Raccolta esami SML con soluzione (Linguaggi di programmazione mod. 2, UNITN, prof. Kuper)

Questi appunti sono stati creati da Pater999.
Altri appunti/esercizi/esami/consigli si possono trovare in questa repository github.
L'ultima versione di questo file è scaricabile al seguente link.
Prima di leggere questo file consiglio di leggere il seguente Readme.

Indice esami

Esame giugno 2015

Testo

Come noto, un numero naturale è esprimibile in base agli assiomi di Peano usando il seguente tipo di dato:

```
datatype naturale = zero | successivo of naturale;
```

Usando tale tipo di dato, la somma fra numeri naturali è esprimibile come:

```
val rec somma = fn zero => (fn n => n)
                | successivo a => (fn n => successivo (somma a n));
```

Scrivere una funzione Standard ML, chiamata **prodotto**, che ha tipo `naturale -> naturale -> naturale` , che calcola il prodotto di due numeri naturali. Si noti che la funzione **prodotto** può usare la funzione somma nella sua implementazione.

Soluzione

```
val rec prodotto = fn zero => (fn b => zero)
                   | successivo(a) => (fn b => (somma b (prodotto a b)));
```

Testing (Non fa parte della soluzione - utile per capire)

Moltiplicazione 3x3=9

```
prodotto (successivo (successivo (successivo zero))) (successivo (successivo (successivo zero)));
```

Stamperà a video:

```
val it = successivo (successivo (successivo (successivo (successivo (successivo (successivo (successivo (successivo zero))))))): naturale
```

Moltiplicazione 2x0=0

```
prodotto (successivo (successivo zero)) ( zero);
```

Stamperà a video:

```
val it = zero: naturale
```

Esame luglio 2015

Testo

Si consideri il seguente tipo di dato, che rappresenta una semplice espressione avente due argomenti x e y:

```
datatype Expr = X
              | Y
              | Avg of Expr * Expr
              | Mul of Expr * Expr
```

dove il costruttore X rappresenta il valore del primo argomento x dell'espressione, il costruttore Y rappresenta il valore del secondo argomento y, il costruttore Avg, che si applica ad una coppia (e1, e2), rappresenta la media (intera) dei valori di e1 ed e2, mentre il costruttore Mul (che ancora si applica ad una coppia (e1, e2)) rappresenta il prodotto dei valori di due espressioni e1 ed e2.
Implementare una funzione Standard ML, chiamata **compute**, che ha tipo `Expr -> int -> int -> int` .
Come suggerito dal nome, **compute** calcola il valore dell'espressione ricevuta come primo argomento, applicandola ai valori ricevuti come secondo e terzo argomento e ritorna un intero che indica il risultato finale della valutazione.
IMPORTANTE: notare il tipo della funzione! Come si può intuire da tale tipo, la funzione riceve tre argomenti usando la tecnica del currying. É importante che la funzione abbia il tipo corretto (indicato qui sopra). Una funzione avente tipo diverso da `Expr -> int -> int -> int` non sarà considerata corretta.

Soluzione

```
val rec compute = fn X => (fn x => fn y => x)
                  | Y => (fn x => fn y => y)
                  | Avg(e1, e2) => (fn x => fn y => ((compute e1 x y) + (compute e2 x y)) div 2)
                  | Mul(e1, e2) => (fn x => fn y => (compute e1 x y) * (compute e2 x y));
```

Testing (Non fa parte della soluzione - utile per capire)

```
(compute X) 4 2; (* Restitusce 4 *)
(compute Y) 4 2; (* Restitusce 2 *)
compute (Avg(X,Y)) 4 2; (* Restitusce la media tra 4 e 2 = 3 *)
compute (Mul(X,Y)) 4 2; (* Restitusce moltiplicazione tra 4 e 2 = 8 *)
compute (Avg(X,Y)) 4 (compute (Mul(X,Y)) 4 2); (* Restitusce moltiplicazione tra 4 e 2 = 8 e poi fa la media tra 8 e 4 = 6*)
```

Esame agosto 2015

Testo

Scrivere una funzione Standard ML, chiamata **elementi_pari**, che ha tipo `'a list -> 'a list` . La funzione riceve come parametro una `a-lista` e ritorna una `a-lista` contenente gli elementi della lista di ingresso che hanno posizione pari (il secondo elemento, il quarto elemento, etc. . .).

