Esercizi di Programmazione Prolog

Marco Comini

29 febbraio 2008

Nel seguito useremo il termine PROLOG in contrasto a PROLOG PURO per distinguere Prolog con builtins (come `<` e `is`) da quello senza.

Inoltre quando si parla di funzioni ad n argomenti ed m valori si intende un predicato ad $n+m$ argomenti, dove i primi n argomenti sono gli input e gli altri m gli output.

Si scrivano i programmi con variabili anonime ove possibile.

1 Basics

Si ricordi che le espressioni aritmetiche built-in si usano con il predicato infisso `is/2`. Numeriamo gli elementi delle liste a partire da 1 (quindi in $[x_1, \ldots, x_n]$ l'elemento di posizione 1 è x_1).

1. Scrivere un predicato in PROLOG per la relazione rappresentante la funzione fattoriale (attenzione a non generare soluzioni "extra").

2. Si assuma di aver scritto un predicato `generateList/1` che costruisce una lista. Utilizzando solo `generateList/1` e `append/3` scrivere un goal per determinare se la lista prodotta da `generateList/1` contiene (in ordine) i numeri 6 e 2.

   Si provi la correttezza della soluzione con il seguente programma

   ```
   generateList([_,1,_]).
   generateList([_,6,_,1,2,_,0]).
   ```

   e successivamente con

   ```
   generateList(_).
   ```

3. Scrivere un programma in PROLOG PURO per la seguente relazione: `oddinlist(Xs,Ys)` se e solo se `Ys` è la lista che si ottiene da `Xs` rimuovendo gli elementi di posizione pari. Ad esempio

   ```
   oddinlist([a,b,c,d,e,f],[a,c,e])
   ```

   è vero. Il programma deve essere in grado di rispondere a queries del tipo `oddinlist(Xs,[a,b])`.

4. Scrivere un programma in PROLOG per la seguente relazione: `sumofevensublist(Xs,N)` se e solo se `N` è la somma degli elementi in posizione pari della lista `Xs`. Si assuma di chiamare `sumofevensublist/2` con il primo argomento sempre istanziato ad una lista con interi nelle posizioni pari.

5. Utilizzando solo i predicati dei punti precedenti scrivere una query per calcolare la somma degli elementi di posizione congrua a 3 modulo 4 (i.e. 3,7,11, ...) della lista dei primi 7 numeri naturali.

6. Scrivere in PROLOG puro un predicato che determina se una lista di elementi è palindroma.

7. Scrivere un programma in PROLOG per implementare il QuickSort (assumendo di chiamarlo con primo argomento una lista di numeri interi).

8. Scrivere un programma in PROLOG per la seguente relazione: `min2Odd(Xs,M,N)` se e solo se `M` e `N` sono i 2 minori elementi dispari di Xs (se esistono). Ad esempio

   ```
   min2Odd([2,3,4,6,8,7,5],3,5)
   ```

è vero.

9. Scrivere un predicato `sumPairsList/2` in PROLOG per costruire, a partire da una lista di numeri interi, una lista di coppie in cui

   (a) il primo elemento di ogni coppia è uguale all'elemento di corrispondente posizione nella lista originale e

   (b) il secondo elemento di ogni coppia è uguale alla somma di tutti gli elementi antecedenti della lista originale.

   Ad esempio

   ```
   sumPairsList([1,1,1,9],[(1,0),(1,1),(1,2),(9,3)])
   ```

   è vero.

   Si assuma di chiamare `sumPairsList/2` con il primo argomento sempre istanziato ad una lista di interi. Si utilizzi per le coppie il costruttore infisso `(X,Y)`. Si suggerisce di ispirarsi al metodo dell'accumulatore.

10. Scrivere un predicato `fiblist/2` in PROLOG per costruire, a partire da un numero intero n, la lista dei primi n numeri della successione di Fibonacci. (Si consiglia di costruire prima la lista a rovescio e poi girarla con `reverse/2`)

11. Si scriva un programma che data una lista ne restituisce una permutazione (tutte non-deterministicamente).

12. Si scriva un programma che data una lista di interi $[1, \ldots, n]$ restituisce un permutazione $[p_1, \ldots, p_n]$ in modo che, su una scacchiera di $n \times n$, n regine messe in posizione $(1, p_1), \ldots, (n, p_n)$ non si mangino fra loro.

2 Matrici

Si implementino le matrici come liste di liste per righe.

1. Si scriva una funzione parziale `matrix_dim/3` che data una matrice ne calcola le dimensioni, se la matrice è ben formata, altrimenti fallisce. Si può assumere di avere in ingresso una lista di liste, ma poi si provi a vedere se funziona "a rovescio" (con output ground e input variabile libera). Poi si scriva una funzione `gen_matrix/3` che date le dimensioni generi una matrice di variabili libere.

2. Si scriva un predicato `lowertriangular/1` che determina se una matrice (quadrata) è triangolare inferiore.

3. Una matrice quadrata M di ordine n si dice *convergente* con raggio r se il modulo della somma degli elementi di ogni riga, escluso quello sulla diagonale, è inferiore a r.

