Linguaggi di Programmazione con Laboratorio

Seminario di fine corso:
Funzioni con Invocazioni Tail Recursive in Small21

Silvia Ballotta

Università di Pisa

28 Ottobre 2021

Introduzione

Definizione

Sia f una funzione che nel suo corpo contiene un'invocazione di una funzione g. La chiamata di g si dice "chiamata di coda" se la funzione f restituisce il valore restituito da g senza eseguire ulteriori computazioni.

Introduzione

Definizione

Sia f una funzione che nel suo corpo contiene un'invocazione di una funzione g. La chiamata di g si dice "chiamata di coda" se la funzione f restituisce il valore restituito da g senza eseguire ulteriori computazioni.

- Nel caso in cui g coincida con f possiamo evitare l'allocazione di nuovi record di attivazione per le chiamate successive della funzione stessa, in quanto possiamo sempre riutilizzare l'AR in cima allo stack.

Introduzione

Definizione

Sia f una funzione che nel suo corpo contiene un'invocazione di una funzione g. La chiamata di g si dice "chiamata di coda" se la funzione f restituisce il valore restituito da g senza eseguire ulteriori computazioni.

- Nel caso in cui g coincida con f possiamo evitare l'allocazione di nuovi record di attivazione per le chiamate successive della funzione stessa, in quanto possiamo sempre riutilizzare l'AR in cima allo stack.
- L'obiettivo del progetto è quello di introdurre in Small21 l'invocazione per ricorsione di coda implementando il meccanismo di riuso dell'AR.

Esempi

Calcolo del sesto numero di Fibonacci: $F_6 = 8$

```
Program Fibonacci{
     int y;
     int fib(value int n){ -> diverge se n<0
        if (n == 0) return 1;
        if (n == 1) return 1;
        return (fib((n - 1)) + fib((n - 2)));

            }
     y = fib(5);     }
```

Allocazione di 15 AR

```
Stack:
>{Fibonacci,0,[fib/([int::int],$fib,
[int::int],:fpar:,:cmd:,1$);
 y/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-8,L1<-5,L2<-4,L3<-3,L4<-2,L5<-1,
L6<-0,L7<-1,L8<-2,L9<-1,L10<-0,L11<-3,
L12<-2,L13<-1,L14<-0,L15<-1]
```

Esempi

Calcolo del sesto numero di Fibonacci: $F_6 = 8$

```
Program Fibonacci{
     int y;
     int fib(value int n){ -> diverge se n<0
        if (n == 0) return 1;
        if (n == 1) return 1;
        return (fib((n - 1)) + fib((n - 2)));

            }
     y = fib(5);      }
```

Allocazione di 15 AR

```
Stack:
>{Fibonacci,0,[fib/([int::int],$fib,
[int::int],:fpar:,:cmd:,1$);
 y/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-8,L1<-5,L2<-4,L3<-3,L4<-2,L5<-1,
L6<-0,L7<-1,L8<-2,L9<-1,L10<-0,L11<-3,
L12<-2,L13<-1,L14<-0,L15<-1]
```

```
Program FibonacciTR{
     int n = 5;
     int res1 = 1;
     int y;
     int fibrc(value int res2){
        int x = res1;
        if (n == 0) return res2;
        if (n == 1) return res2;
        n = (n - 1);
        res1 = res2;
        return short fibrc((x + res2));

            }
     y = fibrc(1);      }
```

Vorremmo poter allocare un unico AR!

Esempi

Calcolo del fattoriale di tre: $3! = 6$

```
Program Fattoriale{
     int x = 3;
     int y;
     int fact(value int n){
        if (n < 0) return 0;
        if (n == 0) return 1;
        if (n > 0) return (n*fact((n - 1)));

            }
     y = fact(x);      }
```

Allocazione di 4 AR

```
Stack:
>{Fattoriale,0,[fact/([int::int],$fact,
[int::int],:fpar:,:cmd:,1$);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-3,L1<-6,L2<-3,L3<-2,L4<-1,L5<-0]
```

Esempi

Calcolo del fattoriale di tre: $3! = 6$

```
Program Fattoriale{
     int x = 3;
     int y;
     int fact(value int n){
        if (n < 0) return 0;
        if (n == 0) return 1;
        if (n > 0) return (n*fact((n - 1)));

            }
     y = fact(x);      }
```

Allocazione di 4 AR

```
Stack:
>{Fattoriale,0,[fact/([int::int],$fact,
[int::int],:fpar:,:cmd:,1$);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-3,L1<-6,L2<-3,L3<-2,L4<-1,L5<-0]
```

```
Program FattorialeRC{
     int x = 3;
     int y;
     int factT(value int acc){
        if (x < 0) return 0;
        if (x == 0) return acc;
        if (x > 0) x = (x - 1);
        return short factT(((x + 1) * acc));

            }
     y = factT(1);      }
```

Vorremmo poter allocare un unico AR!