Per esempio
`elementi_pari [1,5,2,10]` ritorna `[5,10]`

Si noti inoltre che la funzione **elementi_pari** non deve cambiare l'ordine degli elementi della lista rispetto all'ordine della lista ricevuta come argomento (considerando l'esempio precedente, il valore ritornato deve essere `[5,10]`, non `[10,5]`). Si noti che la funzione **elementi_pari** può usare i costruttori forniti da Standard ML per le alfa-liste, senza bisogno di definire alcun datatype o altro.

Soluzione

```
val rec elementi_pari = fn [] => []
                        | a::[] => []
                        | (a::b::l) => b::(elementi_pari l);
```

Testing (Non fa parte della soluzione - utile per capire)

```
elementi_pari []; (* Restituisce la lista vuota [] *)
elementi_pari [1]; (* Restituisce la lista vuota [] *)
elementi_pari [1,4,4,5,6]; (* Restituisce la lista vuota [4,5] *)
```

Esame settembre 2015

Testo

Si consideri il seguente tipo di dato:

```
datatype codice = rosso of string
                | giallo of string
                | verde of string;
```

che rappresenta un paziente in arrivo al pronto soccorso.
La stringa rappresenta il cognome del paziente, mentre i tre diversi costruttori rosso, giallo e verde rappresentano la gravità del paziente (codice rosso: massima urgenza, codice verde: minima urgenza).
Quando un paziente con codice rosso arriva al pronto soccorso, viene messo in lista d'attesa dopo tutti i pazienti con codice rosso (ma prima di quelli con codice giallo o verde); quando arriva un paziente con codice giallo, viene messo in lista d'attesa dopo tutti i pazienti con codice rosso o giallo (ma prima di quelli con codice verde), mentre quando arriva un paziente con codice verde viene messo in lista d'attesa dopo tutti gli altri pazienti.

Si scriva una funzione **arriva** (avente tipo codice `list -> codice -> codice list`) che riceve come argomenti la lista dei pazienti in attesa (lista di elementi di tipo codice) ed un paziente appena arrivato (elemento di tipo codice) e ritorna la lista aggiornata dei pazienti in attesa (dopo aver inserito il nuovo paziente nel giusto posto in coda).

Come esempio, l'invocazione:
`arriva [rosso "topolino", rosso "cip", giallo "ciop", verde "paperino", verde "pluto"] (giallo "clarabella");`
deve avere risultato
`[rosso "topolino", rosso "cip", giallo "ciop", giallo "clarabella", verde "paperino", verde "pluto"]`

Soluzione

```
val rec arriva = fn []           => (fn x => [x])
                  | (verde n)::l => (fn (verde nn)  => (verde n)::(arriva l (verde nn))
                                        |     x       => x::((verde n)::l))
                  | (giallo n)::l => (fn (verde nn) => (giallo n)::(arriva l (verde nn))
                                        | (giallo nn) => (giallo n)::(arriva l (giallo nn))
                                        |     x       => x::((giallo n)::l))
                  | (rosso n)::l => (fn     x       => (rosso n)::(arriva l x));
```

Testing (Non fa parte della soluzione - utile per capire)

```
arriva [rosso "topolino", rosso "cip", giallo "ciop", verde "paperino", verde "pluto"] (giallo "clarabella");
arriva [] (verde "pluto");

val listaAttesa = [];
val listaAttesa = arriva listaAttesa (verde "pluto");
val listaAttesa = arriva listaAttesa (verde "pippo");
val listaAttesa = arriva listaAttesa (giallo "pluto");
val listaAttesa = arriva listaAttesa (rosso "ciop");
val listaAttesa = arriva listaAttesa (verde "cip");
```

Esame giugno 2016

Turno 1 G2016

Testo

Si scriva una funzione **hist** (`avente tipo real list -> real * real -> int`) che riceve come argomento una lista di real l ed una coppia di real (c, d). La funzione **hist** ritorna il numero di elementi della lista compresi nell'intervallo (c - d, c + d), estremi esclusi (vale a dire il numero di elementi r tali che (c - d < r < c + d).

