Costrutto di Dichiarazione multipla

Emanuele Angilé

Università di Pisa

22 giugno 2020

Il costrutto

$$\texttt{t ide}_1 = \texttt{exp}_1, ..., \texttt{ide}_k = \texttt{exp}_k$$

Il costrutto di dichiarazione multipla permette di dichiarare $k > 0$ variabili e inizializzarle ad altrettante espressioni denotabili tutte dello stesso tipo $\texttt{t} \in \texttt{Simple}$, con `Simple={[Int], [Bool]}`.

Permette inoltre di utilizzare all'interno delle espressione $\texttt{exp}_j$ le variabili $\texttt{ide}_1, ..., \texttt{ide}_{j-1}$ fino a quel momento inizializzate.

Il Costrutto in Small20

- Modifiche in Sintassi Astratta:

```
Dcl2 ::= [Decl] Ide Exp
Dcl2Seq ::= Dcl2 | Dcl2[x]Dcl2Seq
Dcl ::=...| [VarM] Type Dcl2Seq
```

- Modifiche in Sintassi Concreta:

```
Dcl2 ::= ide=Exp
Dcl2Seq ::= Dcl2|Dcl2,Dcl2Seq
Dcl ::=...|var Simple ide=Exp
          |varm Simple Dcl2Seq
```

- Non è stata apportata alcuna modifica alla macchina astratta AM20.

Implementazione in Ocaml della Sintassi

Sintassi Astratta

```
type dcl2=
        Decl of ide*exp
        and
dcl2Seq=
        dcl2 list
        and
dcl =
        ...
        | VarM of tye* dcl2Seq
```

Sintassi concreta

```
let toStringDcl2= (function
                |Decl(ide,exp)-> (toStringI ide) ^ "=" ^ (toStringExp exp))
                and
toStringDcl2Seq l= match l with
        []->""
        |h::t when not(ysame t [])-> (toStringDcl2 h)^ ", " ^(toStringDcl2Seq t)
        |h::t->(toStringDcl2 h);;

toStringDcl = (function
                ...
                | VarM(tye,dcl2)->
                  "varm " ^ (toStringTye tye) ^ " " ^(toStringDcl2Seq dcl2);;
```

Semantica e Sistema Y per Small20

- Aggiornamento delle regole del sistema $\mathtt{Sem}_{\mathtt{DCL}}$ con l'aggiunta di D5:

$$\frac{\begin{array}{c}\langle \mathtt{e}_1,(\rho_0,\mu_0)\rangle \to \langle \mathtt{t}_{\mathtt{e}_1},\mathtt{v}_1,(\rho_0,\mu_{\mathtt{e}_1})\rangle \quad \mathtt{t}=\mathtt{t}_{\mathtt{e}_1} \quad \mathtt{t}\in\mathtt{Simple} \\ \triangleright(\mu_{\mathtt{e}_1},1)=(\mathtt{loc}_{\mathtt{a}_1},\mu_{\mathtt{a}_1}) \quad [\mathtt{I}/([\mathtt{mut}]\mathtt{t}_{\mathtt{e}_1},\mathtt{loc}_{\mathtt{a}_1})]\circ\rho_0=\rho_1 \quad \mu_{\mathtt{a}_1}[\mathtt{loc}_{\mathtt{a}_1}\leftarrow\mathtt{v}_1]=\mu_1 \\ \ldots \\ \langle \mathtt{e}_k,(\rho_{k-1},\mu_{k-1})\rangle \to \langle \mathtt{t}_{\mathtt{e}_k},\mathtt{v}_k,(\rho_{k-1},\mu_{\mathtt{e}_k})\rangle \quad \mathtt{t}=\mathtt{t}_{\mathtt{e}_k} \\ \triangleright(\mu_{\mathtt{e}_k},1)=(\mathtt{loc}_{\mathtt{a}_k},\mu_{\mathtt{a}_k}) \quad [\mathtt{I}/([\mathtt{mut}]\mathtt{t}_{\mathtt{e}_k},\mathtt{loc}_{\mathtt{a}_k})]\circ\rho_{k-1}=\rho_k \quad \mu_{\mathtt{a}_k}[\mathtt{loc}_{\mathtt{a}_k}\leftarrow\mathtt{v}_k]=\mu_k\end{array}}{\langle [\mathtt{varm}]\ \mathtt{t}\ \mathtt{I}_1\mathtt{e}_1\ldots\mathtt{I}_k\mathtt{e}_k,(\rho_0,\mu_0)\rangle \to ([\mathtt{void}],(\rho_k,\mu_k))}$$