   Si scriva un predicato `convergent(m,r)` che determina se una matrice (quadrata) `m` è convergente con raggio `r`.

4. Si scriva un programma PROLOG che data una matrice di dimensioni $m \times n$ in input restituisce in output la corrispondente matrice trasposta (di dimensioni $n \times m$).

   Si faccia una prima versione assumendo di avere in input una variabile non libera (una matrice, ma non necessariamente ground) e poi si faccia la versione che accetta anche variabili libere.

5. Si scriva un predicato `isSymmetric` che, data una matrice quadrata, determina se è simmetrica. Si faccia attenzione a garantire il funzionamento quando lo si chiama con una variabile libera.

6. Si scriva un programma che date due matrici di dimensioni $n \times k$ e $k \times m$ in input restituisce in output la corrispondente matrice prodotto (di dimensioni $n \times m$).

3 Alberi Binari

Gli alberi binari in Prolog si implementano semplicemente scrivendo termini con un costruttore di arità 3 per i nodi e di arità 0 per l'albero vuoto. Solitamente si usano `void/0` e `node/3`, in particolare `node(key,left-son,right-son)`. Ad esempio `node(6,node(3,void,void),void)`.

Si ricorda che un BST (Albero Binario di Ricerca) è un albero binario con la seguente proprietà: i valori contenuti nel sottoalbero di sinistra sono tutti minori (o uguali) a quelli della radice e i valori del sottoalbero di destra tutti maggiori.

1. Scrivere un predicato `treesum/2` in PROLOG per calcolare, a partire da un albero binario di numeri interi, la somma dei valori presenti nell'albero. Si assuma di chiamare `treesum/2` con il primo argomento sempre istanziato ad una albero binario di interi.

2. Scrivere un predicato `sumEvenOfTree/2` in PROLOG per calcolare, a partire da un albero binario di numeri interi, la somma dei valori pari presenti nell'albero. Si assuma di chiamare `sumEvenOfTree/2` con il primo argomento sempre istanziato ad una albero binario di interi.

3. Scrivere un predicato `bstMember/2` per determinare se un valore è presente in un BST. Se un valore è molteplicemente presente si vuole avere tutte le soluzioni.

4. Scrivere un predicato `bstInsert/3` per inserire un valore in un BST.

5. Scrivere un predicato `bst2list/2` che determina la lista ordinata dei valori di un BST. Ci si assicuri di scrivere un predicato a complessità lineare.

6. Usando i predicati precedenti si costruisca un predicato di sorting per liste.

7. Scrivere in PROLOG un predicato `samesums/1` che presa una lista di alberi $[t_1, \ldots, t_n]$ determina se le somme $s_1, \ldots, s_n$ dei valori degli elementi di ogni t_i sono tutte uguali fra loro. Quindi, per esempio,

   ```
   samesums([node(4,void,node(3,void,void)),
             node(3,node(4,void,void),void)]  )
   ```

   è vero.

8. Scrivere in PROLOG un predicato `growingsums/1` che presa una lista di alberi $[t_1, \ldots, t_n]$ determina se le somme $s_1, \ldots, s_n$ dei valori degli elementi di ogni t_i sono in ordine strettamente crescente.

9. Scrivere un predicato `almostBalanced/1` in PROLOG per determinare se un albero binario ha la seguente proprietà: per ogni nodo le altezze dei figli destro e sinistro differiscono al massimo di 1. L'altezza è la lunghezza del massimo cammino radice-foglia. Si ricordi che si possono usare i funzionali abs e max.

10. Scrivere un predicato `annotate/2` in PROLOG che preso un BST (Albero Binario di Ricerca) costruisca un nuovo BST che in ogni nodo contenga, al posto del valore originale, una coppia composta dal medesimo valore e dalla altezza del nodo stesso (ie. $1 + max(depth(sx), depth(dx))$. Ad esempio

    ```
    annotate(node(4,void,node(3,void,void)),
          node((4,2),void,node((3,1),void,void)))
    ```

    è vero.

    [Con una opportuna scelta dell'ordine di ricorsione si può fare in tempo lineare]

11. Si consideri un BST dove per i nodi dell'albero si usi un costruttore `node/4` con un argomento extra in cui viene mantenuta l'altezza del nodo stesso.

    Si scriva una funzione `wbstinsert/3` che inserisce un nuovo valore in tale albero (in modo che i valori altezza del risultato siano corretti).

12. Sia $\max_n t$ il più grande dei valori strettamente minori di n di un BST di interi t (se ne esistono), si scriva una funzione `boundedMaximum/3` che dato un numero n e una lista l di BST determina la lista dei $\max_n t$ che esistono, al variare di t in l. Si consiglia di usare il built-in `(`G`->`G_1`;`G_2`)` (che a seconda del successo di G esegue G_1 o G_2).

13. Si scriva una funzione `diameter/2` che dato un BST ne determina il diametro. Il diametro di un BST è la lunghezza del massimo cammino fra due nodi, indipendentemente dall'orientamento degli archi.

