Progetto di Linguaggi di programmazione: Traduttore da Lisp a linguaggio per SECD

May 24, 2006

1 Introduzione

Il progetto consiste nel realizzare un programma ML che traduca programmi Lisp in (equivalenti) programmi nel linguaggio per la macchina astratta SECD. In questo progetto vengono considerati esclusivamnte programmi Lisp NON RICORSIVI. In una successiva estensione considereremo anche i programmi con ricorsione.

Nel seguito chiameremo **COMP** (per COMPilatore) il programma da fare. Per semplificare al massimo il parsing che COMP deve eseguire, assumiamo che i programmi Lisp che costituiscono l'input di COMP vengano presentati in un formato particolarmente semplice. Questo formato viene chiamato **FS** (per Forma Semplice) ed è descritto nella sezione seguente. Anche i programmi SECD che COMP produce in output sono in una forma semplifcata (simile alla FS) che è illustrata nella Sezione 3. La traduzione che COMP deve effettuare viene descritta caso per caso nella Sezione 4 e, finalmente, nella Sezione 5 viene descritto precisamente cosa fare e come e quando consegnare il progetto. Questa sezione contiene anche uno schema del programma COMP e alcuni esempi di traduzione.

2 Lisp in Forma Semplice

In pratica, il Lisp in Forma Semplice (FS) è tale che la sua analisi sintattica diventa molto semplice perchè l'abbiamo già fatta noi *a mano*. La FS è descritta dal seguente tipo ML `Sexpr` nel senso che i programmi in FS sono valori di tipo `Sexpr`. Il tipo `s_espressione`e è il tipo delle costanti che possono venire usate nei programmi Lisp. Come si vede dalla definizione di `s_espressione` esso rappresenta gli interi, le stringhe, i booleani (`T,F`), il NIL e le s_espressioni composte con questi valori scalari.

```
datatype s_espressione = NUM of int |
                         STRINGA of string |
                         T | F | NIL |
                         DOT of s_espressione * s_espressione
```

```
datatype Sexpr = Var of string |
                 Quote of s_espressione |
                 Op of string * Sexpr list |
                 If of Sexpr * Sexpr * Sexpr |
                 Lambda of string list * Sexpr |
                 Call of Sexpr * Sexpr list   |
                 Let of Sexpr * string list * Sexpr list
```

Nel seguito spieghiamo il significato dei diversi casi del tipo `Sexpr` e contemporaneamente spieghiamo (in modo informale) il significato dei costrutti Lisp che vengono rappresentati da quel particolare tipo. In alcuni punti, per distinguere tra parti diverse di un tipo aggiungiamo degli indici a queste parti. Deve essere chiaro che gli indici hanno solo questa funzione disambiguante e nessun altro significato.

- `Var of string`: l'espressione serve a rappresentare le variabili dei programmi. La variabile `x` verrà rapresentata con `Var(''x'')`;
- Ogni costante dei programmi Lisp FS va scritta come elemento del tipo `s_espressione` e inserita nel programma come argomento dell'operatore `Quote`. Per esempio la costante `1` è scritta come `Quote(NUM(1))`, mentre la lista `[4 ''apple'']` è scritta come `Quote(DOT(NUM(1),DOT(STRINGA(''apple''),NIL)))`;
- `Op of string * Sexpr list`: il costruttore `Op` rappresenta i diversi operatori del Lisp. Il particolare operatore viene identificato dalla stringa `string`, mentre gli operandi sono contenuti nella successiva lista di tipo `Sexpr list`. Gli operatori considerati sono i seguenti:
  - Unari: CAR, CDR, ATOM;
  - Binari: ADD, SUB, MUL, DIV, REM, EQ, LEQ, CONS;

  Il significato degli operatori dovrebbe essere chiaro.
- `If of Sexpr_1 * Sexpr_2 * Sexpr_3`: l'espressione rappresenta il seguente condizionale: if Sexpr_1 then Sexpr_2 else Sexpr_3. Ovviamente queste espressioni rappresentano l'arci-noto costrutto condizionale.
- `Lambda of string list * Sexpr`: l'espressione è una λ-espressione dove `string list` è la lista delle variabili legate e `Sexpr` è il corpo della λ-espressione. Per esempio, il programma Lisp `(Lambda(x y) (ADD x y))` in FS diventa la sequente espressione: `(Lambda([''x'', ''y''], Op(''ADD'', [Var(''x''), Var(''y'')])))`.
- `Call of Sexpr_1 * Sexpr list_2` corrisponde all'invocazione di una funzione con una data lista di parametri attuali. `Sexpr_1` è la funzione invocata e la `Sexpr list_2` è la lista dei parametri attuali. Per esempio, il programma Lisp, `((LAMBDA(x y)(ADD x y)) 5 4)` diventa in FS la seguente espressione:

  `Call(Lambda([''x'', ''y''], Op(''ADD'', [Var(''x''), Var(''y'')])), [Quote(Num(5)), Quote(Num(4))])`.

