem - "
                                       ^ (toStringExp expf)
                                       ^ " deve essere un intero" in
                                         raise(TypeErrorC msg)))
                | _ -> (let msg = "E37: cmdSem - "
                                  ^ (toStringExp expi)
                                  ^ " deve essere un intero" in
                                        raise(TypeErrorC msg)))
```

Implementazione

```
| Iter(id,n,c)->
    if(n>=0) then
     (match getEnv rho id with
      | DVar(ty,loc) ->
            (match getStore mu loc with
              |(Mint(v))->
                (match cmdSem c (rho,mu) with
                  |(Void,(rhoc,muc))->
                    (if(getStore muc loc=Mint(v)) then
                      (let n1=n - 1 in
                       let v1=v+1 in
                       let va=Mint(v1) in
                       let mu2= upd muc loc va in
                       let c1=Iter(id,n1,c) in
                           cmdSem c1 (rhoc,mu2))
                       else (let msg="E:36: cmdSem-"
                                  ^ (toStringCmd 0 c)
                                  ^ " influenza "
                                  ^ (toStringI id) in
                                          raise(TypeErrorC msg)))
                    | _ -> (let msg = "E29: cmdSem - "
                                   ^ (toStringCmd 0 cmd) in
                                          raise(TypeErrorC msg)))
                |_->(let msg="Error" in
                           raise(TypeErrorC msg)))
             | _ -> (let msg = "E40: cmdSem - "
                          ^ (toStringI id)
                          ^ "non indica una variabile " in raise(TypeErrorC msg)))

            else (Void,(rho,mu))
```

Verifica del codice ed esempi

Verifichiamone il corretto funzionamento partendo da un programma piuttosto semplice, che calcola il fattoriale di un numero k:

```
let sd= StmD(Var(Int, "k", N 5)) ;;
let sd1= StmD(Var(Int, "r", N 1)) ;;
let id= "j";;
let e1=N 1;;
let e2= Val("k");;
let vr=Val("r");;
let vj=Val("j");;
let d2= Upd(vr, Mul(vr,vj)) ;;
let c1=For2(id,e1,e2,Cmd(d2)) in
let p=Prog("fact",Seq[sd;sd1;StmC(c1)]) in
let q= printProg p in
run p;;
```

Verifica del codice ed esempi

```
Program fact{
    var int k = 5;
    var int r = 1;
    for j = 1 to k do     r = (r * j);
}
```

```
=============== Traccia del Programma fact ===========
=====================================================
Stack: [j/(int,L2); r/(int,L1); k/(int,L0)]
Store: [L0<-5,L1<-120,L2<-6]
=====================================================
==================== Traccia: Fine ==================
```

Verifica del codice ed esempi

Passiamo alla verifica dell'esempio riportato tra le considerazioni preliminari, adattato al nuovo for determinato:

```
let k=N 0;;
let dc= StmD(Var(Int,"tot",N 0)) in
let dy=StmD(Var(Int,"y",N 4)) in
let di= Var(Int,"i",k) in
let d= Upd(Val("tot"),Plus(Val("tot"),N 1)) in
let c= For2("i",N 0,Upd(Val("y"),Plus(Val("y"),N 1)),Cmd(d)) in
let p=Prog("ex",Seq[dc;dy;StmC(c)]) in
let q= printProg p in
 run p;;
```

Verifica del codice ed esempi

Come era possibile prevedere, il for determinato garantisce la terminazione:

```
Program ex{
    var int tot = 0;
    var int y = 4;
    for i = 0 to (y = (y + 1)) do     tot = (tot + 1);
}
```

```
============== Traccia del Programma ex ==========
================================================
Stack: [i/(int,L2); y/(int,L1); tot/(int,L0)]
Store: [L0<-6,L1<-5,L2<-6]
================================================
==================== Traccia: Fine ================
```

Verifica del codice ed esempi

Ecco infine aluni esempi di errori:

- E::37 (e_f non ha tipo intero)

```
1              let dc= StmD(Var(Int,"tot",N 0)) in
2              let u= Upd(Val("tot"),Plus(Val("tot"),N 1)) in
3              let c=For2("i",B False,N 7, Cmd(u)) in
4              let p=Prog("e37",Seq[dc;StmC(c)]) in
5              let q= printProg p in
6              run p;;
7
8              Program e37{
9      var int tot = 0;
10     for i = false to 7 do    tot = (tot + 1);
11 }
12
13 Exception: TypeErrorC "E37: cmdSem - false deve essere un intero".
```

Verifica del codice ed esempi

- E::38 (e_F non ha tipo intero)

```
1             let dc= StmD(Var(Int,"tot",N 0)) in
2             let u= Upd(Val("tot"),Plus(Val("tot"),N 1)) in
3             let e2=Eq(N 7, Plus(N 5, N 2)) in
4             let c=For2("i",N 0,e2, Cmd(u)) in
5             let p=Prog("e38",Seq[dc;StmC(c)]) in
6             let q= printProg p in
7             run p;;
8
9             Program e38{
10       var int tot = 0;
11       for i = 0 to (7 == (5 + 2)) do    tot = (tot + 1);
12    }
13
14    Exception: TypeErrorC "E38: cmdSem - (7 == (5 + 2)) deve essere un intero".
```

Verifica del codice ed esempi

- E::36 (Cmd modifica id)

```
1              let dc= StmD(Var(Int,"tot",N 0)) in
2              let u= Upd(Val("i"),Plus(Val("tot"),N 1)) in
3              let e2=N 8 in
4              let u2= IfT(GT(Val("i"),N 5), Cmd(u)) in
5              let c=For2("i",N 0,e2, u2) in
6              let p=Prog("e36",Seq[dc;StmC(c)]) in
7              let q= printProg p in
8              run p;;
9
10  Program e36{
11     var int tot = 0;
12     for i = 0 to 8 do    if (i > 5) i = (tot + 1);
13  }
14
15  Exception: TypeErrorC "E:36: cmdSem-if (i > 5) i = (tot + 1) modifica i".
```