Vincoli e limitazioni

- In Small21 può essere trasmesso un unico parametro di tipo Simple alle funzioni e in una prima analisi questa potrebbe essere una grande limitazione per funzioni con invocazioni tail recursive.

Vincoli e limitazioni

- In Small21 può essere trasmesso un unico parametro di tipo Simple alle funzioni e in una prima analisi questa potrebbe essere una grande limitazione per funzioni con invocazioni tail recursive.
- È possibile far fronte a questo vincolo di Small21 sfruttando variabili non locali visibili nell'AR della funzione secondo lo scope statico.

Vincoli e limitazioni

- In Small21 può essere trasmesso un unico parametro di tipo Simple alle funzioni e in una prima analisi questa potrebbe essere una grande limitazione per funzioni con invocazioni tail recursive.
- È possibile far fronte a questo vincolo di Small21 sfruttando variabili non locali visibili nell'AR della funzione secondo lo scope statico.

```
Program FibonacciTR{
     int n = 5;
     int res1 = 1;
     int y;
     int fibrc(value int res2){
        int x = res1;   <-- res1 non locale
        if (n == 0) return res2;    <-- n non locale
        if (n == 1) return res2;
        n = (n - 1);
        res1 = res2;
        return short fibrc((x + res2));

            }
     y = fibrc(1);     }
```

Sintassi Astratta e Concreta

- Sintassi Astratta di Small21

  Exp ::= ... | [short] Ide Exp

Sintassi Astratta e Concreta

- Sintassi Astratta di Small21

  Exp ::= ... | [short] Ide Exp

- Sintassi Concreta: CFG per Small21

  Exp $\rightarrow$ ExpB == ExpB | ExpB
  ExpB $\rightarrow$ ExpB or ExpR | truth | ExpR
  ExpR $\rightarrow$ ExpA $>$ ExpA | ExpA $<$ ExpA| ExpA
  ExpA $\rightarrow$ ExpA + ExpT | ExpA $-$ ExpT | ExpT
  ExpT $\rightarrow$ num | (Exp) | ide (AP) | DExp | **short** ide (Exp)
  DExp $\rightarrow$ ide | ide [ExpA]

Sistema dei tipi di Small21

Exp ::= [Short] Ide Exp

Sistema Y:

$$\mathtt{Y}_\rho(\mathtt{ide}) = \mathtt{[abs]t::t1}$$

$$\langle \mathtt{e},\ \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{te}, \mathtt{Y}_\rho)$$

$$[\mathrm{Y40}]\ \frac{\mathtt{t} \in \mathtt{Simple} \qquad \mathtt{t1\ =\ te}}{\langle \mathtt{[short]\ ide\ e}, \mathtt{Y}_\rho\rangle \rightarrow_{\mathtt{Y}} (\mathtt{t}, \mathtt{Y}_\rho)}$$

Sistema dei tipi di Small21

Exp ::= [Short] Ide Exp

Sistema Y:

$$\mathtt{Y}_\rho(\mathtt{ide}) = \mathtt{[abs]t::t1}$$

$$\langle \mathtt{e},\ \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{te}, \mathtt{Y}_\rho)$$

$$[\mathrm{Y40}]\ \frac{\mathtt{t} \in \mathtt{Simple} \qquad \mathtt{t1 = te}}{\langle \mathtt{[short]\ ide\ e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}, \mathtt{Y}_\rho)}$$

Gestione Errori di Tipo:

$$[\mathrm{E68}]\ \frac{\mathtt{Y}_\rho(\mathtt{ide}) \neq \mathtt{[abs]t::t1}}{\langle \mathtt{[short]\ ide\ e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{[terr]}, \mathtt{Y}_\rho)}$$

Sistema dei tipi di Small21

Exp ::= [Short] Ide Exp

Sistema Y:

$$\mathtt{Y}_\rho(\mathtt{ide}) = \mathtt{[abs]t::t1}$$

$$\langle \mathtt{e},\ \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{te}, \mathtt{Y}_\rho)$$

$$[\mathtt{Y40}]\ \frac{\mathtt{t} \in \mathtt{Simple} \qquad \mathtt{t1\ =\ te}}{\langle \mathtt{[short]\ ide\ e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{t}, \mathtt{Y}_\rho)}$$