Come esempio, l'invocazione
`hist [0.1, 0.5, 1.0, 3.0, 2.5] (1.0, 0.5);`
deve avere risultato 1;
e `hist [0.1, 0.5, 1.0, 3.0, 2.5] (1.0, 0.6);`
deve avere risultato 2.

Soluzione

```
val rec hist = fn [] => (fn (c:real, d:real) => 0)
               | [e] => (fn (c:real, d:real) => if (e > (c-d) andalso e < (c+d)) then 1 else 0)
               | (e :: l) => (fn (c:real, d:real) => if (e > (c-d) andalso e < (c+d)) then 1 + hist l (c, d) else 0 + hist l (c, d));
```

Testing (Non fa parte della soluzione - utile per capire)

```
hist [0.1, 0.5, 1.0, 3.0, 2.5] (1.0, 0.5);
hist [0.1, 0.5, 1.0, 3.0, 2.5] (1.0, 0.6);
hist [] (2.4, 12.6);
hist [4.0] (4.1, 0.0);
```

Turno 2 G2016

Testo

Si scriva una funzione **noduplen** (avente tipo `''a list -> int`) che riceve come argomento una lista di ''a l. La funzione **noduplen** ritorna il numero di elementi della lista senza considerare i duplicati.

Come esempio, l'invocazione
`noduplen ["pera", "pera", "pera", "pera"];`
deve avere risultato 1;
`noduplen ["red", "red", "green", "blue"];`
deve avere risultato 3.

Soluzione

```
val rec noduplen = fn [] => 0
                   | a::[] => 1
                   | a::b => if (List.exists ((fn y => a = y)) b) then 0+(noduplen b) else 1+(noduplen b);
```

Testing (Non fa parte della soluzione - utile per capire)

```
noduplen ["pera", "pera", "pera", "pera"];
noduplen ["red", "red", "green", "blue"];
noduplen ["red"];
noduplen [];
noduplen [1,2,4,5,6,0,1,4,5];
noduplen [true, false, true, false];
noduplen [#"A",#"a",#"B",#"b"];
```

Esame luglio 2016

Turno 1 L2016

Testo

Si consideri il tipo di dato:

```
datatype lambda_expr = Var of string
                     | Lambda of string * lambda_expr
                     | Apply of lambda_expr * lambda_expr;
```

che rappresenta un'espressione del Lambda-calcolo.
Il costruttore Var crea un'espressione costituita da un'unica funzione / variabile (il cui nome è un valore di tipo string); Il costruttore Lambda crea una Lambda-espressione a partire da un'altra espressione, legandone una variabile (indicata da un valore di tipo string); Il costruttore Apply crea un'espressione data dall'applicazione di un'espressione ad un'altra.

Si scriva una funzione **is_free** (avente tipo `string -> lambda_expr -> bool`) che riceve come argomenti una stringa (che rappresenta il nome di una variabile / funzione) ed una Lambda-espressione, ritornando true se la variabile indicata appare come libera nell'espressione, false altrimenti (quindi, la funzione ritorna false se la variabile è legata o se non appare nell'espressione).

Come esempio, l'invocazione
`is_free "a" (Var "a")`
deve avere risultato true, l'invocazione
`is_free "b" (Var "a")`
deve avere risultato false, l'invocazione
`is_free "a" (Lambda ("a", Apply((Var "a"), Var "b")))`
deve avere risultato false, l'invocazione
`is_free "b" (Lambda ("a", Apply((Var "a"), Var "b")))`
deve avere risultato true e così via.