- Aggiornamento del sistema Y:

$$\mathtt{Y5}: \frac{\langle \mathtt{e}_i,\mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}_i,\mathtt{Y}_\rho)\ \forall 1\leq i\leq k \qquad \mathtt{t}\in\mathtt{Simple} \qquad \mathtt{t}_1=\cdots=\mathtt{t}_k=\mathtt{t}}{\langle [\mathtt{varm}]\ \mathtt{t}\ \mathtt{I}_1\mathtt{e}_1\ldots\mathtt{I}_k\mathtt{e}_k,\mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} ([\mathtt{void}],[\mathtt{I}_n/[\mathtt{mut}]\mathtt{t}]\circ(\cdots\circ([\mathtt{I}_1/[\mathtt{mut}]\mathtt{t}]\circ\mathtt{Y}_\rho)))}$$

Gestione degli errori di tipo

Con la regola Y5 appena aggiunta, gli errori di tipo possibili sono sostanzialmente due:

- il primo si ha se il tipo t dichiarato non è Simple

$$\text{E8:}\frac{\mathtt{t} \notin \mathtt{Simple}}{\langle [\mathtt{varm}]\ \mathtt{t}\ \mathtt{I}_1\mathtt{e}_1 \ldots \mathtt{I}_\mathtt{k}\mathtt{e}_\mathtt{k}, \mathtt{Y}_\rho \rangle \rightarrow_\mathtt{Y} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

- il secondo è dato dalla presenza di almeno un' espressione $\mathtt{e_i}$ la cui valutazione sia di tipo $\mathtt{t_i}$ diverso dal tipo t dichiarato

$$\text{E9:}\frac{\langle \mathtt{e_i}, \mathtt{Y}_\rho \rangle \rightarrow_\mathtt{Y} (\mathtt{t_i}, \mathtt{Y}_\rho)\ \forall 1 \leq \mathtt{i} \leq \mathtt{k} \quad \exists\ \mathtt{i} : \mathtt{t_i} \neq \mathtt{t}}{\langle [\mathtt{varm}]\ \mathtt{t}\ \mathtt{I}_1\mathtt{e}_1 \ldots \mathtt{I}_\mathtt{k}\mathtt{e}_\mathtt{k}, \mathtt{Y}_\rho \rangle \rightarrow_\mathtt{Y} ([\mathtt{terr}], \mathtt{Y}_\rho)}$$

Implementazione in ocaml

Inserimento del caso [varm] in Sem_{DCL}:

```
let rec dclSem dcl (rho,(Store(d,g)as mu)) =
            | VarM (ty, dcl2) ->
            (match dcl2 with
                    |Decl(ide,exp)::tail->
                    (match expSem exp (rho,mu) with
                      |(te,v,(rhoe,mue))
                      when (isSimple ty) && (ysame ty te)
                                  -> (let (loca,mua) = allocate mue 1 in
                                      let den = DVar((Mut ty),loca) in
                                      let rhoF = bind rhoe ide den in
                                      let muF = upd mua loca (eTOm v) in
                                      dclSem(VarM(ty,tail))(rhoF,muF))
                       |(te,v,(rhoe,mue))
                       when (isSimple ty)
                                   -> raise(TypeErrorI("E9: dclSem",ide))
                       |_ -> raise(TypeErrorI("E8: dclSem",ide)))
                    |[]->(Void,(rho,mu)))
```

Primo esempio di uso del costrutto

```
let dcl1= VarM (Int,[Decl ("id1",N 30);Decl ("id2",Div (Val "id1",N 5))]) in
let dcl2= VarM (Bool,[Decl ("bl1",B True);Decl ("bl2",GT(Val"id2",N 10))]) in
let p=Prog("ex1",Seq[StmD dcl1; StmD dcl2]) in
let s = toStringProg p in
let _ = printf "\n%s\n" s in
run p;;
```

```
Program ex1{
   varm int id1=30, id2=(id1 / 5);
   varm bool bl1=true, bl2=(id2 > 10);
}

=============== Traccia del Programma ex1 ===========
====================================================
 Stack: [bl2/(bool,L3); bl1/(bool,L2); id2/(int,L1); id1/(int,L0)]
 Store: [L0<-30,L1<-6,L2<-true,L3<-false]
====================================================
==================== Traccia: Fine ==================
```

Errori di tipo

- Esempio di errore 9 ($\exists$ i : $t_i \neq t$)

```
1       let dclerr9= VarM (Int,[Decl("id1",N 9); Decl("id2",B False)]) in
2       let p=Prog("error-nine", StmD dclerr9)in
3       let s = toStringProg p in
4       let _ = printf "\n%s\n" s in
5       run p;;
6
7   Program error-nine{
8      varm int id1=9, id2=false}
9   Exception: TypeErrorS "E36: E32: E9: dclSem varm int id1=9, id2=false".
```

- Esempio di errore 8 ($t \neq$ Simple)

```
1       let dclerr8= VarM (Void,[Decl("id1",N 11); Decl("id2",N 13)]) in
2       let p=Prog("error-eight", StmD dclerr8)in
3       let s = toStringProg p in
4       let _ = printf "\n%s\n" s in
5       run p;;
6
7   Program error-eight{
8      varm void id1=11, id2=13}
9   Exception: TypeErrorS "E36: E32: E8: dclSem varm void id1=11, id2=13".
```