Gestione Errori di Tipo:

$$[\mathtt{E68}]\ \frac{\mathtt{Y}_\rho(\mathtt{ide}) \neq \mathtt{[abs]t::t1}}{\langle \mathtt{[short]\ ide\ e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{[terr]}, \mathtt{Y}_\rho)}$$

$$[\mathtt{E69}]\ \frac{\mathtt{Y}_\rho(\mathtt{ide}) = \bot}{\langle \mathtt{[short]\ ide\ e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{[terr]}, \mathtt{Y}_\rho)}$$

$$[\text{E70}]\ \frac{\mathtt{Y}_\rho(\mathtt{ide}) = \mathtt{[abs]t::t1} \qquad \langle \mathtt{e},\ \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{[terr]}, \mathtt{Y}_\rho)}{\langle \mathtt{[short]\ ide\ e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{[terr]}, \mathtt{Y}_\rho)}$$

$$[\text{E70}]\ \frac{\mathtt{Y}_\rho(\mathtt{ide}) = \mathtt{[abs]t::t1} \qquad \langle \mathtt{e},\ \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{[terr]}, \mathtt{Y}_\rho)}{\langle \mathtt{[short]\ ide\ e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{[terr]}, \mathtt{Y}_\rho)}$$

$$[\text{E71}]\ \frac{\mathtt{Y}_\rho(\mathtt{ide}) = \mathtt{[abs]t::t1} \qquad \langle \mathtt{e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{te}, \mathtt{Y}_\rho) \qquad \mathtt{t1} \neq \mathtt{te}}{\langle \mathtt{[short]\ ide\ e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{[terr]}, \mathtt{Y}_\rho)}$$

$$[\text{E70}]\ \frac{\mathtt{Y}_\rho(\mathtt{ide}) = \mathtt{[abs]t::t1} \qquad \langle \mathtt{e},\ \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{[terr]}, \mathtt{Y}_\rho)}{\langle \mathtt{[short]\ ide\ e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{[terr]}, \mathtt{Y}_\rho)}$$

$$[\text{E71}]\ \frac{\mathtt{Y}_\rho(\mathtt{ide}) = \mathtt{[abs]t::t1} \qquad \langle \mathtt{e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{te}, \mathtt{Y}_\rho) \qquad \mathtt{t1} \neq \mathtt{te}}{\langle \mathtt{[short]\ ide\ e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{[terr]}, \mathtt{Y}_\rho)}$$

$$[\text{E72}]\ \frac{\mathtt{Y}_\rho(\mathtt{ide}) = \mathtt{[abs]t::t1} \qquad \mathtt{t} \notin \mathtt{Simple}}{\langle \mathtt{[short]\ ide\ e}, \mathtt{Y}_\rho\rangle \to_{\mathtt{Y}} (\mathtt{[terr]}, \mathtt{Y}_\rho)}$$

Introduciamo un sistema che permette al costrutto Short di gestire le dichiarazioni presenti nella funzione invocata.

$\rightarrow_{uD}$: **sistema per l'iteratore di Update Declaration**

$$[\text{UD1}]\ \frac{\begin{array}{c}\mathtt{d} = \mathtt{[var]\ t\ id\ e} \\ \langle \mathtt{e}, \sigma \rangle \rightarrow \lfloor \mathtt{te}, \mathtt{ve}, (\Delta_1, \mu_1) \rfloor \\ \Delta_1(\mathtt{id}) = (\mathtt{ti}, \mathtt{loc_{ti}}) \qquad \mathtt{ti} = \mathtt{[mut]\ ts} \\ \mathtt{ts} \in \mathtt{Simple} \qquad \mathtt{te} = \mathtt{ts} \\ \mu_1[\mathtt{loc_{ti}} \leftarrow \mathtt{ve}] = \mu_{\mathtt{F}}\end{array}}{\langle \mathtt{d}, \sigma \rangle \rightarrow_{uD} (\Delta_1, \mu_{\mathtt{F}})}$$

$$\mathtt{d} = \mathtt{d1} \;\; [\mathtt{seqD}] \;\; \mathtt{d2}$$

$$[\mathrm{UD2}]\frac{\langle \mathtt{d1}, \sigma \rangle \rightarrow_{uD} \sigma_1 \quad \langle \mathtt{d2}, \sigma_1 \rangle \rightarrow_{uD} \sigma_{\mathtt{F}}}{\langle \mathtt{d}, \sigma \rangle \rightarrow_{uD} \sigma_{\mathtt{F}}}$$

$$[\text{UD2}]\frac{\begin{array}{c}\texttt{d} = \texttt{d1 [seqD] d2} \\ \langle \texttt{d1}, \sigma \rangle \rightarrow_{uD} \sigma_1 \\ \langle \texttt{d2}, \sigma_1 \rangle \rightarrow_{uD} \sigma_{\texttt{F}}\end{array}}{\langle \texttt{d}, \sigma \rangle \rightarrow_{uD} \sigma_{\texttt{F}}}$$

$$[\text{UD3}]\frac{\begin{array}{c}\texttt{d} \neq \texttt{[var] t I e} \\ \texttt{d} \neq \texttt{d1 [seqD] d2}\end{array}}{\langle \texttt{d}, \sigma \rangle \rightarrow_{uD} \sigma}$$

