Esercizi

Grammatiche, scope, Haskell

Grammatiche ambigue

- descrivere il linguaggio generato dalle seguenti grammatiche, mostrare che grammatiche sono ambigue, esibendo due alberi di derivazione distinti per la stessa stringa:

```
S ::= a | SS

S ::= () | (S) | S S

S ::= a | S + S

S ::=   | a  | aR
R ::= b | bS
```

- proporre delle grammatiche non-ambigue equivalenti
- proporre delle espressioni regolari equivalenti, se esistono
- proporre degli NFA equivalenti

Soluzione . . .

Linguaggi regolari e non

Sia A l'alfabeto definito da $A = \{0, 1\}$, determinare i linguaggi regolari tra i seguenti:

- $L_1 = \{w \mid w \in A^*, \text{ ha lunghezza pari}\}$
- $L_2 = \{w \mid w \in A^*, \text{ contiene un numero pari di } 0\}$
- $L_3 = \{w \mid w \in A^*, \text{ la differenza tra il numero di 0 e 1 è un numero pari }\}$
- $L_4 = \{w \mid w \in A^*, \text{ contiene lo stesso numero di 0 e 1 }\}$
- $L_5 = \{ww \mid w \in A^*\}$
- $L_6 = \{0^n 1^m\}$
- $L_7 = \{0^n 1^m \mid n < m\}$

Descrivere ogni linguaggio regolare mediante espressione regolare e NFA

Provare che $L_1 = L_3$

Soluzione . . .

Linguaggi e grammatiche

- Data la grammatica

```
S ::= S 0 | R 1
R ::= R 0 | S 1 | 1
```

- Stabilire se è ambigua.
- Nel caso lo sia, definire una grammatica non ambigua equivalente.
- Definire il linguaggio generato.
- Stabilire se è un linguaggio regolare.
- Nel caso, descriverlo come espressione regolare e come NFA.
- Usando Lex-Alex YACC-Happy, costruire un programma riconoscitore.

Soluzione . . .

Linguaggi e grammatiche

- Ripetere l'esercizio precedente considerando la grammatica

```
S ::= a | S b S
```

Linguaggi e grammatiche

- A partire dalla grammatica degli identificatori `Id` e delle espressioni `Exp`, definire, una sottogrammatica, non ambigua, di C,capace di generare:
  - istruzioni di assegnamento
  - sequenze di istruzioni di assegnamento
  - blocchi
  - cicli for

Linguaggi e grammatiche

- definire i numeri divisibili per 2 in base 3, come automa, come espressione regolare e come grammatica regolare
- definire i numeri divisibili per 3 in base 2, come automa, come espressione regolare e come grammatica regolare
- riconoscitore Lex o Alex che cataloga le sequenze di numeri ternari in base alla parità

Linguaggi e grammatiche

- Sia:
  - `L1 := {xwy | x w = wR y,  x,y ∈ {a,b}, w ∈ {a,b}* }`
  - `L2 := {w | w = wR , w ∈ {a,b}* }`
  - dove wR è la stringa w rovesciata
- si diano le stringhe di L1 ed L2 di lunghezza ≤ 4.
- si descrivano i due linguaggi
- si diano due grammatiche non ambigue, con simboli iniziali S1 ed S2 per generare L1 ed L2
- si dica se la grammatica ottenuta unendo le due precedenti e la produzione S $\rightarrow$ S1 | S2 è ambigua

Soluzione . . .

Linguaggi e grammatiche

- Date le produzioni:

```
S1 → a | a S1 a | b A b
S2 → S1 a | A
A → ε | a A a | b A b
S  → S1 | S2
```

- e sia `LR` il linguaggio `{v | v = vR , v ∈ {a, b}* }`
- si diano le stringhe di `L(S1)` e `L(S2)` di lunghezza ≤ 4
- determinare se `L(A)` e `LR` coincidono
- descrivere a parole i linguaggi `L(S1 ), L(A)`
- si caratterizzino tutte le stringhe di `LR \ L(S1 )`
- determinare quali tra le grammatiche aventi come simboli iniziali `S1, S2, A` e `S` sono ambigue
- infine si caratterizzino tutte le stringhe ambigue nella grammatica con simbolo iniziale `S`

Soluzione . . .