Soluzione

```
val rec is_free = fn s => fn Var v => s = v
                           | Lambda (v, e) => if (s = v) then false else is_free s e
                           | Apply (e1, e2) => (is_free s e1) orelse (is_free s e2);
```

Testing (Non fa parte della soluzione - utile per capire)

```
is_free "a" (Var "a");
is_free "b" (Var "a");
is_free "a" (Lambda ("a", Apply((Var "a"), Var "b")));
is_free "b" (Lambda ("a", Apply((Var "a"), Var "b")));
```

Turno 2 L2016

Testo

Basandosi sul tipo di dato espressione e la funzione **eval** definiti come segue:

```
local
    val rec eval = fn costante n => n
                   | somma (a1, a2) => (eval a1) + (eval a2)
                   | sottrazione (a1, a2) => (eval a1) - (eval a2)
                   | prodotto (a1, a2) => (eval a1) * (eval a2)
                   | divisione (a1, a2) => (eval a1) div (eval a2);
in
    val semplifica = fn costante n => costante(n)
                     | somma (a1, a2) => costante((eval a1) + (eval a2))
                     | sottrazione (a1, a2) => costante((eval a1) - (eval a2))
                     | prodotto (a1, a2) => costante((eval a1) * (eval a2))
                     | divisione (a1, a2) => costante((eval a1) div (eval a2))
end;
```

il tipo espressione può essere esteso come segue per supportare il concetto di variabile:

```
datatype espressione = costante of int
                     | variabile of string
                     | somma of espressione * espressione
                     | sottrazione of espressione * espressione
                     | prodotto of espressione * espressione
                     | divisione of espressione * espressione
                     | var of string * espressione * espressione;
```

Si riscriva la funzione **eval** per supportare i due nuovi costruttori *variabile e var*. Variabile x, con x di tipo stringa, é valutata al valore della variabile di nome x (per fare questo, eval deve cercare nell'ambiente un legame fra tale nome ed un valore). Var (x, e1, e2) é valutata al valore di e2 dopo aver assegnato ad x il valore di e1.
Per poter valutare correttamente variabile e var, **eval** deve quindi ricevere come argomento l'ambiente in cui valutare le variabili. Tale ambiente può essere rappresentato come una lista di coppie `(stringa, intero)` ed avrà quindi tipo `(string, int)list` .
La funzione **eval** deve quindi avere tipo `(string, int)list -> espressione -> int` .

Soluzione

```
local
    val rec cerca = fn s => fn [] => 0
                    | (s1, v)::l => if s1 = s then v else cerca s l
in
    val rec eval = fn env => fn costante n => n
                             | variabile s => cerca s env
                             | somma (a1, a2) => (eval env a1) + (eval env a2)
                             | sottrazione (a1, a2) => (eval env a1) - (eval env a2)
                             | prodotto (a1, a2) => (eval env a1) * (eval env a2)
                             | divisione (a1, a2) => (eval env a1) div (eval env a2)
                             | var (v, e1, e2) => eval ((v, eval env e1)::env) e2
end;
```

Esame agosto 2016

Testo

Si consideri una possibile implementazione degli insiemi di interi in standard ML, in cui un insieme di interi rappresentato da una funzione da int a bool:

```
type insiemediinteri = int -> bool;
```

La funzione applicata ad un numero intero ritorna true se il numero appartiene all'insieme, false altrimenti. L'insieme vuoto è quindi rappresentato da una funzione che ritorna sempre false:

```
val vuoto:insiemediinteri = fn n => false;
```

ed un intero può essere aggiunto ad un insieme tramite la funzione aggiungi:

```
val aggiungi = fn f:insiemediinteri => fn x:int => (fn n:int => if (n = x)
                                                                then true
                                                                else false):insiemediinteri;
```

É possibile verificare se un intero è contenuto in un insieme tramite la funzione contiene:

```
val contiene = fn f:insiemediinteri => fn n:int => f n;
```

Si implementi la funzione **intersezione**, avente tipo `insiemediinteri -> insiemediinteri -> insiemediinteri` , che dati due insiemi di interi ne calcola l'intersezione.