Regole di Inferenza $\mathbf{SEM_{EXP}}$

Exp ::= [Short] Ide Exp

$$[\text{S20}]\ \frac{\begin{array}{c}\sigma = (\mathtt{ar} + \Delta, \mu) \qquad \mathtt{ar} = \{\mathtt{id},\mathtt{ls},\mathtt{fr},\mathtt{cnt},\mathtt{v}\} \qquad \mathtt{ide} = \mathtt{id} \\ \Delta(\mathtt{id}) = (\mathtt{t_r},\mathtt{c_r}) \qquad \mathtt{c_r} = *\mathtt{id},\mathtt{t_r},\mathtt{fps},\mathtt{dcl},\mathtt{sts},\mathtt{k}* \\ \mathtt{t_r} = [\mathtt{abs}]\mathtt{t}[:]\mathtt{t1} \qquad \mathtt{t} \in \mathtt{Simple} \\ \mathtt{fps} = [\mathtt{fp}][\mathtt{value}]\mathtt{t1}\ \mathtt{i} \qquad \langle \mathtt{e}, \sigma\rangle \to \lfloor \mathtt{te}, \mathtt{ve}, (\Delta_1, \mu_1)\rfloor \qquad \mathtt{t1} = \mathtt{te} \\ \Delta_1(\mathtt{i}) = (\mathtt{ti}, \mathtt{loc_{ti}}) \qquad \mu_1[\mathtt{loc_{ti}} \leftarrow \mathtt{ve}] = \mu_2 \\ \langle \mathtt{dcl}, (\Delta_1, \mu_2)\rangle \to_{uD} (\mathtt{ar_3} + \Delta_3, \mu_3) \qquad \mathtt{ar_3} = \{\mathtt{id},\mathtt{ls},\mathtt{fr3},\mathtt{cnt3},\mathtt{v3}\} \\ \{\mathtt{id},\mathtt{ls},\mathtt{fr3},\mathtt{sts},\mathtt{v3}\} = \mathtt{ar_4} \qquad (\mathtt{ar_4} + \Delta_3, \mu_3) \to_{R*} (\mathtt{ar_F} + \Delta_F, \mu_F) \\ \mathtt{ar_F} = \{\mathtt{id},\mathtt{ls},\mathtt{fr5},\mathtt{cnt5},\mathtt{v_F}\} \end{array}}{\langle [\mathtt{short}]\ \mathtt{ide}\ \mathtt{e}, \sigma\rangle \to \lfloor \mathtt{t}, \mathtt{v_F}, (\mathtt{ar_F} + \Delta_F, \mu_F)\rfloor}$$

$$\sigma = (\mathtt{ar} + \Delta, \mu) \qquad \mathtt{ar} = \{\mathtt{id},\mathtt{ls},\mathtt{fr},\mathtt{cnt},\mathtt{v}\} \qquad \mathtt{ide} = \mathtt{id}$$

$$\Delta(\mathtt{id}) = (\mathtt{t_r},\mathtt{c_r}) \qquad \mathtt{c_r} = *\mathtt{id},\mathtt{t_r},\mathtt{fps},\mathtt{dcl},\mathtt{sts},\mathtt{k}*$$

$$\mathtt{t_r} = [\mathtt{abs}]\mathtt{t}[:]\mathtt{t1} \qquad \mathtt{t} \in \mathtt{Simple}$$

$$\mathtt{fps} = [\mathtt{fp}][\mathtt{ref}]\mathtt{t1}\ \mathtt{i} \qquad \langle \mathtt{e}, \sigma\rangle \rightarrow_{\mathtt{DEN}} \lfloor [\mathtt{mut}]\mathtt{te}, \mathtt{loce}, (\Delta_1, \mu_1)\rfloor$$

$$\mathtt{t1} = \mathtt{te} \qquad \Delta_1(\mathtt{i}) = (\mathtt{ti}, \mathtt{loc_{ti}}) \qquad \mathtt{loce}{=}\mathtt{loc_{ti}}$$

$$\langle \mathtt{dcl}, (\Delta_1, \mu_1)\rangle \rightarrow_{uD} (\mathtt{ar}_3 + \Delta_3, \mu_3) \qquad \mathtt{ar}_3 = \{\mathtt{id},\mathtt{ls},\mathtt{fr3},\mathtt{cnt3},\mathtt{v3}\}$$

$$\{\mathtt{id},\mathtt{ls},\mathtt{fr3},\mathtt{sts},\mathtt{v3}\} = \mathtt{ar}_4 \qquad (\mathtt{ar}_4 + \Delta_3, \mu_3) \rightarrow_{R*} (\mathtt{ar_F} + \Delta_\mathtt{F}, \mu_\mathtt{F})$$

$$[\mathrm{S21}]\ \frac{\mathtt{ar_F} = \{\mathtt{id},\mathtt{ls},\mathtt{fr5},\mathtt{cnt5},\mathtt{v_F}\}}{\langle [\mathtt{short}]\ \mathtt{ide}\ \mathtt{e}, \sigma\rangle \rightarrow \lfloor \mathtt{t},\mathtt{v_F},(\mathtt{ar_F} + \Delta_\mathtt{F}, \mu_\mathtt{F})\rfloor}$$

