# Esercizi Svolti 8: Relazione di un Progetto d'Esame

Marco Bernardo

Università degli Studi di Urbino Carlo Bo
Insegnamento di Programmazione Logica e Funzionale

## 1 Specifica del Problema

Scrivere un programma Haskell e un programma Prolog che acquisiscono da tastiera due insiemi finiti di numeri interi e poi stampano su schermo il risultato di unione, intersezione, differenza, differenza simmetrica e prodotto cartesiano dei due insiemi.

*[Confrontare lo svolgimento delle fasi successive con quanto riportato nelle prossime pagine.]*

# 2 Analisi del Problema

## 2.1 Dati di Ingresso del Problema

Per ciascuna delle cinque operazioni, i dati di ingresso del problema sono rappresentati da due insiemi finiti di numeri interi.

*[Nota metodologica 1: non è appropriato scrivere "... dall'acquisizione di due insiemi finiti di numeri interi ..." perché l'acquisizione è un'operazione, non un dato.]*

*[Nota metodologica 2: è sbagliato scrivere "... dalla cardinalità e dagli elementi di due insiemi finiti di numeri interi ..." perché, non essendo richiesto dalla specifica del problema, acquisire la cardinalità di ciascun insieme non può che essere una scelta di progetto, quindi non riguarda l'analisi del problema.]*

## 2.2 Dati di Uscita del Problema

L'unico dato di uscita del problema è rappresentato da un insieme finito di numeri interi per le prime quattro operazioni e da un insieme finito di coppie ordinate di numeri interi per la quinta operazione.

*[Nota metodologica 3: non è appropriato scrivere "... dalla stampa di un insieme finito di ..." perché la stampa è un'operazione, non un dato.]*

## 2.3 Relazioni Intercorrenti tra i Dati del Problema

Dati due insiemi $A$ e $B$ contenuti in un universo $U$, le operazioni in questione sono definite come segue:

- Unione: $A \cup B = \{u \in U \mid u \in A \text{ o } u \in B\}$.
- Intersezione: $A \cap B = \{u \in U \mid u \in A \text{ e } u \in B\}$.
- Differenza: $A \setminus B = \{u \in U \mid u \in A \text{ e } u \notin B\}$.
- Differenza simmetrica: $A \Delta B = \{u \in U \mid u \in A \text{ e } u \notin B, \text{ oppure } u \in B \text{ e } u \notin A\}$.
- Prodotto cartesiano: $A \times B = \{(a, b) \mid a \in A \text{ e } b \in B\}$.

Queste operazioni godono di numerose proprietà, tra cui si evidenziano le seguenti in quanto utili ai fini della risoluzione del problema:

- $A \cup \emptyset = \emptyset \cup A = A$.
- $A \cap \emptyset = \emptyset \cap A = \emptyset$.
- $A \setminus \emptyset = A$, $\emptyset \setminus A = \emptyset$.
- $A \Delta B = (A \setminus B) \cup (B \setminus A) = (A \cup B) \setminus (A \cap B)$.
- $A \times \emptyset = \emptyset \times A = \emptyset$.

*[Nota metodologica 4: oltre alle relazioni intercorrenti tra i dati di ingresso e i dati di uscita, potrebbero sussistere relazioni all'interno dei dati di ingresso così come relazioni all'interno dei dati di uscita, che vanno esse pure evidenziate in questa sottosezione. Per esempio, se la specifica del problema chiede di acquisire da tastiera un insieme finito e una relazione binaria su di esso, i quali sono entrambi dati di ingresso, occorre indicare che la relazione binaria è un sottoinsieme del prodotto cartesiano dell'insieme per se stesso e che la cardinalità della relazione non può pertanto superare il quadrato della cardinalità dell'insieme.]*

# 3 Progettazione dell'Algoritmo

## 3.1 Scelte di Progetto

Gli insiemi finiti di numeri interi si prestano in modo naturale a essere rappresentati mediante strutture dati lineari. Queste verranno allocate dinamicamente in quanto non si conosce a priori la cardinalità degli insiemi che dovranno contenere, in maniera da non imporre limitazioni rispetto alla specifica del problema.