Soluzione

```
val intersezione = fn i1:insiemediinteri => fn i2:insiemediinteri => (fn n => ((contiene i1 n) andalso (contiene i2 n))):insiemediinteri;
```

Esame febbraio 2017

Testo

Si consideri una possibile implementazione degli insiemi di interi in standard ML, in cui un insieme di interi rappresentato da una funzione da int a bool:

```
type insiemediinteri = int -> bool;
```

La funzione applicata ad un numero intero ritorna true se il numero appartiene all'insieme, false altrimenti. L'insieme vuoto è quindi rappresentato da una funzione che ritorna sempre false:

```
val vuoto:insiemediinteri = fn n => false;
```

ed un intero può essere aggiunto ad un insieme tramite la funzione aggiungi:

```
val aggiungi = fn f:insiemediinteri => fn x:int => (fn n:int => if (n = x)
                                                                then true
                                                                else false):insiemediinteri;
```

É possibile verificare se un intero è contenuto in un insieme tramite la funzione contiene:

```
val contiene = fn f:insiemediinteri => fn n:int => f n;
```

Si implementi la funzione **unione**, avente tipo `insiemediinteri -> insiemediinteri -> insiemediinteri` che dati due insiemi di interi ne calcola l'unione.

Soluzione

```
val unione = fn i1:insiemediinteri => fn i2:insiemediinteri => (fn n => ((contiene i1 n) orelse (contiene i2 n))):insiemediinteri;
```

Esame giugno 2017

Turno 1 G2017

Testo

Si scriva una funzione **sommali** (avente tipo `int -> int list -> int`) che riceve come argomento un intero n ed una lista di interi l. La funzione **sommali** somma ad n gli elementi di l che hanno posizione pari (se la lista contiene meno di 2 elementi, sommali ritorna n).

Come esempio, l'invocazione

```
sommali 0 [1,2];
```

deve avere risultato 2;

```
sommali 1 [1,2,3];
```

deve avere risultato 3;

`sommali 2 [1,2,3,4];`
deve avere risultato 8.

Soluzione

```
fun sommali z [] = z
 |  sommali z (a::[]) = z
 |  sommali z (a::b::c) = b + (sommali z c);
```

altro modo extra usando fn -> stesso risultato

```
val rec sommali_diversa = fn z => fn [] => z
                                  | a::[] => z
                                  | a::b::c => b + (sommali z c);
```

Testing (Non fa parte della soluzione - utile per capire)

```
sommali 0 [1,2];
sommali 1 [1,2,3];
sommali 2 [1,2,3,4];

sommali_diversa 0 [1,2];
sommali_diversa 1 [1,2,3];
sommali_diversa 2 [1,2,3,4];
```

Turno 2 G2017

Testo

Si scriva una funzione **sommali** (avente tipo `int -> int list -> int`) che riceve come argomento un intero n ed una lista di interi l. La funzione **sommali** somma ad n gli elementi di l che hanno posizione multipla di 3 (se la lista contiene meno di 3 elementi, sommali ritorna n).

Come esempio, l'invocazione
`sommali 0 [1,2,3];`
deve avere risultato 3,
`sommali 1 [1,2,3];`
deve avere risultato 4: e
`sommali 2 [1,2,3,4,5,6];`
deve avere risultato 11.

Soluzione

```
fun sommali z [] = z
 |  sommali z (a::[]) = z
 |  sommali z (a::b::[]) = z
 |  sommali z (a::b::c::d) = c + (sommali z d);
```

altro modo extra usando fn -> stesso risultato

```
val rec sommali_diversa = fn z => fn [] => z
                                  | a::[] => z
                                  | a::b::[] => z
                                  | a::b::c::d => c + (sommali z d);
```

Testing (Non fa parte della soluzione - utile per capire)

```
sommali 0 [1,2,3];
sommali 1 [1,2,3];
sommali 2 [1,2,3,4,5,6];

sommali_diversa 0 [1,2,3];
sommali_diversa 1 [1,2,3];
sommali_diversa 2 [1,2,3,4,5,6];
```

Esame luglio 2017

Turno 1 L2017

Testo

Si consideri il tipo di dato `FOR = For of int * (int -> int);` i cui valori `For(n, f)` rappresentano funzioni che implementano un ciclo for come il seguente:

```
int ciclofor (int x) {
    for (int i = 0; i < n; i++) {
        x = f(x);
    }
}
```

Si scriva una funzione **eval** (avente tipo `FOR -> (int -> int)`) che riceve come argomento un valore di tipo FOR e ritorna una funzione da interi ad interi che implementa il ciclo indicato qui sopra (applica n - 1 volte la funzione f all'argomento).