OSS: $\mathtt{loce}{=}\mathtt{loc_{ti}}$ poichè in Small21 un binding del frame di un AR non può essere modificato.

$$\sigma = (\mathtt{ar} + \Delta, \mu) \qquad \mathtt{ar} = \{\mathtt{id},\mathtt{ls},\mathtt{fr},\mathtt{cnt},\mathtt{v}\} \qquad \mathtt{ide} = \mathtt{id}$$

$$\Delta(\mathtt{id}) = (\mathtt{t_r},\mathtt{c_r}) \qquad \mathtt{c_r} = *\mathtt{id},\mathtt{t_r},\mathtt{fps},\mathtt{dcl},\mathtt{sts},\mathtt{k}*$$

$$\mathtt{t_r} = [\mathtt{abs}]\mathtt{t}[::]\mathtt{t1} \qquad \mathtt{t} \in \mathtt{Simple}$$

$$\mathtt{fps} = \mathtt{EFP} \qquad \mathtt{e} = [\mathtt{emptyE}]$$

$$\langle \mathtt{dcl}, \sigma\rangle \to_{uD} (\mathtt{ar}_3 + \Delta_3, \mu_3) \qquad \mathtt{ar}_3 = \{\mathtt{id},\mathtt{ls},\mathtt{fr3},\mathtt{cnt3},\mathtt{v3}\}$$

$$\{\mathtt{id},\mathtt{ls},\mathtt{fr3},\mathtt{sts},\mathtt{v3}\} = \mathtt{ar}_4 \qquad (\mathtt{ar}_4 + \Delta_3, \mu_3) \to_{R^*} (\mathtt{ar_F} + \Delta_\mathtt{F}, \mu_\mathtt{F})$$

$$[\mathsf{S22}]\ \frac{\mathtt{ar_F} = \{\mathtt{id},\mathtt{ls},\mathtt{fr5},\mathtt{cnt5},\mathtt{v_F}\}}{\langle [\mathtt{short}]\ \mathtt{ide}\ \mathtt{e}, \sigma\rangle \to \lfloor \mathtt{t},\mathtt{v_F},(\mathtt{ar_F} + \Delta_\mathtt{F}, \mu_\mathtt{F})\rfloor}$$

Modifiche all'Interprete di Small21

```
exp =
        Val of ide
        | Arrow1 of ide * exp
        | N of num
        | Plus of exp * exp
        | Minus of exp * exp
        | Times of exp * exp
        | LT of exp * exp
        | GT of exp * exp
        | Eq of exp * exp
        | B of bool
        | Or of exp * exp
        | Apply of ide * apars
        | Short of ide * exp
        | EE
    and

toStringExp = (function
|....
| Short(ide, exp) -> "short " ^ toStringI ide ^ "(" ^ (toStringExp exp) ^ ")"
|....
```

Modifiche all'Interprete di Small21

```
let rec upDcl dcl (sk, mu)=
        match dcl with
        | Var(t, id, e)-> (let (te, ve, (sk1, mu1)) = expSem e (sk,mu) in
                          match getS sk1 id  with
                          | DVar(Mut ts, loct1) when (isSimple ts) && (ysame te ts)
                                  -> let muF = upd mu1 loct1 (eTOm ve) in
                          (sk1, muF)
                          | _ ->  raise(SystemErrorE("upDcl")))
        | SeqD(d1, d2)-> (let (sk1, mu1)= upDcl d1 (sk,mu) in
                          let (skF, muF)= upDcl d2 (sk1, mu1) in
                          (skF, muF))
        | _ -> (sk,mu)
        ;;
```