- `Let of Sexpr_0 * string list_1 * Sexpr list_2`: l'espressione `Sexpr_0` è il corpo del `LET`, `string list_1` è la lista delle variabili locali e `Sexpr list_2` è la lista delle espressioni che inizializzano i loro valori. Per esempio il programmma Lisp `LET x=5 and y=4 in x+y` che valuta l'espressione `x+y` nell'ambiente in cui `x=5` e `y=4`, diventa in FS la seguente espressione: `Let(Op(''ADD'', [Var(''x''), Var(''y'')]), [''x'', ''y''], [Quote(Num(5)), Quote(Num(4))])`.

3 La macchina SECD e il suo linguaggio

La macchina SECD consiste di una quadrupla `<S,E,C,D>` come segue:

- La pila S va vista come i registri di una CPU, essa contiene i risultati parziali e finali del calcolo;
- L'ambiente E contiene liste di liste di valori da associare alle variabili; ogni valore nell'ambiente viene identificato da una coppia di interi `<n1,n2>` in cui il primo individua la sottolista di E da considerare ed il secondo identifica l'elemento di questa sottolista. Questa maniera di accedere ai valori in E viene usata per il comando `LD`, vedi nel seguito di questa stessa Sezione. La coppia `<n1,n2>` viene calcolata durante la compilazione del programma Lisp utilizzando una lista di liste di nomi (di variabili) che ha identica struttura di E e usando la quale la compilazione determinerà la posizione delle variabili che vengono usate nel programma che viene compilato. Questo calcolo è descritto nella Sezione 4 nel punto che concerne la traduzione di `Var(''x'')`, che corrisponde infatti all'uso della variabile "x" nel programma che si sta compilando.
- Il controllo C è il programma SECD che deve essere eseguito;
- Il dump D è un deposito in cui viene salvato lo stato della macchina SECD per esempio quando viene invocata una funzione oppure quando viene eseguito un comando condizionale. Si confronti con l'illustrazione dei comandi SECD che segue.

Descriviamo ora il linguaggio per la macchina astratta SECD. Nel seguito chiameremo questo linguaggio semplicemente Linguaggio SECD. Si deve tenere a mente che il significato dei diversi comandi di questo calcolo è di far evolvere la macchina SECD. I programmi prodotti dal software COMP sono in linguaggio SECD e quindi per realizzare COMP è indispensabile capire bene cosa fanno le operazioni di questo linguaggio. Come per i programmi Lisp, definiamo nel seguito un tipo ML che serve a rappresentare in modo semplice i programmi SECD. Presentiamo subito questo tipo e, spiegando i suoi diversi casi, spiegheremo anche cosa fanno le diverse operazioni del linguaggio SECD.

```
datatype secdexpr = ADD | SUB |  MUL | DIV | REM | EQ | LEQ | CAR |
                    CDR | CONS |  ATOM | JOIN | RTN | AP |
```

```
LD of int*int |
LDC of s_espressione |
SEL of secdexpr list * secdexpr list|
LDF of  secdexpr list
```

Segue la spiegazione dei diversi casi del tipo `secdexpr`:

- i costruttori ADD, SUB, MUL, DIV, REM, EQ, LEQ, CAR, CDR, CONS, ATOM, JOIN, RTN, AP rappresentazione le operazioni 0-arie della SECD. Tutte queste operazioni a parte le ultime 3 hanno un significato immediato e che corrisponde alle analoghe operazioni Lisp. Può forse sorprendere il fatto che non abbiano operandi (mentre in Lisp li hanno). Il motivo è che nella macchina SECD, gli operandi di questi operatori vengono messi sullo stack con opportune LD o LDC (vedi i punti successivi) prima di eseguire l'operazione che li usa. Il risultato dell'operazione viene poi inserito sullo stack della SECD al posto degli operandi. Per quanto riguarda gli ultimi 3 operatori, il `JOIN` viene spiegato più sotto assieme al costruttore `SEL`, mentre `RTN` e `AP` verranno descriti subito dopo la presente lista di punti.

