Costrutto di Swapping

Melania Pelosi

Università di Pisa

15 luglio 2020

Costrutto di Swapping

$$\texttt{dexp}_1 <-> \texttt{dexp}_2$$

Date due espressioni denotabili $\texttt{dexp}_1$ e $\texttt{dexp}_2$ calcolanti valori modificabili dello stesso tipo $t \in$ Simple, il costrutto esprime un comando che scambia tra loro i valori associati ai modificabili.

Ad esempio:

```
int x = 4;
int y = 8;
x <-> y;
```

Al termine dell'esecuzione si ottiene x $= 8$ e y $= 4$.

Il costrutto in Small20

- Sintassi Astratta: AST Ocaml

  `Cmd ::= ... | [Swap] Exp Exp`

- Sintassi Concreta: una CGF per Small20

  `NonConditionalCmd ::= ... | DExp <-> DExp`

  dove `DExp ::= ide | ide [ExpA2]` | ε

- Macchina Astratta: non è stata apportata nessuna modifica alla macchina astratta AM20.

Il costrutto in Small20

- Sintassi Astratta: AST Ocaml

  `Cmd ::= ... | [Swap] Exp Exp`

- Sintassi Concreta: una CGF per Small20

  `NonConditionalCmd ::= ... | DExp <-> DExp`

  dove `DExp ::= ide | ide [ExpA2]` | ε

- Macchina Astratta: non è stata apportata nessuna modifica alla macchina astratta AM20.

```
cmd =
    ...
    | Swap of exp * exp
    ...

toStringCmd tab = (function
    ...
    | Swap(exp1,exp2) -> ((indent tab) ^ "(" ^ (toStringExp exp1) ^ " <-> " ^ (toStringExp exp2) ^ ")")
    ...
```

Strutture esprimibili

- Il costrutto $\texttt{dexp}_1 <-> \texttt{dexp}_2$ permette di scambiare gli r-value di due espressioni denotabili dello stesso tipo.
- Può essere utilizzato in particolare negli algoritmi di ordinamento di array di interi.
- Senza il comando di Swapping, per scambiare il valore di due espressioni denotabili è necessario introdurre una variabile temporanea e allocare memoria per essa. Il costrutto, invece, procede allo scambio senza allocare ulteriore memoria nella macchina astratta AM20.

Regole di inferenza

- Sistema $\mathtt{Sem}_{\mathtt{CMD}}$: aggiunta regole per $\mathtt{Cmd} ::= \ldots \mid [\mathtt{Swap}]\ \mathtt{Exp}\ \mathtt{Exp}$

$$\text{C10:}\ \frac{\begin{array}{c}\langle \mathtt{e}_1,(\rho,\mu)\rangle \rightarrow_{\mathtt{DEN}} \lfloor [\mathtt{mut}]\ \mathtt{t}_1,\mathtt{loc}_1,(\rho,\mu)\rfloor \\ \langle \mathtt{e}_2,(\rho,\mu)\rangle \rightarrow_{\mathtt{DEN}} \lfloor [\mathtt{mut}]\ \mathtt{t}_2,\mathtt{loc}_2,(\rho,\mu)\rfloor \\ \mathtt{t}_1 = \mathtt{t}_2 \quad \mathtt{t}_1 \in \mathtt{Simple} \\ \mu(\mathtt{loc}_1) = \mathtt{v}_1 \quad \mu(\mathtt{loc}_2) = \mathtt{v}_2 \\ \mu[\mathtt{loc}_1 \leftarrow \mathtt{v}_2] = \mu_1 \quad \mu_1[\mathtt{loc}_2 \leftarrow \mathtt{v}_1] = \mu_F\end{array}}{\langle [\mathtt{swap}]\ \mathtt{e}_1\ \mathtt{e}_2,(\rho,\mu)\rangle \rightarrow ([\mathtt{void}],\ (\rho,\mu_F))}$$