Modifiche all'Interprete di Small21

```
expSem exp (sk,(Store(d,g)as mu)) =
    match exp with
            Short(ide, e) ->
            (match getH sk with
                    | Name id when (ide = id)
                            (* include check for E69 *)
                            ->(match getS sk ide with
                                    (* include check for E72 *)
                                    | DAbs(Abs(t,aa),ClosT(_,_,fps,dcl,sts,k)) when (isSimple t)
                                    -> match fps with
                                            | FP(Value,t1,i)->(let (te,ve, (sk1, mu1))=expSem e (sk,mu) in (* include check for E70 *)
                                                                    if (ysame t1 te) then
                                                                    (match getS sk1 i with
                                                                    | DVar(Mut ti, locti)
                                                                            -> (let mu2 = upd mu1 locti (eTOm ve) in
                                                                            let (sk3, mu3)= upDcl dcl (sk1, mu2) in
                                                                            let sk4 = resetC sk3 [UnL sts] in
                                                                            let (_, (skf, muf)) = nextCmd(sk4, mu3) in
                                                                            (t, getR skf, (skf,muf)))
                                                                    | _ -> raise(SystemErrorE("expSem: uncorret parameter transmission")))
                                                                    else raise(TypeErrorE("E71: expSem", exp)))

                                            | FP(Ref, t1, i) -> (match dexpSem e (sk,mu) with (* include check for E70 *)
                                                                    | (Mut te, loce, (sk1,mu1)) when (ysame t1 te)
                                                                    -> match getS sk1 i with
                                                                            | DVar(Mut ti, locti) when (loce=locti)
                                                                            -> (    let (sk3, mu3)= upDcl dcl (sk1, mu1) in
                                                                                    let sk4 = resetC sk3 [UnL sts] in
                                                                                    let (_, (skf, muf)) = nextCmd(sk4, mu3) in
                                                                                    (t, getR skf, (skf,muf)))
                                                                            | _ -> raise(SystemErrorE("expSem: uncorret parameter transmission"))
                                                                    | _ -> raise(SystemErrorE("expSem: uncorret parameter transmission \ E71")))
                                    | EFP when (e = EE) -> (
                                                    let (sk3, mu3)= upDcl dcl (sk, mu) in
                                                    let sk4 = resetC sk3 [UnL sts] in
                                                    let (_, (skf, muf)) = nextCmd(sk4, mu3) in
                                                    (t, getR skf, (skf,muf)))
                                    | _ -> raise(TypeErrorE("E71: expSem", exp))
                            | _ -> raise(TypeErrorE("E68: expSem", exp)))
    | _ -> raise(TypeErrorE("Wrong use of Short: expSem", exp)))
    ;;
```

Verifica del comportamento della struttura

Calcolo del sesto numero di Fibonacci: $F_6 = 8$

```
(*Chiamata ricorsiva con Short*)
let d1=Var(Int, "n", N 5) in
let d2=Var(Int, "res1", N 1) in
let d3=Var(Int, "y", EE) in
let d4=Var(Int, "x", Val("res1")) in
let s1=IfT(Eq(Val("n"), N 0), Return(Val("res2"))) in
let s2=IfT(Eq(Val("n"), N 1), Return(Val("res2"))) in
let s3=Upd(Val("n"), Minus(Val("n"), N 1)) in
let s4=Upd(Val("res1"), Val("res2")) in
let s5=Return(Short("fibrc", Plus(Val("x"), Val("res2")))) in
let s6=Upd(Val("y"), Apply("fibrc", AP(N 1))) in
let d5=Pcd(Int, "fibrc", FP(Value, Int, "res2") ,
        BlockP(d4, SeqS(SeqS(SeqS(SeqS(s1,s2), s3), s4), s5))) in
let p=Prog("FibonacciTR",
        Block(SeqD(SeqD(SeqD(d1, d2), d3), d5), UnL s6)) in
progSem p;;
```

Verifica del comportamento della struttura

Calcolo del sesto numero di Fibonacci: $F_6 = 8$

```
(*Chiamata ricorsiva con Short*)
let d1=Var(Int, "n", N 5) in
let d2=Var(Int, "res1", N 1) in
let d3=Var(Int, "y", EE) in
let d4=Var(Int, "x", Val("res1")) in
let s1=IfT(Eq(Val("n"), N 0), Return(Val("res2"))) in
let s2=IfT(Eq(Val("n"), N 1), Return(Val("res2"))) in
let s3=Upd(Val("n"), Minus(Val("n"), N 1)) in
let s4=Upd(Val("res1"), Val("res2")) in
let s5=Return(Short("fibrc", Plus(Val("x"), Val("res2")))) in
let s6=Upd(Val("y"), Apply("fibrc", AP(N 1))) in
let d5=Pcd(Int, "fibrc", FP(Value, Int, "res2") ,
        BlockP(d4, SeqS(SeqS(SeqS(SeqS(s1,s2), s3), s4), s5))) in
let p=Prog("FibonacciTR",
        Block(SeqD(SeqD(SeqD(d1, d2), d3), d5), UnL s6)) in
progSem p;;
```

```
Program FibonacciTR{
     int n = 5;
     int res1 = 1;
     int y;
     int fibrc(value int res2){
        int x = res1;
        if (n == 0) return res2;
        if (n == 1) return res2;
        n = (n - 1);
        res1 = res2;
        return short fibrc((x + res2));

            }
     y = fibrc(1);     }
```