- `LD of int1*int2`: è il comando di caricamento sulla pila S della SECD del valore che compare nella posizione `<n1,n2>` dell'ambiente E.Precisamente, visto che E è una lista di liste di valori, `LD(n1,n2)` carica sulla cima dello stack S l'n2-esimo valore dell'n1-esima lista di E.

- `LDC of s_espressione`: è il comando che carica sulla pila S una s_espressione senza valutarla. Per la definizione di `s_espressione` si veda la Sezione 2. Si noti che le costanti usate nei programmi Lisp in FS e quelle dei programmi SECD sono valori dello stesso tipo `s_espressioni`. In questo modo COMP non deve fare nulla per tradurre le costanti.

- `SEL of secdexpr list1 * secdexpr list2`: è il ben noto comando condizionale che, a seconda del valore di verità in cima alla pila S, esegue le istruzioni del suo primo parametro (caso then) oppure del secondo (ramo else). Più precisamente se la situazione della SECD è la seguente:

  ```
  (T S)  E  (SEL(C_T,C_F) C) D --> S  E  C_T  (C D)
  ```

  Quindi il programmma C che segue `SEL` viene ricordato sul dump D e viene eseguito C_T visto che sullo stack S c'è `T` che rappresenta `true` ci fosse `F` verrebbe eseguito C_F. Sia C_T che C_F devono avere come ultima istruzione la `JOIN` la cui funzione è la seguente:

  ```
  S  E  (JOIN) (C D) --> S  E  C  D
  ```

  cioè si restaura il controllo C salvato sul dump.

- `LDF of secdexpr list`: è il comando di caricamento nella pila S di una funzione cioè della lista di espressioni `secdexpr` che costituisce il corpo della funzione. Per capire la prossima spiegazione conviene tenere a mente che il valore di una funzione è una coppia chiamata **chiusura** e che consiste del corpo della funzione e dell'ambiente E al momento dell'esecuzione dell'LDF. L'ambiente al caricamento è in generale diverso da quello che sarà presente al momento in cui la funzione viene invocata. Quindi ricordare l'ambiente permette di fare lo **static binding** per le eventuali variabili globali nel corpo della funzione. L'effetto di `LDF` sulla SECD è il seguente:

  ```
  S  E  (LDF C' C)  D --> ((C' E) S) E  C  D
  ```

  Come si vede viene inserita sullo stack la chiusura (C' E).

È arrivato il momento di spiegare le istruzioni SECD `AP` (`AP` sta per apply=invoca) e `RTN` (per return). L'istruzione `AP` ha il seguente effetto:

```
((C' E')A  S) E  (AP C)  D -->  NIL  (A E')   C'  ((S E C) D)
```

Intuitivamente questa istruzione carica la funzione sul controllo (per eseguirla) e costruisce l'ambiente in cui fare questa esecuzione e questo è il punto da capire. Inoltre salva sul dump la tripla (S E C) in modo da ripristinarla quando l'esecuzione della funzione finisce e qui possiamo già dire che il corpo della funzione deve sempre terminare col l'istruzione `RTN` che si occupa di fare questo ripristino. Ma vediamo prima la costruzione dell'ambiente A E': E' è la seconda componente della chiusura e quindi l'ambiente al momento del caricamento della funzione sullo stack, mentre A è la lista dei parametri attuali dell'invocazione. A viene inserito come prima lista del nuovo ambiente il che causa il fatto che i parametri formali della funzione troveranno il loro valore nell'ambiente con la coppia `<0,k>`. Questo va considerato quando si compila il corpo della funzione, confronta con la traduzione di una lambda-formula nella Sezione 4. La compilazione di una lamba-formula è tale che la funzione termina sempre con l'istruzione `RTN` il cui effetto è il seguente:

```
(x) E'  (RTN)  (S E C D) --> (x S)  E  C  D
```

Si osservi che l'ambiente E' viene eliminato e che il valore x che è il risultato calcolato dalla funzione viene messo a disposizione del calcolo che segue inserendolo in cima allo stack ripristinato (x S). Il calcolo continua con il programma C.

