UNIVERSITÀ DEGLI STUDI DI URBINO CARLO BO
Corso di Laurea in Informatica Applicata

"Programma per calcolare le chiusure di una relazione."

Relazione del progetto d'esame per il corso di
Programmazione Logica e Funzionale

Sessione Autunnale a.a. 2024/25

Balducci Milena Matricola 321791
Guardabascio Edoardo Matricola 323174

1 Specifica del problema

Scrivere un programma Haskell e un programma Prolog che acquisiscono dalla tastiera un insieme finito di numeri naturali e una relazione binaria su quell'insieme e poi stampano sullo schermo la chiusura riflessiva, simmetrica e transitiva di quella relazione, calcolata ricorsivamente.

2 Analisi del problema

2.1 Dati di Ingresso del Problema

I dati di ingresso sono costituiti da un insieme finito di numeri interi e da un insieme di coppie ordinate di numeri interi, che rappresenta una relazione binaria definita sull'insieme.

2.2 Dati di Uscita del Problema

I dati di uscita sono tre insiemi di coppie ordinate di numeri interi, che rappresentano rispettivamente:

- La chiusura riflessiva della relazione;
- La chiusura simmetrica della relazione;
- La chiusura transitiva della relazione.

2.3 Relazioni Intercorrenti tra i Dati del Problema

Dati due insiemi A e B, una relazione R tra i due insiemi è un sottoinsieme del prodotto cartesiano $A \times B$, e si indica con $R \subseteq A \times B$.
Ai fini della risoluzione del problema, è utile riportare le definizioni di dominio, codominio e campo di una relazione. Sia $R \subseteq A \times B$:

- Il dominio di R è l'insieme $dom(R) = \{a \in A \mid \exists b \in B : (a,b) \in R\}$;
- Il codominio di R è l'insieme $cod(R) = \{b \in B \mid \exists a \in A : (a,b) \in R\}$;
- Il campo di R è l'insieme $campo(R) = dom(R) \cup cod(R) \subseteq A \cup B$.

Quando $A = B$, si dice che R è binaria su A.

Data una relazione $R \subseteq A \times A$, si dice che essa è:

- Riflessiva se e solo se $\forall a \in campo(R),\ (a,a) \in R$;
- Simmetrica se e solo se $\forall a_1, a_2 \in campo(R),\ (a_1,a_2) \in R \longrightarrow (a_2,a_1) \in R$;
- Transitiva se e solo se $\forall a_1, a_2, a_3 \in campo(R),\ (a_1,a_2) \in R \wedge (a_2,a_3) \in R \rightarrow (a_1,a_3) \in R$.

La chiusura riflessiva di una relazione R è l'insieme $R^{\Gamma} = R \cup I_{campo(R)}$, dove $I_{campo(R)} = \{(a,a) \mid a \in campo(R)\}$ è la relazione identità su $campo(R) \subseteq A$.
Per calcolare la chiusura riflessiva di una relazione è sufficiente determinare l'insieme identità, che può essere definito in maniera equivalente come $I_{campo(R)} = \{(x,x) \mid \exists (a,b) \in R : x = a \vee x = b\}$, ed effettuare l'unione tra I e R.

La chiusura simmetrica di una relazione R è l'insieme $R^S = R \cup R^{-1}$, dove $R^{-1} = \{(a_2,a_1) \mid (a_1,a_2) \in R\}$ è la relazione inversa di R. Per calcolarla è sufficiente determinare R^{-1} ed effettuare l'unione tra i due insiemi.

La chiusura transitiva di R, indicata con R^*, è la più piccola relazione transitiva su A tale che $R \subseteq R^*$. Formalmente:

$$\forall a,b \in A,\ (a,b) \in R^* \iff \exists n \geq 1,\ \exists x_0, x_1, ..., x_n \in A \mid x_0 = a, x_n = b, (x_i, x_{i+1}) \in R\ \ \forall i = 0, 1, ..., n-1$$

Per calcolare la chiusura transitiva di una relazione R è dunque necessario verificare, per ogni coppia di elementi $a,b \in campo(R)$, se esiste un percorso da a a b. In caso affermativo, se la coppia (a,b) non appartiene già a R, essa viene aggiunta.