Verifica del comportamento della struttura

Calcolo del sesto numero di Fibonacci: $F_6 = 8$

```
(*Chiamata ricorsiva con Short*)
let d1=Var(Int, "n", N 5) in
let d2=Var(Int, "res1", N 1) in
let d3=Var(Int, "y", EE) in
let d4=Var(Int, "x", Val("res1")) in
let s1=IfT(Eq(Val("n"), N 0), Return(Val("res2"))) in
let s2=IfT(Eq(Val("n"), N 1), Return(Val("res2"))) in
let s3=Upd(Val("n"), Minus(Val("n"), N 1)) in
let s4=Upd(Val("res1"), Val("res2")) in
let s5=Return(Short("fibrc", Plus(Val("x"), Val("res2")))) in
let s6=Upd(Val("y"), Apply("fibrc", AP(N 1))) in
let d5=Pcd(Int, "fibrc", FP(Value, Int, "res2") ,
        BlockP(d4, SeqS(SeqS(SeqS(SeqS(s1,s2), s3), s4), s5))) in
let p=Prog("FibonacciTR",
        Block(SeqD(SeqD(SeqD(d1, d2), d3), d5), UnL s6)) in
progSem p;;
```

```
Program FibonacciTR{
     int n = 5;
     int res1 = 1;
     int y;
     int fibrc(value int res2){
        int x = res1;
        if (n == 0) return res2;
        if (n == 1) return res2;
        n = (n - 1);
        res1 = res2;
        return short fibrc((x + res2));

            }
     y = fibrc(1);     }
```

```
Stack:
>{FibonacciTR,0,[fibrc/([int::int],
$fibrc, [int::int],:fpar:,:cmd:,1$);
 y/(Mint,L2);
 res1/(Mint,L1);
 n/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-5,L2<-8,L3<-8,L4<-5]
```

Verifica del comportamento della struttura

Calcolo del fattoriale di tre: $3! = 6$

```
(*Chiamata ricorsiva con Short*)
let d1=Var(Int, "x", N 3) in
let d2=Var(Int, "y", EE) in
let s1=IfT(LT(Val("x"), N 0), Return(N 0)) in
let s2=IfT(Eq(Val("x"), N 0), Return(Val("acc"))) in
let e=Times(Plus(Val("x"), N 1), Val("acc"))  in
let s3=IfT(GT(Val("x"), N 0), SeqS(Upd(Val("x"),
        Minus(Val("x"), N 1)), Return(Short("factT", e)))) in
let d3=Pcd(Int, "factT", FP(Value, Int, "acc"),
        BlockP(ED, SeqS(SeqS(s1,s2), s3))) in
let p=Prog("FattorialeRC", Block(SeqD(d1, SeqD(d2, d3)),
        UnL(Upd(Val("y"),Apply("factT", AP(N 1)))))) in
progSem p;;
```

Verifica del comportamento della struttura

Calcolo del fattoriale di tre: $3! = 6$

```
(*Chiamata ricorsiva con Short*)
let d1=Var(Int, "x", N 3) in
let d2=Var(Int, "y", EE) in
let s1=IfT(LT(Val("x"), N 0), Return(N 0)) in
let s2=IfT(Eq(Val("x"), N 0), Return(Val("acc"))) in
let e=Times(Plus(Val("x"), N 1), Val("acc"))  in
let s3=IfT(GT(Val("x"), N 0), SeqS(Upd(Val("x"),
        Minus(Val("x"), N 1)), Return(Short("factT", e)))) in
let d3=Pcd(Int, "factT", FP(Value, Int, "acc"),
        BlockP(ED, SeqS(SeqS(s1,s2), s3))) in
let p=Prog("FattorialeRC", Block(SeqD(d1, SeqD(d2, d3)),
        UnL(Upd(Val("y"),Apply("factT", AP(N 1)))))) in
progSem p;;
```

```
Program FattorialeRC{
     int x = 3;
     int y;
     int factT(value int acc){
        if (x < 0) return 0;
        if (x == 0) return acc;
        if (x > 0) x = (x - 1);
        return short factT(((x+1)*acc));

            }
     y = factT(1);     }
```