4 Traduzione lisp $\rightarrow$ linguaggio SECD

Il compilatore COMP da realizzare deve consistere di una funzione ricorsiva:

```
COMP(e: Sexpr, n: string list list, c: secdexpr list): secdexpr list
```

i cui parametri hanno il seguente significato:

- `Sexpr e` è il programma Lisp in FS che deve essere tradotto in linguaggio SECD; il risultato di COMP è il programma SECD prodotto;

- `string list list n` è la lista dei nomi (o variabili) che possono occorrere liberi in `e`. All'inizio della compilazione, normalmente, `n` sarà `[]`, e nei momenti successivi della compilazione conterrà le variabili legate dalle istruzioni `let` e `lambda` nel cui scope si trovano le istruzioni che vengono compilate in quel momento.

  A lezione abbiamo visto che la relazione tra le variabili che compaiono nel programma Lisp e l'indirizzo dell'r-valore della variabile che serve nel codice SECD corrispondente viene costruita attraverso l'uso di 2 liste di liste che chiameremo `n` e $\hat{v}$ (si osservi che n sta per nomi e v per valori). La lista `n` viene usata durante la compilazione e ad ogni istante della compilazione è una lista di liste delle variabili legate nella parte di programma già compilato e quindi questa lista contiene tutte le variabili libere della parte di programma che resta da compilare. Si noti che il tipo `string list list` di `n` è coerente con quanto detto. Invece, la lista `v` è in realtà contenuta nel registro ambiente E della SECD durante l'esecuzione del programma SECD. La compilazione infatti deve avere cura di produrre un programma SECD che eseguendo costruisce su E un ambiente che ha la stessa struttura della lista di nomi `n` usata durante la compilazione. Visto che durante la compilazione `n` cambia e durante l'esecuzione E cambia, si deve osservare che l'esecuzione è sempre l'esecuzione di un'istruzione prodotta dalla compilazione e questo collega momenti della compilazione (col loro `n`) e momenti dell'esecuzione (col loro E) e le cose sono organizzate in modo tale che l'`n` e la E collegati come appena descritto abbiano esattamente la stessa struttura. Quindi se il nome `x` si trova nella posizione `<n1,n2>` di `n` al momento in cui la compilazione produce il codice di un'istruzione che usa `x`, siamo sicuri che all'esecuzione di questa istruzione, il valore di `x` sia nella posizione `<n1,n2>` dell'ambiente E.

- il terzo parametro `secdexpr list c` contiene ad ogni istante della compilazione il codice SECD prodotto fino a quell'istante. Non è indispensabile, ma rende la definizione di COMP più agevole. Da quanto detto segue che inizialmente `c=[]`. Procedendo con la compilazione, alla lista di istruzioni SECD `c` vengono aggiunte nuove istruzioni in testa alla lista stessa. Questo può sembrare strano per chi è abituato ai programmi imperativi in cui la compilazione inizia creando il codice che corrisponde all'inizio del programma e procede appendendo le successive istruzioni alla destra. I programmi Lisp sono espressioni e la FS ci permette di accedere immediatamente alle diverse componenti dell'espressione in modo da compilarle nell'ordine giusto che non è necessariamente quello da sinistra a destra. Per esempio se il programma Lisp consiste di un'invocazione di una lambda-astrazione con certi parametri attuali, verrà per prima cosa prodotto il codice della lambda-astrazione e successivamente si aggiungeranno (in testa alla lista `c`) i codici che calcolano i parametri attuali. Quindi quando verrà eseguito il codice SECD prodotto, verranno innanzitutto calcolati i parametri attuali e poi eseguita la funzione.

Elenchiamo qui sotto la relazione che descrive per ogni espressione Sexpr la corrispondente traduzione in un programma SECD. **È importante capire che il programma COMP deve realizzare esattamente questa traduzione.**

Consideriamo i diversi casi di espressioni Sexpr. Indichiamo con | il concatenamento di liste e con :: l'istruzione di `cons`, cioè `x::[x_1,...,x_k]=[x, x_1, ... ,x_k]` e `[x, y] | [z, w]=[x, y, z, w]`.

- **Sexpr e $\longrightarrow$ comp(e,n,[])**

- `Var(''x'')`$\longrightarrow$ `[LD(location(x: string,n: string list list))]`. Alla variabile `x` nel programma Lisp deve corrispondere il caricamento nello stack S del suo valore preso dall'ambiente E . Il calcolo della posizione che il valore avrà in E durante l'esecuzione è realizzato attraverso la funzione `location(x: string,n: string list list):int*int` che prende in input il nome della variabile e la doppia lista dei nomi `n` e restituisce la posizione della variabile nella doppia lista. I rapporti tra `n` ed E è spiegato all'inizio di questa sezione.