Eventuali elementi duplicati verranno tollerati in fase di acquisizione, ma saranno poi eliminati in fase di memorizzazione perché la molteplicità degli elementi è irrilevante in un insieme.

## 3.2 Passi dell'Algoritmo

I passi dell'algoritmo per risolvere il problema sono i seguenti:

- Acquisire il primo insieme finito di numeri interi.
- Acquisire il secondo insieme finito di numeri interi.
- Calcolare e stampare l'unione dei due insiemi:
  - Caso base: se il secondo insieme è vuoto, l'unione è uguale al primo insieme.
  - Caso generale: se il secondo insieme non è vuoto, si verifica se il primo elemento del secondo insieme appartiene o meno al primo insieme e lo si aggiunge all'unione in caso di non appartenenza, poi si procede ricorsivamente col resto del secondo insieme.
- Calcolare e stampare l'intersezione dei due insiemi:
  - Caso base: se il primo insieme è vuoto, l'intersezione è vuota.
  - Caso generale: se il primo insieme non è vuoto, si verifica se il primo elemento del primo insieme appartiene o meno al secondo insieme e lo si aggiunge all'intersezione in caso di appartenenza, poi si procede ricorsivamente col resto del primo insieme.
- Calcolare e stampare la differenza tra i due insiemi:
  - Caso base: se il secondo insieme è vuoto, la differenza è uguale al primo insieme.
  - Caso generale: se il secondo insieme non è vuoto, si elimina il primo elemento del secondo insieme dal primo insieme, poi si procede ricorsivamente col resto del secondo insieme.
- Calcolare e stampare la differenza simmetrica tra i due insiemi:
  - Calcolare la differenza tra il primo insieme e il secondo insieme.
  - Calcolare la differenza tra il secondo insieme e il primo insieme.
  - Calcolare l'unione delle due differenze precedentemente calcolate.
- Calcolare e stampare il prodotto cartesiano dei due insiemi:
  - Caso base: se il primo insieme è vuoto, il prodotto cartesiano è vuoto.
  - Caso generale: se il primo insieme non è vuoto, si aggiungono al prodotto cartesiano tutte le coppie ordinate formate dal primo elemento del primo insieme e da un elemento del secondo insieme, poi si procede ricorsivamente col resto del primo insieme.

# 4 Implementazione dell'Algoritmo

File sorgente `operazioni_insiemi.hs`:

```
{- Programma Haskell per calcolare operazioni insiemistiche. -}

import Data.List  -- necessario per usare nub, che elimina elementi duplicati da una lista

main :: IO ()
main = do putStrLn "Inserire il primo insieme di numeri interi racchiuso tra parentesi quadre:"
          i1 <- getLine
          let as = nub (read i1 :: [Int])
          putStrLn ">> Primo insieme privo di eventuali elementi duplicati:"
          putStrLn $ show as
          putStrLn "Inserire il secondo insieme di numeri interi racchiuso tra parentesi quadre:"
          i2 <- getLine
          let bs = nub (read i2 :: [Int])
          putStrLn ">> Secondo insieme privo di eventuali elementi duplicati:"
          putStrLn $ show bs
          putStrLn "Unione:"
          putStrLn $ show (unione as bs)
          putStrLn "Intersezione:"
          putStrLn $ show (intersezione as bs)
          putStrLn "Differenza:"
          putStrLn $ show (differenza as bs)
          putStrLn "Differenza simmetrica:"
          putStrLn $ show (diff_simm as bs)
          putStrLn "Prodotto cartesiano:"
          putStrLn $ show (prod_cart as bs)

{- La funzione unione calcola l'unione di due insiemi:
   - il primo argomento è il primo dei due insiemi;
   - il secondo argomento è il secondo dei due insiemi.
   L'uso di reverse consente di stampare in ordine gli elementi del secondo insieme
   che fanno parte dell'insieme unione. -}

unione :: [Int] -> [Int] -> [Int]
unione as bs = unione' as (reverse bs)
  where
    unione' as []                   = as
    unione' as (b : bs) | elem b as = unione' as bs
                        | otherwise = unione' as bs ++ [b]

{- La funzione intersezione calcola l'intersezione di due insiemi:
   - il primo argomento è il primo dei due insiemi;
   - il secondo argomento è il secondo dei due insiemi. -}

intersezione :: [Int] -> [Int] -> [Int]
intersezione [] _                    = []
intersezione (a : as) bs | elem a bs = a : intersezione as bs
                         | otherwise = intersezione as bs
```

```
{- La funzione differenza calcola la differenza tra due insiemi:
   - il primo argomento è il primo dei due insiemi;
   - il secondo argomento è il secondo dei due insiemi. -}

differenza :: [Int] -> [Int] -> [Int]
differenza as []       = as
differenza as (b : bs) = differenza (filter (/= b) as) bs

{- La funzione diff_simm calcola la differenza simmetrica tra due insiemi:
   - il primo argomento è il primo dei due insiemi;
   - il secondo argomento è il secondo dei due insiemi. -}

diff_simm :: [Int] -> [Int] -> [Int]
diff_simm as bs = unione (differenza as bs) (differenza bs as)

{- La funzione prod_cart calcola il prodotto cartesiano di due insiemi:
   - il primo argomento è il primo dei due insiemi;
   - il secondo argomento è il secondo dei due insiemi. -}

prod_cart :: [Int] -> [Int] -> [(Int, Int)]
prod_cart as bs = [(a, b) | a <- as, b <- bs]
```