3 Progettazione dell'Algoritmo

3.1 Scelte di Progetto

Gli insiemi finiti di numeri interi possono essere rappresentati in modo naturale tramite strutture dati lineari. Poiché la cardinalità degli insiemi non è nota a priori, tali strutture verranno allocate dinamicamente, evitando così qualsiasi limitazione rispetto alla specifica del problema.

Durante l'acquisizione è possibile che compaiano elementi duplicati; questi verranno successivamente rimossi nella fase di memorizzazione, in quanto la presenza di duplicati non influisce sulla definizione matematica di insieme.

3.2 Passi dell'Algoritmo

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

- Acquisire l'insieme finito di numeri interi.
- Acquisire la relazione binaria sull'insieme.
- Calcolare e stampare la chiusura riflessiva della relazione:
  - Caso base: se la relazione è vuota, la sua chiusura riflessiva è vuota.
  - Caso generale: se la relazione non è vuota, per ciascun elemento si costruisce la coppia che lo mette in relazione con se stesso; si verifica se tale coppia è già presente e, in caso contrario, la si aggiunge.
- Calcolare la chiusura simmetrica della relazione:
  - Caso base: se la relazione è vuota, la sua chiusura simmetrica è vuota.
  - Caso generale: se la relazione non è vuota, per ogni coppia ordinata si controlla se è già presente la coppia invertita; in caso contrario, questa viene aggiunta.
- Calcolare la chiusura transitiva della relazione:
  - Caso base: se la relazione è vuota, la sua chiusura transitiva è vuota.
  - Caso generale: se la relazione non è vuota, si esaminano le coppie presenti. Ogni volta che si trovano una coppia (a,b) e una coppia (b,c), si aggiunge la coppia (a,c) se non è già presente. Il procedimento viene ripetuto considerando anche le nuove coppie inserite, poiché queste possono a loro volta generare ulteriori coppie. Quando un'intera scansione della relazione non produce più nuovi elementi, il processo termina.

4 Implementazione dell'Algoritmo

File sorgente `chiusure_relazione.hs`:

```
{- Programma Haskell per calcolare le chiusure di una relazione -}

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

main :: IO ()
main = do putStrLn "Inserisci l'insieme di numeri naturali separati da spazi:"
          insiemeStr <- getLine
          let insieme = nub (map read (words insiemeStr) :: [Int])
          putStrLn $ "Insieme privo di elementi duplicati: " ++ show insieme
          relazione <- acquisisci_relazione insieme
          putStrLn $ "Relazione priva di elementi duplicati: " ++ show relazione
          putStr "Chiusura riflessiva di R:"
          putStrLn $ show (riflessiva relazione)
          putStr "Chiusura simmetrica di R:"
          putStrLn $ show (simmetrica relazione)
          putStr "Chiusura transitiva di R:"
          putStrLn $ show (transitiva relazione)

{- L'azione di input/output acquisisci_relazione acquisisce la relazione da tastiera,
   verifica che sia valida sull'insieme e la restituisce priva di eventuali duplicati:
   - il suo unico argomento è l'insieme sul quale la relazione deve essere valida. -}

acquisisci_relazione :: [Int] -> IO [(Int, Int)]
acquisisci_relazione insieme = do putStrLn "Inserisci le coppie separate da spazi (es 1,2 2,3):"
                                  relazioneStr <- getLine
                                  let rel = nub (map parse_coppia (words relazioneStr))
                                  if all (\(a,b) -> a `elem` insieme && b `elem` insieme) rel
                                     then return rel
                                  else do putStrLn "Errore: alcune coppie contengono elementi
                                  non presenti nell'insieme."
                                      acquisisci_relazione insieme

{- La funzione parse_coppia converte una stringa del tipo "1,2" in una coppia di interi (1,2):
   - il suo unico argomento è la stringa da trasformare. -}

parse_coppia :: String -> (Int, Int)
parse_coppia s = let [a,b] = map read (split_stringa (==',') s)
                 in (a,b)

{- La funzione split_stringa spezza una stringa in sottostringhe,
   usando come separatore qualunque carattere soddisfi p:
   - il primo argomento è il predicato da soddisfare
   - il secondo argomento è la stringa da spezzare -}

split_stringa :: (Char -> Bool) -> String -> [String]
split_stringa p s = case dropWhile p s of
                    "" -> []
                    s' -> w : split_stringa p s''
                        where (w, s'') = break p s'

{- La funzione riflessiva calcola la chiusura riflessiva di una relazione:
   - il suo unico argomento è la relazione stessa -}

riflessiva :: [(Int, Int)] -> [(Int, Int)]
```

```
riflessiva [] = []
riflessiva rel = nub (rel ++ [(x,x) | (a,b) <- rel, x <- [a,b], (x,x) `notElem` rel])

{- La funzione simmetrica calcola la chiusura simmetrica di una relazione,
   - il suo unico argomento è la relazione stessa -}

simmetrica :: [(Int, Int)] -> [(Int, Int)]
simmetrica [] = []
simmetrica rel = nub $ rel ++ [(b,a) | (a,b) <- rel, (b,a) `notElem` rel]

{- La funzione transitiva calcola la chiusura transitiva di una relazione:
   - il suo unico argomento è la relazione stessa.-}

transitiva :: [(Int, Int)] -> [(Int, Int)]
transitiva [] = []  -- caso base esplicito
transitiva rel = let nuove = [(a,d) | (a,b) <- rel, (c,d) <- rel, b == c, (a,d) `notElem` rel]
                 in if null nuove
                    then rel
                    else transitiva (nub (rel ++ nuove))
```