  La funzione `location` calcola la coppia di interi (i, j) dove i è il numero della sottolista a cui appartiene la variabile e j è la sua posizione all'interno della sottolista. Per esempio, data la variabile X e la doppia lista dei nomi `n=[[''M'' ''N''],[''C'' ''F'' ''G''], [''X'' ''Y'']]`, `location(''X'',n)=(2,0)`, che indica che `X` è il primo elemento della terza sottolista di `n`. Si noti che la numerazione delle sottoliste e degli elementi all'interno delle sottoliste comincia da 0. Quindi a `Var(''X'')` in Lisp corrisponde il programma SECD `[LD(2,0)]`.

- `QUOTE(k)` $\longrightarrow$ `LDC(k)`. La dichiarazione di una costante corrisponde in linguaggio SECD al suo caricamento sullo stack, senza essere valutata. Esempio: `QUOTE(NUM(3))` diventa `LDC(NUM(3))`.

- `Op(''ADD'' ,[e_1,e_2])` $\longrightarrow$ `COMP(e_1,n,[]) | COMP(e_2,n,[]) | [ADD]`
  Esempio: `Op(''ADD'', [Var(''X''), Var(''Y'')])` diventa (usando la `n` del punto precedente) `[ LD(2,0), LD(2,1), ADD]`;

- I casi degli altri operatori aritmetici SUB, MUL, eccetera sono simili al precedente e quindi tralasciati.

- `Op(''EQ'' ,[e_1,e_2])` $\longrightarrow$ `COMP(e_1,n,[]) | COMP(e_2,n,[]) | [EQ]`

- Gli altri operatori LEQ, CAR, CDR, CONS e ATOM sono tralasciati.

- `(If (e_1, e_2, e_3)` $\longrightarrow$
  `COMP(e_1,n,[]) | [SEL(comp(e_2,n,[])|[JOIN], comp(e_3,n,[])|[JOIN])]`

  Esempio: Il programma Lisp

`(ADD Y (IF (LEQ X Y) X (QUOTE 1)))` in FS corrisponde alla seguente `Sexpr`:

`Op(''ADD'',[Var(''Y''), If(Op(''LEQ'',[Var(''X''),Var(''Y'')]), Var(''X''), QUOTE(NUM(1)))])`. La sua compilazione con lista dei nomi `n=[[''X'' ''Y'']]`, dà il seguente programma SECD:

`[LD(0,1), LD(0,0), LD(0,1), LEQ, SEL([LD(0,0), JOIN],[LDC(NUM(1)), JOIN]), ADD]`.

- `Lambda([x_1... x_k], e)` $\longrightarrow$ `[LDF(COMP(e, [x_1... x_k]::n, [RTN]))]`.

  In altre parole, la traduzione in linguaggio SECD di una lambda-espressione Lisp è l'applicazione dell'operatore `LDF` (LOad Function) alla lista di `secdexpr` che è la traduzione in SECD del corpo della funzione `e`, utilizzando come lista dei nomi la lista che si ottiene da quella data in input, `n` aggiungendo in testa la lista costituita dai nomi dei parametri formali `[x_1...x_k]`.

  Esempio: Il programma Lisp `LAMBDA (X) (ADD X Y)` in FS corrisponde alla seguente `Sexpr`:

  `Lambda([''X''],Op(''ADD'', [Var(''X''), Var(''Y'')]))`, che compilata con lista dei nomi `n=[[''X'' ''Y'']]` dà il seguente programma SECD: `[LDF([LD(0,0),LD(1,1), ADD, RTN])]`.

  Si osservi che in questo esempio abbiamo che `(x_1...x_k]=[''X'']` e `e=Op(''ADD'', [Var ''X'', Var ''Y''])`, e dunque la nuova lista dei nomi in cui compilare `e` è `n=[[''X'']['X'' ''Y'']]`. Infatti, nel valutare `e`, a `Var(''X'')` corrisponde `LD(0,0)`, che indica il primo elemento della prima lista in `n`, mentre a a `Var ''Y''` corrisponde `LD(1,1)` cioè il secondo elemento della seconda lista in `n`.