File sorgente `operazioni_insiemi.pl`:

```
/* Programma Prolog per calcolare operazioni insiemistiche. */

main :- write('Inserire il primo insieme di numeri interi racchiuso tra parentesi quadre'),
        write(' e terminato da punto:'), nl,
        read(I1), interi_e_diversi(I1, AS),
        write('>> Primo insieme privo di eventuali elementi duplicati:'), nl,
        write(AS), nl,
        write('Inserire il secondo insieme di numeri interi racchiuso tra parentesi quadre'),
        write(' e terminato da punto:'), nl,
        read(I2), interi_e_diversi(I2, BS),
        write('>> Secondo insieme privo di eventuali elementi duplicati:'), nl,
        write(BS), nl,
        write('Unione:'), nl,
        unione(AS, BS, CS1), write(CS1), nl,
        write('Intersezione:'), nl,
        intersezione(AS, BS, CS2), write(CS2), nl,
        write('Differenza:'), nl,
        differenza(AS, BS, CS3), write(CS3), nl,
        write('Differenza simmetrica:'), nl,
        diff_simm(AS, BS, CS4), write(CS4), nl,
        write('Prodotto cartesiano:'), nl,
        prod_cart(AS, BS, CS5), write(CS5), nl.

interi_e_diversi([], []).
interi_e_diversi([A | AS], [A | AS1]) :- integer(A), delete(AS, A, AS2), interi_e_diversi(AS2, AS1).

/* Il predicato unione calcola l'unione di due insiemi:
   - il primo argomento è il primo dei due insiemi;
   - il secondo argomento è il secondo dei due insiemi;
   - il terzo argomento è l'insieme unione.
   L'uso di reverse consente di stampare in ordine gli elementi del secondo insieme
   che fanno parte dell'insieme unione. */

unione(AS, BS, CS) :- reverse(BS, BS1), unione1(AS, BS1, CS).

unione1(AS, [], AS).
unione1(AS, [B | BS], CS) :- member(B, AS), !, unione1(AS, BS, CS).
unione1(AS, [B | BS], CS) :- unione1(AS, BS, CS1), append(CS1, [B], CS).

/* Il predicato intersezione calcola l'intersezione di due insiemi:
   - il primo argomento è il primo dei due insiemi;
   - il secondo argomento è il secondo dei due insiemi;
   - il terzo argomento è l'insieme intersezione. */

intersezione([], _, []).
intersezione([A | AS], BS, CS) :- member(A, BS), !, intersezione(AS, BS, CS1), append([A], CS1, CS).
intersezione([_ | AS], BS, CS) :- intersezione(AS, BS, CS).
```

```
/* Il predicato differenza calcola la differenza tra due insiemi:
   - il primo argomento è il primo dei due insiemi;
   - il secondo argomento è il secondo dei due insiemi;
   - il terzo argomento è l'insieme differenza. */

differenza(AS, [], AS).
differenza(AS, [B | BS], CS) :- delete(AS, B, AS1), differenza(AS1, BS, CS).

/* Il predicato diff_simm calcola la differenza simmetrica tra due insiemi:
   - il primo argomento è il primo dei due insiemi;
   - il secondo argomento è il secondo dei due insiemi;
   - il terzo argomento è l'insieme differenza simmetrica. */

diff_simm(AS, BS, CS) :- differenza(AS, BS, CS1), differenza(BS, AS, CS2), unione(CS1, CS2, CS).

/* Il predicato prod_cart calcola il prodotto cartesiano di due insiemi:
   - il primo argomento è il primo dei due insiemi;
   - il secondo argomento è il secondo dei due insiemi;
   - il terzo argomento è l'insieme prodotto cartesiano. */

prod_cart([], _, []).
prod_cart([A | AS], BS, CS) :- gen_coppie(A, BS, CS1), prod_cart(AS, BS, CS2), append(CS1, CS2, CS).

gen_coppie(_, [], []).
gen_coppie(A, [B | BS], CS) :- gen_coppie(A, BS, CS1), append([(A, B)], CS1, CS).
```