- Sistema dei tipi Y

$$\mathtt{Y}_{\mathtt{swap}}:\frac{\begin{array}{c}\langle \mathtt{e}_1,\mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{DY}} ([\mathtt{mut}]\ \mathtt{t}_1,\mathtt{Y}_\rho) \\ \langle \mathtt{e}_2,\mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{DY}} ([\mathtt{mut}]\ \mathtt{t}_2,\mathtt{Y}_\rho) \\ \mathtt{t}_1 = \mathtt{t}_2 \quad \mathtt{t}_1 \in \mathtt{Simple}\end{array}}{\langle [\mathtt{swap}]\ \mathtt{e}_1\ \mathtt{e}_2,\mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} ([\mathtt{void}],\mathtt{Y}_\rho)}$$

Regole di inferenza

- Gestione degli errori di tipo

$$\text{E37:}\frac{\langle \mathtt{e}_1,\mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{DY}} (\mathtt{[mut]}\ \mathtt{t}_1,\mathtt{Y}_\rho) \quad \mathtt{t}_1 \notin \mathtt{Simple}}{\langle \mathtt{[swap]}\ \mathtt{e}_1\ \mathtt{e}_2,\mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{[terr]},\mathtt{Y}_\rho)}$$

Errore nel caso in cui la prima espressione denotabile non sia di tipo Simple.

$$\text{E38:}\frac{\langle \mathtt{e}_2,\mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{DY}} (\mathtt{[mut]}\ \mathtt{t}_2,\mathtt{Y}_\rho) \quad \mathtt{t}_2 \notin \mathtt{Simple}}{\langle \mathtt{[swap]}\ \mathtt{e}_1\ \mathtt{e}_2,\mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{[terr]},\mathtt{Y}_\rho)}$$

Errore nel caso in cui la seconda espressione denotabile non sia di tipo Simple.

$$\text{E39:}\frac{\langle \mathtt{e}_1,\mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{DY}} (\mathtt{[mut]}\ \mathtt{t}_1,\mathtt{Y}_\rho) \quad \langle \mathtt{e}_2,\mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{DY}} (\mathtt{[mut]}\ \mathtt{t}_2,\mathtt{Y}_\rho) \quad \mathtt{t}_1 \neq \mathtt{t}_2 \quad \mathtt{t}_1 \in \mathtt{Simple}}{\langle \mathtt{[swap]}\ \mathtt{e}_1\ \mathtt{e}_2,\mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{[terr]},\mathtt{Y}_\rho)}$$

Errore nel caso in cui le due espressioni non siano dello stesso tipo.

Implementazione: interprete di Small20

```
cmdSem cmd (rho,(Store(d,g)as mu)) = (
    match cmd with
        ...
      | Swap (e1,e2) ->(
          match dexpSem e1 (rho,mu) with
              |(Mut t1,loc1,(rho,mu))
                 when (isSimple t1)
                  ->(match dexpSem e2 (rho,mu) with
                      |(Mut t2,loc2,(rho,mu))
                         when (isSimple t2) && (ysame t1 t2)
                          ->(let v1= mTOe(getStore mu loc1) in
                              let v2= mTOe(getStore mu loc2) in
                              let mu1=upd mu loc1 (eTOm v2) in
                              let muF=upd mu1 loc2 (eTOm v1) in
                              (Void,(rho,muF)))
                      |(Mut t2, loc2, (rho,mu))
                         when (isSimple t2)
                          -> (let msg = "E39: cmdSem - " ^ (toStringExp e1) ^ ", " ^ (toStringExp e2) in
                              raise(TypeErrorC msg))
                      |_ ->(let msg = "E39: E38: cmdSem - " ^ (toStringExp e2) in
                              raise(TypeErrorC msg)))
              |_ ->(let msg = "E37: cmdSem - " ^ (toStringExp e1) in
                  raise(TypeErrorC msg))))
      and
```