Come esempio, `se val f = fn x => x * 2` , allora eval `(For(3, f))` ritornerà una funzione che dato un numero i ritorna i = 8:

Esempi esecuzione:

> val f = fn x => x * 2;
> `val f = fn: int -> int`
> eval (For(3, f));
> `val it = fn: int -> int`
> val g = eval (For(3, f));
> `val g = fn: int -> int`
> g 5;
> `val it = 40: int`

Soluzione

```
datatype FOR = For of int * (int -> int);
val rec eval = fn For (n, f) => fn x => if (n > 0)
                                        then
                                            eval (For (n - 1, f)) (f x)
                                        else
                                            x;
```

Testing (Non fa parte della soluzione - utile per capire)

```
val f = fn x => x * 2;
eval (For(3, f));
val g = eval (For(3, f));
g 5;
```

Turno 2 L2017

Testo

Si consideri il tipo di dato `FOR = For of int * (int -> int);` i cui valori `For(n, f)` rappresentano funzioni che implementano un ciclo for come il seguente:

```
int ciclofor (int x) {
    for (int i = 1; i < n; i++) {
        x = f(x);
    }
}
```

Si scriva una funzione **eval** (avente tipo `FOR -> (int -> int)`) che riceve come argomento un valore di tipo FOR e ritorna una funzione da interi ad interi che implementa il ciclo indicato qui sopra (applica n - 1 volte la funzione f all'argomento).

Come esempio, se `val f = fn x => x * 2` allora eval `(For(3, f))` ritornerà una funzione che dato un numero i ritorna i = 4:

Esempi esecuzione:

```
val f = fn x => x * 2;
 val f = fn: int -> int
eval (For(3, f));
 val it = fn: int -> int
val g = eval (For(3, f));
 val g = fn: int -> int
g 5;
 val it = 20: int
```

Soluzione

```
datatype FOR = For of int * (int -> int);
val rec eval = fn For (n, f) => fn x => if (n > 1)
                                        then
                                            eval (For (n - 1, f)) (f x)
                                        else
                                            x;
```

Testing (Non fa parte della soluzione - utile per capire)

```
val f = fn x => x * 2;
eval (For(3, f));
val g = eval (For(3, f));
g 5;
```

Esame settembre 2017

Testo

Si consideri il tipo di dato

```
datatype intonil = Nil | Int of int;
```

ed una possibile implementazione semplificata di ambiente (che considera solo valori interi) basata su di esso:

```
type ambiente = string -> intonil
```

In questa implementazione, un ambiente è rappresentato da una funzione che mappa nomi (valori di tipo string) in valori di tipo intonil (che rappresentano un intero o nessun valore). Tale funzione applicata ad un nome ritorna il valore intero ad esso associato oppure Nil. Usando questa convenzione, l'ambiente vuoto (in cui nessun nome è associato a valori) può essere definito come:

```
val ambientevuoto = fn _:string => Nil;
```

Basandosi su queste definizioni, si definisca una funzione **lega** con `tipo ambiente -> string -> int -> ambiente` . che a partire da un ambiente (primo argomento) genera un nuovo ambiente (valore di ritorno) uguale al primo argomento più un legame fra il nome e l'intero ricevuti come secondo e terzo argomento.

Esempio:
- `((lega ambientevuoto "a"1)"a")` deve ritornare Int 1;
- `((lega ambientevuoto "a"1)"boh")` deve ritornare Nil;
- `((lega (lega ambientevuoto "a"1)"boh"~1)"boh")` deve ritornare Int ~1;
- `((lega (lega ambientevuoto "a"1)"boh"~1)"mah")` deve ritornare Nil.