# 5 Testing del Programma

## Test Haskell 1

Primo insieme: `[]`
Secondo insieme: `[]`

Unione: `[]`
Intersezione: `[]`
Differenza: `[]`
Differenza simmetrica: `[]`
Prodotto cartesiano: `[]`

## Test Haskell 2

Primo insieme: `[1]`
Secondo insieme: `[]`

Unione: `[1]`
Intersezione: `[]`
Differenza: `[1]`
Differenza simmetrica: `[1]`
Prodotto cartesiano: `[]`

## Test Haskell 3

Primo insieme: `[]`
Secondo insieme: `[1]`

Unione: `[1]`
Intersezione: `[]`
Differenza: `[]`
Differenza simmetrica: `[1]`
Prodotto cartesiano: `[]`

## Test Haskell 4

Primo insieme: `[1]`
Secondo insieme: `[1]`

Unione: `[1]`
Intersezione: `[1]`
Differenza: `[]`
Differenza simmetrica: `[]`
Prodotto cartesiano: `[(1, 1)]`

## Test Haskell 5

Primo insieme: `[1]`
Secondo insieme: `[2]`

Unione: `[1, 2]`
Intersezione: `[]`
Differenza: `[1]`
Differenza simmetrica: `[1, 2]`
Prodotto cartesiano: `[(1, 2)]`

### Test Haskell 6

Primo insieme: `[1, 2, 3]`
Secondo insieme: `[3, 4, 5]`

Unione: `[1, 2, 3, 4, 5]`
Intersezione: `[3]`
Differenza: `[1, 2]`
Differenza simmetrica: `[1, 2, 4, 5]`
Prodotto cartesiano: `[(1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 3), (3, 4), (3, 5)]`

### Test Haskell 7

Primo insieme: `[1, 2, 3]`
Secondo insieme: `[3, 2, 1]`

Unione: `[1, 2, 3]`
Intersezione: `[1, 2, 3]`
Differenza: `[]`
Differenza simmetrica: `[]`
Prodotto cartesiano: `[(1, 3), (1, 2), (1, 1), (2, 3), (2, 2), (2, 1), (3, 3), (3, 2), (3, 1)]`

### Test Haskell 8

Primo insieme: `[1, 2, 3]`
Secondo insieme: `[6, 5, 4]`

Unione: `[1, 2, 3, 6, 5, 4]`
Intersezione: `[]`
Differenza: `[1, 2, 3]`
Differenza simmetrica: `[1, 2, 3, 6, 5, 4]`
Prodotto cartesiano: `[(1, 6), (1, 5), (1, 4), (2, 6), (2, 5), (2, 4), (3, 6), (3, 5), (3, 4)]`