  Ricordiamo che l'effetto dell'esecuzione di `LDF` sulla SECD è quello di posizionare in cima alla pila il valore della funzione cioè la chiusura composta dalla coppia `<P,E>` dove `P` è la compilazione del corpo della funzione (naturalmente tradotto in SECD) ed `E` è l'ambiente della SECD quando viene eseguita l'operazione `LCD`. Per maggiori chiarimenti si veda la descrizione di `LDF` nella Sezione 3.

- `Call(e,[e_1...e_k])` $\longrightarrow$ `[LDC(NIL)]| COMP(e_k, n,[]) | [CONS] |...|COMP(e_1, n, [])|[CONS]|COMP(e, n, []) | [AP]`.

  Si può capire il programma SECD prodotto, osservando gli effetti della sua esecuzione sulla SECD. Innanzitutto provvede a caricare sullo stack S una lista contenente i valori dei parametri attuali dell'invocazione. Successivamente l'esecuzione del corpo di `e` (che deve essere una funzione, altrimenti il programma sarebbe sbagliato) caricherà su S una chiusura e finalemnte il comando AP eseguirà la funzione nell'ambiente giusto salvando l'attuale stato della SECD sul dump D. La descrizione di AP nella Sezione 3 può risultare utile per capire questa traduzione.

Esempio: Il seguente programma Lisp FS, `Call (Lambda([''X''],Op(''ADD'', [Var(''X''), Var(''Y'')])), [ QUOTE(NUM(2))])` con lista dei nomi `n=[[''X'' ''Y'']]` viene tradotto nel seguente programma SECD: `[LDC(NIL), LDC(NUM(2)), CONS, LDF([LD(0,0), LD(1,1), ADD, RTN]), AP]`.

Quando questo programma viene eseguito dalla SECD, prima viene caricata sulla pila S la costante `NIL` seguita da `NUM(2)` poi si esegue il `CONS` che mette su S la lista `[NUM(2)]` dei parametri attuali. Infine `LDF` carica su S la chiusura della funzione e cioè il corpo della funzione `[LD(0,0), LD(1,1), ADD, RTN]` e l'ambiente corrente ed `AP` eseguirà la funzione stessa.

- La traduzione dell'espressione `Sexpr Let` è estremamente simile a quella della `Call` descritta al punto precedente. Consideriamo il seguente programma Lisp FS: `Let(e, [''x1'', ''x2''],[e_1, e_2])` che corrisponde al programma Lisp `Let x1=e1 and x2=e2 in e`. Esso deve dare praticamente la stessa compilazione di `Call(e,[e_1 e_2])`. Precisamente la traduzione è la seguente: `Let(e,[''x_1'',...,''x_k''],[e_1...e_k])` $\longrightarrow$ `[LDC(NIL)]| COMP(e_k, n,[]) | [CONS] |...|COMP(e_1, n, [])|[CONS]| [LDF(COMP(e, [''x_1'',...,''x_k'']::n, [RTN]))] | [AP]`.

5 Consegna

Si chiede di realizzare il programma COMP usando il linguaggio ML. Il programma deve venire consegnato 5 giorni lavorativi prima dell'orale in cui si intende sostenere l'esame. La consegna pu avvenire via mail all'indirizzo: gilberto@math.unipd.it

Per domande sul progetto potete consultare, oltre a me (ricevimento=lunedì 14:30-16:30), anche la dott.ssa Brent Venable (mercoledì 11-13 ufficio n.4 p.t., mail=kvenable@math.unipd.it).

Per facilitare il compito viene fornita la seguente soluzione parziale:

```
fun COMP(e: Sexpr, n: string list list, c: secdexpr list): secdexpr list=
   case e of
        Var x => LD(location(x,0,n))::c |
// Il secondo parametro 0 e' aggiunto rispetto alla specifica di
//location data in precedenza nella Sezione 4 (Traduzione di Var(''x''))
// Esso serve a contare la sottolista  di n che si sta considerando

        Quote k => LDC(k)::c |

        Op(operator,sl) =>
            (case operator of
                "ADD" => COMP(hd(sl),n,COMP(hd(tl(sl)),n,(ADD::c))) |
                "SUB" =>
                "MUL" =>
                "DIV" =>
                "REM" =>
```

```
            "EQ" =>
            "LEQ" =>

            "CAR" => COMP(hd(sl),n,(CAR::c))|
            "CDR" =>
            "ATOM" =>

            "CONS" => COMP(hd(tl(sl)),n,COMP(hd(sl),n,(CONS::c)))
            ) |

        If(cond,e1,e2) =>
            let
              val thenpt = COMP(e1,n,[JOIN])
              val elsept = COMP(e2,n,[JOIN])
            in
              COMP(cond,n,SEL(thenpt,elsept)::c)
            end |

        Lambda(...) =>
                |
        Call(...) =>
                |
        Let(...) =>

                end;
```