File sorgente `chiusure_relazione.pl`:

```
/* Programma Prolog per calcolare le chiusure di una relazione */

main :- write('Inserisci l\'insieme di numeri naturali tra parentesi quadre: '), nl,
        read(I),
        sort(I, IU),
        format('Insieme privo di duplicati: ~w~n', [IU]),
        acquisisci_relazione(IU, R),
        format('Relazione priva di duplicati: ~w~n', [R]),
        riflessiva(R, CR),
        format('Chiusura riflessiva di R: ~w~n', [CR]),
        simmetrica(R, CS),
        format('Chiusura simmetrica di R: ~w~n', [CS]),
        transitiva(R, CT),
        format('Chiusura transitiva di R: ~w~n', [CT]).

/* Il predicato acquisisci_relazione acquisisce la relazione da tastiera,
   verifica che sia valida sull'insieme e la restituisce priva di eventuali duplicati:
   -il primo argomento è l'insieme sul quale la relazione deve essere valida
   -il secondo argomento è la relazione, priva di duplicati.*/

acquisisci_relazione(I, RU) :- write('Inserisci la relazione come lista di coppie (es [(1,2),(2,3)]): '), nl,
                               read(R), sort(R, RU),
                               ( relazione_valida(RU, I)
                               -> true
                               ; write('Errore: alcune coppie contengono elementi non presenti nell'insieme.'), nl
                                 acquisisci_relazione(I, RU)
                               ).

/* Il predicato relazione_valida verifica che la relazione sia valida sull'insieme inserito:
   -il primo argomento è la relazione
   -il secondo argomento è l'insieme.*/

relazione_valida([], _I).
relazione_valida([(A,B)|T], I) :- member(A, I), member(B, I), relazione_valida(T, I).


/* il predicato riflessiva calcola la chiusura riflessiva di una relazione:
   - il primo argomento è la relazione
   - il secondo argomento è la chiusura riflessiva.*/

riflessiva([], []).
riflessiva(R, CR) :- findall((X,X), (member((A,B), R), (X=A; X=B)), CPR),
                     append(R, CPR, Temp), sort(Temp, CR).

/* il predicato simmetrica calcola la chiusura simmetrica di una relazione,
   -il primo argomento è la relazione
   -il secondo argomento è la chiusura simmetrica.*/

simmetrica([], []).
simmetrica(R, CS) :- findall((B,A), (member((A,B), R), \+ member((B,A), R)), CPS),
                     append(R, CPS, Temp), sort(Temp, CS).

/* il predicato transitiva calcola la chiusura transitiva di una relazione:
   -il primo argomento è la relazione
   -il secondo argomento è la chiusura transitiva.*/
```

```
transitiva([], []).
transitiva(R, CT) :- findall((A,D), (member((A,B), R), member((B,D), R), \+ member((A,D), R)), Nuove),
                     (Nuove = [] ->  CT = R ; append(R, Nuove, Temp),
                     sort(Temp, TempOrd), transitiva(TempOrd, CT)).
```