Linguaggi e grammatiche

Analizzare la grammatica (con P simbolo iniziale):

$$\texttt{P} \rightarrow \quad \texttt{a D} \mid \texttt{b P}$$
$$\texttt{D} \rightarrow \varepsilon \mid \texttt{a P} \mid \texttt{b D}$$

Linguaggi e grammatiche

- Si dia la grammatica delle stringhe palindrome, costruite a partire dai caratteri `a` e `b`.
  - Si consideri il problema di determinare se il linguaggio generato può essere riconosciuto da un automa a pila.
    - non deterministico
    - deterministico

Scope esercizi 2/7/19

Scope dinamico, deep binding, r-value prima di l-value, espressioni argomenti da sinistra a destra, indici vettore da 1:

```
int w[3]={2,4,6}, y=3, z=5;
int foo(valres int[] v, name int z){
  foreach(int x:v)
    {write(v[y--] = x + z)};
  return v[z];
}
write(foo(w, y + z--));
write(w[z]);
```

segue

```
int x = 5;
int y = 7;
void P(ref int x, int z, int R(name int)){
  z = R(x + y);
  write(x, y, z);
}
int Q(name int w){
  return(w + x++);
}
P(y, x, Q);
write (x, y);
```

Scope compito 17/6/19

Scoping statico, assegnamento che calcola prima l'r-value e poi l'l-value, valutazione delle espressioni da sinistra a destra e indici vettori inizianti da 0

```
int i=2, j=1, v[3]={4,3,5};
int foo(val int i, ref int z){
  while ((v[i--] += v[i]) < 15){
    z = v[++i] + (v[j]++);
    write (v[i]);};
  return i;
};
write(foo(j++, v[j--]));
write(v[i],j);
```

segue

```
int x=1;
int y=0;
void P(valres int y, ref int z, int R(name int)){
  z = y++ + R(x + y);
  write(x, y, z);
  z = R(z++);
}
{
  int x =3;
  int Q(name int w){
    return(w + x);
  }
  P(x, y, Q);
  write(y, x++);
}
write(y, x);
```

Scope

- Si mostri l'evoluzione delle variabili e dell'output del seguente frammento di programma C-like con: assegnamento che calcola l'r-value prima dell'l-value, indici dei vettori che iniziano da 0, valutazione degli argomenti da sinistra a destra:

```
{
char x[10] = "abcdefghij" ;
int i = 4 ;
char magic(ref char y, ref int j){
    char c = y ;
    x[j] = x[j--] = x[++i];
    write(y) ;
    return c ;
}
write(magic(x[i++], i) );
write(x[i++], i--);
}
```

Scope

Si mostri l'evoluzione delle variabili e l'output del seguente frammento di programma C-like con: assegnamento che calcola r-value dopo l-value, espressioni da destra a sinistra, argomenti chiamate da destra a sinistra e indici vettori inizianti da 1:

```
int v[5]={5,4,3,2,1}, i=3, j=2;
void f(name int k, valres int[] w, val int h)
{for (int i=j++ to j++)
   write(w[i] += v[i] *= ++w[j]-h );
}
f(v[--j]++,v,v[i--]++);
write(v[1]+v[2]);
```

Scope

Si mostri l'evoluzione delle variabili e l'output del seguente frammento di programma C-like con: assegnamento che calcola r-value dopo l-value, espressioni da sinistra a destra e indici vettori inizianti da 1

```
int v[3]={2,4,6}, i=-4, j=5;
int f(valres int k, name int h) {
  int i = 0;
  for (int i = h to h+k-4)
    write(v[i] += (j *= i));
  return k++;
}
write(f(j,i++ + j) + ++j);
write(v[i+3], i++, j);
```

Stack di attivazione

Si mostri l'evoluzione dello stack di attivazione e l'output del seguente programma espresso C-like con:

- scope dinamico e shallow binding,
- assegnamento che calcola l-value dopo r-value,
- valutazione delle espressioni da destra a sinistra,
- argomenti chiamate da destra a sinistra e
- indici dei vettori inizianti da 0 (% è il modulo aritmetico, $-1\%3 = 2$).

segue

```
{
int x=1, y=5 , v[]={ y++, x + y, --x };

int F (valres int y, ref int z ) {
  int v []= { --z, y ++ + z , x };
  return ( v[0] + v[1] + v[2]);
}
int G(int H(valres int, int ref), name int z){
  int x = 2;
  write(v[ z %3]);
  return(H(y, x) + x++);
}
{
int x = 4;
write(v[(x ++)%3]++);}
write( G(F, x + y));
}
```