14. Scrivere un predicato PROLOG `treepath/2` che determina percorsi all'interno di un albero binario. Quindi, a titolo di esempio,

    ```
    treepath(node(1,node(2,node(3,void,void),void),node(4,void,void)),Xs)
    ```

    restituisce 2 soluzioni per `Xs`: `[1,2,3]` e `[1,4]`. Non si escluda `treepath(void,[])`.

    Si faccia attenzione a *non generare la stessa soluzione più volte.*

15. Si costruiscano degli alberi con le costanti `void/0`, `bal/3`, `left/3` e `right/3`. L'idea è di usare left, right o bal a seconda che il sottoalbero sinistro sia più profondo del destro, meno o uguale. Un albero è detto AVL se la differenza fra le altezze dei sottoalberi destro e sinistro di un qualunque nodo è al massimo 1. Si scriva un predicato PROLOG `isAVL/1` che dato un albero determina se è AVL e se i costruttori dei nodi sono consistenti con lo (s)bilanciamento.

    A titolo di esempio,

    ```
    isAVL( left(3, right(2,void,bal(5,void,void)), bal(7,void,void) ) )
    ```

    è vero.

16. Si scriva un predicato PROLOG `diff2next/2` che, dato un albero binario di ricerca, costruisce un albero binario di ricerca (annotato) di coppie dove il primo elemento di ogni coppia è l'elemento dell'albero originale mentre il secondo elemento è la differenza rispetto al valore successivo (secondo l'ordinamento dei valori contenuti). Si utilizzi la costante `infty/0` per annotare il nodo di valore massimo. A titolo di esempio,

    ```
    diff2next(node(4,void,node(7,node(5,void,void),void)),T)
    ```

    restituisce la soluzione

    ```
    T = node((4,1),void,node((7,infty),node((5,2),void,void),void)).
    ```

    Si deve passare su ogni nodo dell'albero **una volta sola**.

17. Si scriva un predicato PROLOG `bst2stream/2` che, dato un albero binario di ricerca t, i cui elementi (in ordine) siano $x_1, \ldots, x_n$ costruisce il termine `stream`$(x_1, [x_2 - x_1, \ldots, x_n - x_{n-1}])$. A titolo di esempio,

    ```
    bst2stream(bst(5,bst(2,void,void), bst(6,void,void)),Xs)
    ```

    restituisce la soluzione `Xs = stream(2,[3,1])`.

18. Si scriva un predicato PROLOG `zap2min/2` che, dato un albero binario di ricerca, costruisce un albero binario (non necessariamente di ricerca) con la stessa forma contenente in ogni nodo il valore minimo dell'albero originale.

    Si deve passare su ogni nodo dell'albero **una volta sola**, si sfrutti la potenza dell'unificazione.

    Se serve si assuma di poter avere una costante distinta dai possibili valori dell'albero.

19. Si scriva una funzione `limitedVisit` che dato un BST e due valori x, y costruisce la lista (ordinata) degli elementi dell'albero compresi nell'intervallo di valori da x a y.

20. Si scriva un predicato `isBST/1` che dato un albero verifica se i valori in esso contenuti soddisfano la proprietà strutturale dei Binary Search Trees.

21. Si scriva un predicato `isRBT` che dato un Red Black Tree, dove per i nodi dell'albero si usi un costruttore `node/4` con un argomento colore `black/0` o `red/0`, determina se è ben formato, cioè se

    - ogni nodo contiene un valore non minore dei valori del suo sottoalbero sinistro e minore dei valori del sottoalbero destro;
    - tutti i cammini dalla radice a una foglia hanno lo stesso numero di nodi Black;
    - i nodi Red devono avere genitore Black;
    - la radice è Black.

22. Scrivere un predicato PROLOG `tree2bsf(T,Xs)` che, dato un albero binario, determina la lista degli elementi che si ottengono con una visita a livelli.

    Si cerchi di scrivere un programma che riesce a dare sia soluzioni "dirette" (`T` instanziata e `Xs` libera) che "a rovescio" (`Xs` instanziata e `T` libera).

4 CLP(FD) e CLP(R)

Per gli esercizi su CLP(FD) conviene usare gprolog.

1. Si scriva un programma CLP(FD) che data una lista di valori ne calcola la somma ma solo quando tutti gli elementi della lista sono tutti diversi fra loro.

2. Si scriva un programma CLP(FD), e poi la sua variante CLP(R), che data una matrice di dimensioni $m \times n$, un vettore di dimensione n restituisce in output il prodotto (di dimensioni $m \times 1$) ma solo quando tutti gli elementi del prodotto stesso sono tutti diversi fra loro.

3. Si scriva un programma CLP(FD), e poi la sua variante CLP(R) che data una matrice di dimensioni $m \times n$, un vettore di dimensione n e una costante max restituisce in output il prodotto (di dimensioni $m \times 1$) ma solo quando tutti gli elementi del prodotto stesso non superano il valore max.

4. Si scriva un programma CLP(FD) che dato n restituisce una lista di posizioni $[p_1, \ldots, p_n]$ in modo che, su una scacchiera di $n \times n$, n regine messe in posizione $(1, p_1), \ldots, (n, p_n)$ non si mangino fra loro. Si provi ad apprezzare la differenza di velocità rispetto alla versione LP.