5 Testing del Programma

Test Haskell 1
Insieme: `[1,2,3]`
Relazione: `[(1,2), (2,3)]`
Chiusura riflessiva: `[(1,2), (2,3), (1,1), (2,2), (3,3)]`
Chiusura simmetrica: `[(1,2), (2,3), (2,1), (3,2)]`
Chiusura transitiva: `[(1,2), (2,3), (1,3)]`

Test Haskell 2
Insieme: `[1, 2]`
Relazione: `[(1,1), (2,2)]`
Chiusura riflessiva: `[(1,1), (2,2)]`
Chiusura simmetrica: `[(1,1), (2,2)]`
Chiusura transitiva: `[(1,1), (2,2)]`

Test Haskell 3
Insieme: `[1, 2, 3]`
Relazione: `[(1,2),(2,1),(2,3),(3,2)]`
Chiusura riflessiva: `[(1,2),(2,1),(2,3),(3,2),(1,1),(2,2),(3,3)]`
Chiusura simmetrica: `[(1,2),(2,1),(2,3),(3,2)]`
Chiusura transitiva: `[(1,2),(2,1),(2,3),(3,2),(1,1),(1,3),(2,2),(3,1),(3,3)]`

Test Haskell 4
Insieme: `[1, 2, 4]`
Relazione: `[(1,2),(1,4),(2,4)]`
Chiusura riflessiva: `[(1,2),(1,4),(2,4),(1,1),(2,2),(4,4)]`
Chiusura simmetrica: `[(1,2),(1,4),(2,4),(2,1),(4,1),(4,2)]`
Chiusura transitiva: `[(1,2),(1,4),(2,4)]`

Test Haskell 5
Insieme: `[1, 2, 3]`
Relazione: `[]`
Chiusura riflessiva: `[]`
Chiusura simmetrica: `[]`
Chiusura transitiva: `[]`

Test Haskell 6
Insieme: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
Relazione: `[(1,2),(2,3),(3,5),(5,7),(8,9)]`
Chiusura riflessiva: `[(1,2),(2,3),(3,5),(5,7),(8,9),(1,1),(2,2),(3,3),(5,5),(7,7),(8,8),(9,9)]`
Chiusura simmetrica: `[(1,2),(2,3),(3,5),(5,7),(8,9),(2,1),(3,2),(5,3),(7,5),(9,8)]`
Chiusura transitiva: `[(1,2),(2,3),(3,5),(5,7),(8,9),(1,3),(2,5),(3,7),(1,5),(2,7),(1,7)]`

Test Haskell 7
Insieme: `[1, 2, 3]`
Relazione: `[(1,1),(1,2),(2,2),(2,3)]`
Chiusura riflessiva: `[(1,1),(1,2),(2,2),(2,3),(3,3)]`
Chiusura simmetrica: `[(1,1),(1,2),(2,2),(2,3),(2,1),(3,2)]`
Chiusura transitiva: `[(1,1),(1,2),(2,2),(2,3),(1,3)]`

Test Haskell 8
Insieme: `[1, 2, 3, 4]`
Relazione: `[(1,2),(2,3),(3,4)]`
Chiusura riflessiva: `[(1,2),(2,3),(3,4),(1,1),(2,2),(3,3),(4,4)]`
Chiusura simmetrica: `[(1,2),(2,3),(3,4),(2,1),(3,2),(4,3)]`
Chiusura transitiva: `[(1,2),(2,3),(3,4),(1,3),(2,4),(1,4)]`

Test Haskell 9
Insieme: `[1, 2, 3]`
Relazione: `[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)]`
Chiusura riflessiva: `[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)]`
Chiusura simmetrica: `[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)]`
Chiusura transitiva: `[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)]`