Stack di attivazione

Si mostri l'evoluzione dello stack di attivazione e l'output del seguente programma espresso C-like con:

- scope statico e shallow binding,
- assegnamento che calcola l-value dopo r-value,
- valutazione delle espressioni da destra a sinistra,
- argomenti chiamate da destra a sinistra e
- indici dei vettori inizianti da 0 (% è il modulo aritmetico, $-1\%3 = 2$).

segue

```
int x=4, y=2 ,v[]={x-- , y , ++ y };
int F(int R(valres int), ref int z){
  int x=3;
  write(R(z));
  return(z -= x);}
{
int y=6, v[] = {x-- ,x, --y};
int G(name int x){
  int H(valres int w){
    return(v[(w++)%3]);
  }
  write(v[(y++) %3] );
  return(F(H, x) - x++);
}
write(G(v[x %3]));
}
```

Scope

Si mostri l'evoluzione delle variabili e l'output del seguente frammento di programma C-like con:

```
int x = 3, y = 2;
int foo (name int y){
    int x = 2;
    y += x;
    write (y);
    y++;
}
foo (x);
foo (y);
```

Stack di attivazione

Si mostri l'evoluzione dello stack di attivazione del seguente frammento di programma in un linguaggio C-like con:

- assegnamento che calcola prima r-value poi l-value,
- espressioni da destra a sinistra,
- argomenti chiamate da destra a sinistra
- indici vettori inizianti da 0.
- % è il modulo aritmetico, `-1%3 = 2`;
- nel foreach ad ogni passo di iterazione al parametro w viene assegnato un elemento di a con semantica per riferimento.

Codice

```
int a[3]={1, 2, 3}, y=3, x=6;
int f(ref int x, name int k){
  foreach (int w : a)
    write (a[(--x )%3] += --w + k);
  return ++x ;
}
write(f(x, y + --x ));
write(++a[y %3], a[(++ x )%3], x);
```

Stack di attivazione

Si mostri l'evoluzione delle variabili e l'output del seguente frammento di programma espresso in un linguaggio C-like con:

- scope dinamico, deep binding,
- assegnamento che calcola prima l-value e poi r-value
- valutazione delle espressioni da sinistra a destra:

Si mostri infine l'evoluzione del CRT

codice

```
int x=1 ,y=2;
int F(name v){
  y = x + v;
  write(x, y, v);
  x += v;
  return v++;
}
int Q(ref int x, int R(name int)){
  int y = R(x++);
  y += x;
  write(y);
  return (--x + y);
}
write (Q(y, F));
write (x , y);
```

codice

```
int x=-3, y=1 , z=2;
int F(valres int z, name v){
  z = ++x;
  write(x, y, z);
  y += z;
  write(v);
  return z++;
}
int Q(name int v , ref int x, int R(valres int, name int)){
  int w = R(y, v+x);
  y = w+x; write(y);
  return (--x + w);
}
write (Q( x++, y, F));
write (x ,y);
```

Stack attivazione

Si mostri l'evoluzione dello stack di attivazione e l'output del seguente frammento di programma espresso in un linguaggio C-like con:

- scope statico,
- assegnamento che calcola prima l-value e poi r-value
- valutazione delle espressioni da sinistra a destra

Si mostri infine l'evoluzione del display.

Codice

```
int x=2, y= 5;
void G(val int w, int R(ref int)){
  int y = w++;
  write(R(y), ++w);
}
void F(name int y){
  int x = ++ y;
  int H(ref int w){
    return(w++ + y);
  }
  G(H(x), H);
}
F(x)
```

Eccezioni

Descrivere il comportamento del seguente programma con ognuno dei diversi meccanismi di passaggio dei parametri: valore, riferimento, costante, valore-risultato, nome. La valutazione delle espressioni viene fatta da sinistra a destra.

```
{ int y=1
  void f(___ int x) throws E(){
    if (x++ = y++)
      {throw new E();}
    x++;
  }
  try{ f(y) } catch (E){write(y++)};
  write (y);
}
```