Verifica del comportamento della struttura

Calcolo del fattoriale di tre: $3! = 6$

```
(*Chiamata ricorsiva con Short*)
let d1=Var(Int, "x", N 3) in
let d2=Var(Int, "y", EE) in
let s1=IfT(LT(Val("x"), N 0), Return(N 0)) in
let s2=IfT(Eq(Val("x"), N 0), Return(Val("acc"))) in
let e=Times(Plus(Val("x"), N 1), Val("acc"))  in
let s3=IfT(GT(Val("x"), N 0), SeqS(Upd(Val("x"),
        Minus(Val("x"), N 1)), Return(Short("factT", e)))) in
let d3=Pcd(Int, "factT", FP(Value, Int, "acc"),
        BlockP(ED, SeqS(SeqS(s1,s2), s3))) in
let p=Prog("FattorialeRC", Block(SeqD(d1, SeqD(d2, d3)),
        UnL(Upd(Val("y"),Apply("factT", AP(N 1)))))) in
progSem p;;
```

```
Program FattorialeRC{
     int x = 3;
     int y;
     int factT(value int acc){
        if (x < 0) return 0;
        if (x == 0) return acc;
        if (x > 0) x = (x - 1);
        return short factT(((x+1)*acc));

            }
     y = factT(1);     }
```

```
Stack:
>{FattorialeRC,0,[factT/([int::int],
$factT,[int::int],:fpar:,:cmd:,1$);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-0,L1<-6,L2<-6]
```

Verifica del comportamento della struttura

Sfrutto la traccia per controllare l'evoluzione dello stack.

```
Stack:
>{fact,3,[n/(Mint,L4)],:cmdNext:,[N]}
{fact,2,[n/(Mint,L3)],:cmdNext:,[N]}
{fact,1,[n/(Mint,L2)],:cmdNext:,[N]}
{Fattoriale,0,[fact/([int::int],$fact,
[int::int],:fpar:,:cmd:,1$);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-3,L1<-Undef,L2<-3,L3<-2,L4<-1]


Stack:
>{fact,4,[n/(Mint,L5)],:cmdNext:,[N]}
{fact,3,[n/(Mint,L4)],:cmdNext:,[N]}
{fact,2,[n/(Mint,L3)],:cmdNext:,[N]}
{fact,1,[n/(Mint,L2)],:cmdNext:,[N]}
{Fattoriale,0,[fact/([int::int],$fact,
[int::int],:fpar:,:cmd:,1$);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-3,L1<-Undef,L2<-3,L3<-2,L4<-1,L5<-0]
```

Verifica del comportamento della struttura

Sfrutto la traccia per controllare l'evoluzione dello stack.

```
Stack:
>{fact,3,[n/(Mint,L4)],:cmdNext:,[N]}
{fact,2,[n/(Mint,L3)],:cmdNext:,[N]}
{fact,1,[n/(Mint,L2)],:cmdNext:,[N]}
{Fattoriale,0,[fact/([int::int],$fact,
[int::int],:fpar:,:cmd:,1$);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-3,L1<-Undef,L2<-3,L3<-2,L4<-1]


Stack:
>{fact,4,[n/(Mint,L5)],:cmdNext:,[N]}
{fact,3,[n/(Mint,L4)],:cmdNext:,[N]}
{fact,2,[n/(Mint,L3)],:cmdNext:,[N]}
{fact,1,[n/(Mint,L2)],:cmdNext:,[N]}
{Fattoriale,0,[fact/([int::int],$fact,
[int::int],:fpar:,:cmd:,1$);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-3,L1<-Undef,L2<-3,L3<-2,L4<-1,L5<-0]
```

```
Stack:
>{factT,1,[acc/(Mint,L2)],:cmdNext:,[N]}
{FattorialeRC,0,[factT/([int::int],$factT,
[int::int],:fpar:,:cmd:,1$);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-Undef,L2<-3]


Stack:
>{factT,1,[acc/(Mint,L2)],:cmdNext:,[N]}
{FattorialeRC,0,[factT/([int::int],$factT,
[int::int],:fpar:,:cmd:,1$);
 y/(Mint,L1);
 x/(Mint,L0)],:cmdNext:,[N]}
]
Store:
[L0<-1,L1<-Undef,L2<-6]
```

Considerazioni finali

- Gli esempi introdotti mostrano che il costrutto Short aumenta notevolmente l'espressività del Linguaggio Small21.

Considerazioni finali

- Gli esempi introdotti mostrano che il costrutto Short aumenta notevolmente l'espressività del Linguaggio Small21.
- Siamo in grado di scrivere delle funzioni con ricorsione in coda usando un solo record di attivazione, quindi uno spazio di memoria costante.

Riferimenti bibliografici

- Marco Bellia, 2021. *Materiale delle lezioni del corso di LPL: Laboratori 2,3,4,5,6,7*
- Marco Bellia, 2021. Materiale delle lezioni del corso di LPL: Lezione 6.
- Marco Bellia, 2021. Materiale delle lezioni del corso di LPL: Lezione 7.
- Marco Bellia, 2021. Materiale delle lezioni del corso di LPL: Small21-Definizione7.
- Programming Languages: Principles and Paradigms (2010), Gabrielli M., S. Martini, Springer-Verlag, London.

Grazie per l'attenzione!