### Test Haskell 9

Primo insieme: `[1, 2, 3]`
Secondo insieme: `[2, 3, 0]`

Unione: `[1, 2, 3, 0]`
Intersezione: `[2, 3]`
Differenza: `[1]`
Differenza simmetrica: `[1, 0]`
Prodotto cartesiano: `[(1, 2), (1, 3), (1, 0), (2, 2), (2, 3), (2, 0), (3, 2), (3, 3), (3, 0)]`

### Test Haskell 10

Primo insieme: `[1, 2, 3]`
Secondo insieme: `[]`

Unione: `[1, 2, 3]`
Intersezione: `[]`
Differenza: `[1, 2, 3]`
Differenza simmetrica: `[1, 2, 3]`
Prodotto cartesiano: `[]`

### Test Prolog 1

Primo insieme: `[]`
Secondo insieme: `[]`

Unione: `[]`
Intersezione: `[]`
Differenza: `[]`
Differenza simmetrica: `[]`
Prodotto cartesiano: `[]`

### Test Prolog 2

Primo insieme: `[1]`
Secondo insieme: `[]`

Unione: `[1]`
Intersezione: `[]`
Differenza: `[1]`
Differenza simmetrica: `[1]`
Prodotto cartesiano: `[]`

### Test Prolog 3

Primo insieme: `[]`
Secondo insieme: `[1]`

Unione: `[1]`
Intersezione: `[]`
Differenza: `[]`
Differenza simmetrica: `[1]`
Prodotto cartesiano: `[]`

### Test Prolog 4

Primo insieme: `[1]`
Secondo insieme: `[1]`

Unione: `[1]`
Intersezione: `[1]`
Differenza: `[]`
Differenza simmetrica: `[]`
Prodotto cartesiano: `[(1, 1)]`

### Test Prolog 5

Primo insieme: `[1]`
Secondo insieme: `[2]`

Unione: `[1, 2]`
Intersezione: `[]`
Differenza: `[1]`
Differenza simmetrica: `[1, 2]`
Prodotto cartesiano: `[(1, 2)]`

### Test Prolog 6

Primo insieme: `[1, 2, 3]`
Secondo insieme: `[3, 4, 5]`

Unione: `[1, 2, 3, 4, 5]`
Intersezione: `[3]`
Differenza: `[1, 2]`
Differenza simmetrica: `[1, 2, 4, 5]`
Prodotto cartesiano: `[(1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 3), (3, 4), (3, 5)]`

### Test Prolog 7

Primo insieme: `[1, 2, 3]`
Secondo insieme: `[3, 2, 1]`

Unione: `[1, 2, 3]`
Intersezione: `[1, 2, 3]`
Differenza: `[]`
Differenza simmetrica: `[]`
Prodotto cartesiano: `[(1, 3), (1, 2), (1, 1), (2, 3), (2, 2), (2, 1), (3, 3), (3, 2), (3, 1)]`

### Test Prolog 8

Primo insieme: `[1, 2, 3]`
Secondo insieme: `[6, 5, 4]`

Unione: `[1, 2, 3, 6, 5, 4]`
Intersezione: `[]`
Differenza: `[1, 2, 3]`
Differenza simmetrica: `[1, 2, 3, 6, 5, 4]`
Prodotto cartesiano: `[(1, 6), (1, 5), (1, 4), (2, 6), (2, 5), (2, 4), (3, 6), (3, 5), (3, 4)]`

### Test Prolog 9

Primo insieme: `[1, 2, 3]`
Secondo insieme: `[2, 3, 0]`

Unione: `[1, 2, 3, 0]`
Intersezione: `[2, 3]`
Differenza: `[1]`
Differenza simmetrica: `[1, 0]`
Prodotto cartesiano: `[(1, 2), (1, 3), (1, 0), (2, 2), (2, 3), (2, 0), (3, 2), (3, 3), (3, 0)]`

### Test Prolog 10

Primo insieme: `[1, 2, 3]`
Secondo insieme: `[]`

Unione: `[1, 2, 3]`
Intersezione: `[]`
Differenza: `[1, 2, 3]`
Differenza simmetrica: `[1, 2, 3]`
Prodotto cartesiano: `[]`