Eccezioni

Descrivere il comportamento del seguente programma:

```
void ecc() throws E() {
  throw new E();
  }
void f(int x) throws E() {
  if (x == 0) {ecc();}
  try {ecc();} catch (E) {write(2);}
  }
void main {
  try {f(0);} catch (E) {write(0);}
  try {f(1);} catch (E) {write(1);}
  }
```

Matrici

- In una matrice `bool A [8][5]`, memorizzata per righe, colonne, con indici che iniziano da 1, in che posizione di memoria, rispetto all'indirizzo base, b, si trova l'elemento `A[2][4]`. Si supponga che ogni intero richieda 4 locazioni di memoria.
- In una matrice `int A [4][8][16]`, memorizzata per piani, righe, colonne, con indici che iniziano da 0, in che posizione di memoria, rispetto all'indirizzo base, b, si trova l'elemento `A[2][4][8]`. Si supponga che ogni intero richieda 4 locazioni di memoria.
- In una matrice quadrata 5x5, quali elementi sono memorizzati sempre nella stessa posizione, sia che la matrice sia memorizzata per righe che per colonne?
  - In una matrice rettangolare 2x4.
  - 3 x 5.
  - 4 x 7.
  - Generalizzare.

Matrici

- Gli array multidimensionali in Java sono memorizzati come righe di puntatori (row pointer). Determinare le istruzioni assembly ARM per accedere all'elemento:
  - `A[2][4]` di un vettore `int A [4][8]`.
  - `A[2][4][8]` di un vettore `int A [4][8][16]`.

Dimensioni dati

Nelle ipotesi di parole di memoria di 4 byte, memorizzazione a parole allineate, codice caratteri ASCII (UNICODE), quanti byte occupano i seguenti record:

```
struct {
int i;
char a[5];
int j;
char b;
}
```

```
struct {
int i;
char a[5];
char b;
int j;
}
```

Relazioni tra tipi.

L'equivalenza tra tipi in C è strutturale, in generale, ma per nome sul costrutto `struct`.

Determinare, nelle definizioni seguenti, quali variabili sono equivalenti per tipo rispetto a

- equivalenza per nome
- equivalenza per nome lasca
- equivalenza tra tipi in C
- equivalenza strutturale

Definizioni A

```
typedef int integer;
typedef struct{int a; integer b} pair;
typedef struct coppia {integer a; int b} coppia2;
typedef pair[10] array;
typedef struct coppia[10] vettore;
typedef coppia2[10] vettore2
int a;
integer b;
pair c;
struct coppia d;
coppia2 e;
vettore f;
vettore2 g;
coppia[10] i, j;
array k;
```

Definizioni B

```
typedef bool  boolean;
typedef boolean[32] parola
typedef struct{parola a; parola b} coppia
typedef coppia pair

parola a;
boolean[32] b;
bool[32] c;
coppia d
pair e;
struct{parola a; parola b} f
struct{parola b; bool[32] a} g
struct{parola a; bool[32] b} h
```

Conversioni di tipo

Che valore assume la variabile i al termine dell'esecuzione del seguente codice:

```
int i = 2;
float f = i;
union Data {
 int i;
 float f;
} data;
data.f = f;
i = data.i;
```

Tipi polimorfi

Trovare il tipo polimorfo più generale per i seguenti funzionali:

```
(define G (lambda (f x)((f(fx)))))
(define (G f x)(f(f x)))
(define (G f g x)(f(g x)))
(define (G f g x)(f(g (g x))))
(define (H t x y)(if (t x y) x y))
(define (H t x y)(if (t x) x (t y)))
(define (H t x y)(if (t x) y (x y)))
(define (H t x y)(if (t x y) x (t y)))
```

Haskell

Definire le funzioni che risolvano i seguenti problemi. Per ogni funzione definita, comprese quelle ausiliarie, descrivere il relativo tipo Haskell.

- Date due liste, costruire la lista di tutte le possibili coppie: elemento della prima lista, elemento della seconda.
- Controllare se due liste sono l'una una permutazione dell'altra. L'unica operazione permessa sugli elementi della lista è il test di uguaglianza. Si definisca inoltre il tipo Haskell della funzione definita e delle funzioni ausiliarie.

Haskell

Il grafo di una funzione parziale f : A $\rightarrow$ B è definito come l'insieme di coppie {(a, b) | b = f (a)}, ossia l'insieme di coppie in cui i primi elementi costituiscono il dominio della funzione e i secondi elementi definiscono il comportamento della funzione, sui corrispondenti primi elementi.