Test Haskell 10
Insieme: `[1, 2, 3]`
Relazione: `[(1,2),(2,1),(2,3),(3,2)]`
Chiusura riflessiva: `[(1,2),(2,1),(2,3),(3,2),(1,1),(2,2),(3,3)]`
Chiusura simmetrica: `[(1,2),(2,1),(2,3),(3,2)]`
Chiusura transitiva: `[(1,2),(2,1),(2,3),(3,2),(1,1),(1,3),(2,2),(3,1),(3,3)]`

Test Prolog 1
Insieme: `[1,2,3]`
Relazione: `[(1,2), (2,3)]`
Chiusura riflessiva: `[(1,1),(1,2),(2,2),(2,3),(3,3)]`
Chiusura simmetrica: `[(1,2),(2,1),(2,3),(3,2)]`
Chiusura transitiva: `[(1,2),(1,3),(2,3)]`

Test Prolog 2
Insieme: `[1, 2]`
Relazione: `[(1,1), (2,2)]`
Chiusura riflessiva: `[(1,1), (2,2)]`
Chiusura simmetrica: `[(1,1), (2,2)]`
Chiusura transitiva: `[(1,1), (2,2)]`

Test Prolog 3
Insieme: `[1, 2, 3]`
Relazione: `[(1,2),(2,1),(2,3),(3,2)]`
Chiusura riflessiva: `[(1,1),(1,2),(2,1),(2,2),(2,3),(3,2),(3,3)]`
Chiusura simmetrica: `[(1,2),(2,1),(2,3),(3,2)]`
Chiusura transitiva: `[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)]`

Test Prolog 4
Insieme: `[1, 2, 4]`
Relazione: `[(1,2),(1,4),(2,4)]`
Chiusura riflessiva: `[(1,1),(1,2),(1,4),(2,2),(2,4),(4,4)]`
Chiusura simmetrica: `[(1,2),(1,4),(2,1),(2,4),(4,1),(4,2)]`
Chiusura transitiva: `[(1,2),(1,4),(2,4)]`

Test Prolog 5
Insieme: `[1, 2, 3]`
Relazione: `[]`
Chiusura riflessiva: `[]`
Chiusura simmetrica: `[]`
Chiusura transitiva: `[]`

Test Prolog 6
Insieme: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
Relazione: `[(1,2),(2,3),(3,5),(5,7),(8,9)]`
Chiusura riflessiva: `[(1,1),(1,2),(2,2),(2,3),(3,3),(3,5),(5,5),(5,7),(7,7),(8,8),(8,9),(9,9)]`
Chiusura simmetrica: `[(1,2),(2,1),(2,3),(3,2),(3,5),(5,3),(5,7),(7,5),(8,9),(9,8)]`
Chiusura transitiva: `[(1,2),(1,3),(1,5),(1,7),(2,3),(2,5),(2,7),(3,5),(3,7),(5,7),(8,9)]`

Test Prolog 7
Insieme: `[1, 2, 3]`
Relazione: `[(1,1),(1,2),(2,2),(2,3)]`
Chiusura riflessiva: `[(1,1),(1,2),(2,2),(2,3),(3,3)]`
Chiusura simmetrica: `[(1,1),(1,2),(2,1),(2,2),(2,3),(3,2)]`
Chiusura transitiva: `[(1,1),(1,2),(1,3),(2,2),(2,3)]`

Test Prolog 8
Insieme: `[1, 2, 3, 4]`
Relazione: `[(1,2),(2,3),(3,4)]`
Chiusura riflessiva: `[(1,1),(1,2),(2,2),(2,3),(3,3),(3,4),(4,4)]`

Chiusura simmetrica: `[(1,2),(2,1),(2,3),(3,2),(3,4),(4,3)]`
Chiusura transitiva: `[(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)]`

Test Prolog 9
Insieme: `[1, 2, 3]`
Relazione: `[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)]`
Chiusura riflessiva: `[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)]`
Chiusura simmetrica: `[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)]`
Chiusura transitiva: `[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)]`

Test Prolog 10
Insieme: `[1, 2, 3]`
Relazione: `[(1,2),(2,1),(2,3),(3,2)]`
Chiusura riflessiva: `[(1,1),(1,2),(2,1),(2,2),(2,3),(3,2),(3,3)]`
Chiusura simmetrica: `[(1,2),(2,1),(2,3),(3,2)]`
Chiusura transitiva: `[(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)]`