Verifica del codice ed esempi

```
let st1=StmD(Var(Int, "x", N 6));;
let st2=StmD(Var(Int, "y", N 4));;
let st3=StmD(Var(Int , "sum", Plus(Val "x", Val "y")));;
let st4=StmD(Var(Int, "diff", Minus(Val "x", Val "y")));;
let st5=StmC(Swap(Val "sum", Val "diff"));;
let p=Prog("Swap", Seq[st1;st2;st3;st4;st5]);;
printProg p;;
run p;;
```

```
Program Swap{
   var int x = 6;
   var int y = 4;
   var int sum = (x + y);
   var int diff = (x - y);
   (sum <-> diff);
}

- : unit = ()

============== Traccia del Programma Swap ==========
================================================
Stack: [diff/(int,L3); sum/(int,L2); y/(int,L1); x/(int,L0)]
Store: [L0<-6,L1<-4,L2<-2,L3<-10]
================================================
==================== Traccia: Fine ================
- : unit = ()
```

Verifica del codice ed esempi

Errori di tipo

- E39: Le due espressioni denotabili non sono dello stesso tipo.

```
let st1=StmD(Var(Int, "x", N 5));;
let st2=StmD(Var(Bool, "y", B True));;
let st3=StmC(Swap(Val "x", Val "y"));;
let p=Prog("Swap", Seq[st1;st2;st3]);;
printProg p;;
run p;;
```

```
Program Swap{
   var int x = 5;
   var bool y = true;
   (x <-> y);
}

- : unit = ()
Exception: TypeErrorC "E39: cmdSem - x, y".
```

Un esempio: Selection Sort

Il seguente programma, dato un array di interi A, utilizza il comando di swapping per ordinare l'array tramite l'algorito di Selection Sort.

```
let st1= StmD(Array(Int, "A", 6));;
let st2= Seq[StmC(Cmd(Upd(GetArrow("A", N 0),N 8)));StmC(Cmd(Upd(GetArrow("A", N 1),N 3)));
     StmC(Cmd(Upd(GetArrow("A", N 2),N 6)));StmC(Cmd(Upd(GetArrow("A", N 3),N 2)));
     StmC(Cmd(Upd(GetArrow("A", N 4),N 9)));StmC(Cmd(Upd(GetArrow("A", N 5),N 7)))];;
printStm 0 st2;;
let ms1= StmD(Var(Int, "i", N 0));;
let eg1= LT(Val "i", N 6);;
let mi1= StmC(Cmd(Upd(Val "i",Plus(Val "i", N 1))));;
let st3= StmD(VarN(Int, "j"));;
let ms2= StmC(Cmd(Upd(Val "j", Plus(Val "i", N 1))));;
let eg2= LT(Val "j", N 6);;
let mi2= StmC(Cmd(Upd(Val "j",Plus(Val "j", N 1))));;
let e1= LT(GetArrow("A", Val "j"),GetArrow("A", Val "i"));;
let c3= Swap(GetArrow("A", Val "j"),GetArrow("A", Val "i"));;
let c2= IfT(e1,c3);;
let c1= For(ms2,eg2,mi2,c2);;
let st4= StmC(For(ms1,eg1,mi1,c1));;
let p= Prog("SelectionSort", Seq[st1;st2;st3;st4]);;
printProg p;;
run p;;
```

Un esempio: Selection Sort

```
Program SelectionSort{
   int array  A[6];
   A[0] = 8;
   A[1] = 3;
   A[2] = 6;
   A[3] = 2;
   A[4] = 9;
   A[5] = 7;
;
   var int j;
   for (var int i = 0;(i < 6);i = (i + 1))
            for (j = (i + 1);(j < 6);j = (j + 1))
                     if (A[j] < A[i]) (A[j] <-> A[i]);
}

- : unit = ()

============== Traccia del Programma SelectionSort ==========
================================================
Stack: [i/(int,L7); j/(int,L6); A/(int[6],L0)]
Store: [L0<-2,L1<-3,L2<-6,L3<-7,L4<-8,L5<-9,L6<-6,L7<-6]
================================================
==================== Traccia: Fine ================
- : unit = ()
```

Grazie per l'attenzione