- Scrivere una funzione Haskell che preso il grafo di una funzione f da A in B, rappresentato come lista di coppie, ed una lista l di elementi di tipo A, applica la funzione f a tutti gli elementi della lista. Si tenga presente che la funzione f può essere parziale (non definita su alcuni elementi), nel caso la lista l contenga un elemento su cui f non è definita, l'elemento viene rimosso dalla lista.
- Scrivere una funzione Haskell, che data una funzione f , definita sugli interi positivi, costruisce il grafo di f. Dare fino a quattro versioni diverse della soluzione, utilizzando: list comprehension, foldr, e map.

- Risolvere il punto precedente nel caso in cui f sia una funzione definita su tutti i numeri interi, quindi inserire nel grafo generato anche le coppie sul dominio dei numeri negativi. Scrivere una seconda versione utilizzando foldr.

Scrivere una funzione Haskell che, dato un lista di coppie, controlli che la lista non contenga due coppie con il primo elemento uguale.

Haskell

- Data una lista, costruire la lista, di liste, contenente tutte le possibili permutazioni della lista originaria.
- Data una matrice, vista come lista di liste, determinare se si tratta di una matrice quadrata.

Si considerino gli alberi generici definiti come:

```
data Tree a =  Tree a [Tree a]
```

- Dato un albero, definire la lista che rappresenta il suo cammino massimo.
- Dato un albero, calcolare il suo diametro.

Haskell

Si scrivano le seguenti funzioni Haskell:

- funzione che date due liste determina se una delle due è una sottolista dell'altra;
- una funzione che data una lista determina se questa è palindroma;
- una funzione che data una lista determina la lunghezza della più lunga sottolista finale palindroma;
- una funzione che data una lista determina la lunghezza della più lunga sottolista palindroma.

Si definisca il tipo di ogni funzione definita.

Haskell

Su liste aventi come elementi coppie di valori di uno stesso tipo ordinabile, scrivere le seguenti funzioni Haskell:

- una funzione che data una lista restituisce la lista contenente le sole coppie ordinate (il primo elemento della coppia ‘e minore del secondo);
- una funzione che data una lista di coppie le ordina, ossia scambia tra loro gli elementi di una coppie in modo che il primo sia minore del secondo;
- considerando l’ordine lessicografico tra le coppie, definire una funzione che data una lista di coppie la ordina.

Giocando sul fatto di usare o meno: le funzioni ’fold‘, la ricorsione di coda o le funzioni di libreria, fornire un’implentazione alternativa per ciascuna delle funzioni precedenti.

Haskell

Scrivere le seguenti funzioni in Haskell:

- Una funzione che, data lista, determini se questa sia palindroma.
- Una funzione che, data una matrice memorizzata per righe, determini se questa sia simmetrica rispetto all'asse verticale.
- Una funzione che, data una matrice memorizzata per righe, determini se questa sia simmetrica rispetto all'asse orizzontale.

Sistemi di assegnazione di tipo

Nel sistema di tipi per il linguaggio F1, descritto nei lucidi, costruire la derivazione di tipo per le seguenti espressioni. Nota, nel caso di alberi di derivazione piuttosto complessi, i primi passi di derivazione, a partire dagli assiomi, se elementari, possono essere omessi.

```
\f : (Nat -> Nat) . \g : (Nat -> Bool) .
                        \n : Nat . g ( f n )
```

```
 \ z : Bool * Bool . if (first z) then (True, (second z))
                                  else ((second z), False)
```

```
 \ z : Nat + Bool . case z of
                x : Nat then x + 2 |
                y : Bool then if y then 1
                                   else 0
```

```
\ x : (Ref Nat) . \ y : (Ref Nat) .
   x := (deref x) + (deref y)
```

TAS per linguaggio imperativo

```
{ Nat x = 1;
  Nat y = 2;
  if x < y then x = y else x = x  }
```

```
{ Nat x = 1;
  inc ( Nat z){
     x := x + z };
  inc (1);
  inc (x)
}
```

```
{Nat x = 2;
 swap(Nat x, y){Nat z = 0; z = y, y = x};
 while (x < 20) { swap(x, x)}
```

TAS per linguaggio imperativo