Soluzione

```
val lega = fn e:ambiente => fn nome => fn valore => (fn n => if (n = nome)
                                                             then
                                                                (Int valore)
                                                             else
                                                                (e n)):ambiente;
```

Testing (Non fa parte della soluzione - utile per capire)

```
((lega ambientevuoto "a"1)"a");
((lega ambientevuoto "a"1)"boh");
((lega (lega ambientevuoto "a"1)"boh"~1)"boh");
((lega (lega ambientevuoto "a"1)"boh"~1)"mah");
```

Esame gennaio 2018

Testo

Si consideri il tipo di dato:

```
datatype lambda_expr = Var of string
                     | Lambda of string * lambda_expr
                     | Apply of lambda_expr * lambda_expr;
```

che rappresenta un'espressione del Lambda-calcolo.
Il costruttore Var crea un'espressione costituita da un'unica funzione / variabile (il cui nome è un valore di tipo string); Il costruttore Lambda crea una Lambda-espressione a partire da un'altra espressione, legandone una variabile (indicata da un valore di tipo string);
Il costruttore Apply crea un'espressione data dall'applicazione di un'espressione ad un'altra.

Si scriva una funzione **is_bound** (avente tipo `string -> lambda_expr -> bool`) che riceve come argomenti una stringa (che rappresenta il nome di una variabile / funzione) ed una Lambda-espressione, ritornando true se la variabile indicata appare come legata nell'espressione, false altrimenti.

Soluzione

```
val rec is_bound = fn s => fn Var v => s = v
                            | Lambda (v, e) => if (s = v) then true else is_bound s e
                            | Apply (e1, e2) => (is_bound s e1) orelse (is_bound s e2);
```

Testing (Non fa parte della soluzione - utile per capire)

```
is_bound "a" (Var "a");
is_bound "b" (Var "a");
is_bound "a" (Lambda ("a", Apply((Var "a"), Var "b")));
is_bound "b" (Lambda ("a", Apply((Var "a"), Var "b")));
```

Esame giugno 2018

Testo

Si scriva una funzione **conta** (avente tipo `''a list -> int`) che riceve come argomento una lista di ''a l. La funzione **conta** ritorna il numero di elementi della lista senza considerare i duplicati.

Soluzione

```
val rec conta = fn [] => 0
                | a::b => if (List.exists ((fn y => a = y)) b)
                          then
                               (conta b)
                          else
                               1+(conta b);
```

Testing (Non fa parte della soluzione - utile per capire)

```
conta ["pera", "pera", "pera", "pera"];
conta ["red", "red", "green", "blue"];
conta ["red"];
conta [];
conta [1,2,4,5,6,0,1,4,5];
conta [true, false, true, false];
conta [#"A",#"a",#"B",#"b"];
```

Esame luglio 2018

Testo

Si consideri il tipo di dato `ITER = Iter of int * (int -> int);` i cui valori `ITER(n, f)` rappresentano funzioni che implementano un ciclo for come il seguente:

```
int ITER (int x) {
    for (int i = 1; i < n; i++) {
        x = f(x);
    }
}
```

Si scriva una funzione eval (avente tipo `Iter -> (int -> int)`) che riceve come argomento un valore di tipo Iter e ritorna una funzione da interi ad interi che implementa il ciclo indicato qui sopra (applica n - 1 volte la funzione f all'argomento).

Come esempio, se val `f = fn x => x * 2` , allora eval `(Iter(3, f))` ritornerà una funzione che dato un numero i ritorna i = 8:

ESEMPI ESECUZIONE:

> val f = fn x => x * 2;
> `val f = fn: int -> int`
> eval (Iter(3, f));
> `val it = fn: int -> int`
> val g = eval (Iter(3, f));
> `val g = fn: int -> int`
> g 5;
> `val it = 40: int`

Soluzione

```
datatype ITER = Iter of int * (int -> int);
val rec eval = fn Iter (n, f) => fn x => if (n > 0) then eval (Iter (n-1, f)) (f x) else x;
```

Testing (Non fa parte della soluzione - utile per capire)

```
val f = fn x => x * 2;
eval (Iter(3, f));
val g = eval (Iter(3, f));
g 5;
```

Esame agosto 2018

Testo

Scrivere una funzione **prod** che calcoli il prodotto di tutti gli interi tra 2 numeri m ed n. (con m >= m inclusi).