Seguono alcuni esempi di compilazione operata da COMP:

```
ESEMPIO 1

(*Calcola (lambda xyz.(*x+y*y+z*z)/3)*)
xCOMP(Let(
Call(Lambda(["P","Q","R"],Op("DIV",[
Op("ADD",[Op("MUL",[Var "P", Var "P"]),
Op("ADD",[Op("MUL",[Var "Q", Var "Q"]),Op("MUL",[Var "R", Var
"R"])])]), Var "N"])),[Var "X", Var "Y", Var "Z"]),["N"],[Quote (NUM
3)]), [["X","Y","Z"]], []);

Compilata:

[LDC NIL, LDC(NUM 3), CONS, LDF [LDC NIL, LD(1, 2),
CONS, LD(1, 1), CONS, LD(1, 0), CONS, LDF [LD(0, 0), LD(0, 0),
MUL, LD(0, 1), LD(0, 1), MUL, LD(0, 2), LD(0, 2), MUL, ADD,
ADD, LD(1, 0), DIV, RTN], AP, RTN]] : secdexpr list
```

ESEMPIO 2

```
(* se x<y fa y-x/2 altrimenti x-y/2 *)
COMP(Let(
     If(Op("LEQ", [Var "X", Var "Y"]),
        Call(Lambda(["P","Q"],Op("DIV",[Op("SUB",[Var "P", Var "Q"]),
        Var "N"])), [Var "X", Var "Y"]),
        Call(Lambda(["P","Q"],Op("DIV",[Op("SUB",[Var "P", Var "Q"]),
        Var "N"])), [Var "Y", Var "X"])),["N"],[Quote (NUM 2)]),
     [["T","S"],["X","Y"]],[]);
```

Compilata:

```
[LDC NIL, LDC(NUM 2), CONS,
     LDF [LD(2, 0), LD(2, 1), LEQ,
          SEL([LDC NIL, LD(2, 1), CONS, LD(2, 0), CONS,
               LDF [LD(0, 0), LD(0, 1), SUB, LD(1, 0), DIV, RTN], AP, JOIN],
              [LDC NIL, LD(2, 0), CONS, LD(2, 1), CONS,
               LDF [LD(0, 0), LD(0, 1), SUB, LD(1, 0), DIV, RTN], AP, JOIN]),
          RTN]] : secdexpr list
```

ESEMPIO 3

```
(* crea la lista [Z+N,Y+N,X+N] *)
COMP(Let(
Op("CONS",[
Op("CDR",[Op("CDR",[Op("CONS", [Op("ADD",[Var "X", Var "N"]),
Op("CONS",[Op("ADD",[Var "Y", Var "N"]), Op("ADD",[Var "Z", Var
"N"])])])])]),
Op("CONS",[
Op("CDR",[Op("CONS", [Op("ADD",[Var "X", Var "N"]),
Op("CONS",[Op("ADD",[Var "Y", Var "N"]), Op("ADD",[Var "Z", Var
"N"])])])]),
Op("CAR",[Op("CONS", [Op("ADD",[Var "X", Var "N"]),
Op("CONS",[Op("ADD",[Var "Y", Var "N"]), Op("ADD",[Var "Z", Var
"N"])])])])]),["N"],[Quote (NUM 5)]),
[["T","S"],["X","Y", "Z"]],[]);
```

Compilata:

```
[LDC NIL, LDC(NUM 5), CONS,
```

```
LDF [LD(2, 2), LD(0, 0), ADD, LD(2, 1), LD(0, 0), ADD, CONS, LD(2, 0),
     LD(0, 0), ADD, CONS, CAR, LD(2, 2), LD(0, 0), ADD, LD(2, 1),
     LD(0, 0), ADD, CONS, LD(2, 0), LD(0, 0), ADD, CONS, CDR, CONS,
     LD(2, 2), LD(0, 0), ADD, LD(2, 1), LD(0, 0), ADD, CONS, LD(2, 0),
     LD(0, 0), ADD, CONS, CDR, CDR, CONS, RTN]] : secdexpr list
```