Usare poi questa funzione per scrivere una funzione Comb(n,k) la quale calcola il numero di combinazioni di n elementi presi k a k.
Definita dalla formula matematica: `C(n, k) = n!/(k! \* (n − k)!)`
che equivale anche a: `C(n, k) = (n − k + 1) · (n − k + 2)· · · n/1 · 2 · · · k`

Soluzione

```
fun prod(m, n) = if (n <= m) then m else n * prod(m, n-1);

fun comb(n, k) = (prod(n-k+1, n)) div (prod(1, k));
```

Testing (Non fa parte della soluzione - utile per capire)

```
comb(5,3);
comb(7,5);
comb(9,1);
```

Esame giugno 2019

Testo

Scrivere una funzione **f** di tipo `int list -> int list` che presa in input una lista trasformi ogni elemento "a" della lista nel seguente modo:

- se a >= 0 allora l'elemento "a" dovrà essere trasformato in a^2-1
- altrimenti l'elemento "a" dovrà essere trasformato in a^2+1

ESEMPIO
`f [~1,2,3,0,~5,6];`
dovrà dare in output:
`val it = [2, 3, 8, ~1, 26, 35]: int list`

Soluzione

```
fun f [] = []
 |  f (a::b) = if (a>=0) then ((a*a)-1)::(f b) else ((a*a)+1)::(f b);
```

Testing (Non fa parte della soluzione - utile per capire)

```
f [~1,2,3,0,~5,6];
f [0,~1];
```

Esame luglio 2019

> L'esame era molto simile a quello di giugno dovrebbe essere cambiata solo l'operazione da svolgere sugli elementi.
> Sfortunatamente non sono riuscito a reperire il testo ☹

Esame settembre 2019

Testo

Scrivere una funzione chiamata **f** che prende in input un file "text.txt" la funzione deve ritornare la lista dei caratteri senza spazi presenti nel file di testo.

Si assuma che il file sia presente nella stessa cartella.

Esempio:
Dato il file "text.txt" che contiene:
ab e ad c
Eseguendo il comando `use "esercizio.sml"` ;
dovrà essere prodotto il seguente risultato:
`val it = [#"a", #"b", #"e", #"a", #"d", #"c"]: char list`

Soluzione

```
fun f filename =
    let
        val file = TextIO.openIn filename
        val str = TextIO.inputAll file
    in
        TextIO.closeIn file;
        List.filter (fn c => c <> #" " andalso c <> #"\t" andalso c <> #"\n") (explode str)
    end;
f "text.txt";
```

Esame gennaio 2020

Testo

Scrivere una funzione chiamata **f** che prende in input un file "text2.txt" la funzione deve ritornare i caratteri in posizione pari presenti nel file txt. Il file txt conterrà un numero di caratteri pari maggiore di zero.

Si assuma che il file sia presente nella stessa cartella.

ESEMPIO di esecuzione
Dato il file "text2.txt" che contiene:
abcdef
Eseguendo il comando `use "esercizio.sml";`
dovrà essere prodotto il seguente risultato:
`val it = ["b","d","f"]: string list`

Soluzione

```
fun f filename =
    let
        val file = TextIO.openIn filename
        val str = TextIO.inputAll file
        val rec elem_pari = fn [] => []
                            | a::[] => []
                            | a::b::c => Char.toString(b)::(elem_pari c)
    in
        TextIO.closeIn file;
        elem_pari (explode str)
    end;
f "text2.txt";
```

Esame febbraio 2020

Testo

Scrivere una funzione **f** che data una lista di interi restituisca true se essa è ordinata in ordine crescente false altrimenti

Soluzione

```
fun f [] = true
 |  f (a::[]) = true
 |  f (a::b::l) = if a > b then false else f(b::l);
```

Oppure altra **soluzione valida** - presa dalle slide

```
val rec f1 = fn l =>
        if List.null l then true
        else if List.null (tl l) then true
        else (hd l <= hd (tl l)) andalso f1 (tl l);
```

Testing (Non fa parte della soluzione - utile per capire)

```
f [1,2,2,3,4];
f[3,2,1];
f[4,12,3,44,0];

f1 [1,2,2,3,4];
f1[3,2,1];
f1[4,12,3,44,0];
```