Progetto Programmazione Procedurale

Alessio Muzi | Informatica Applicata | A.S. 2021/2022

Sommario:

1 – Specifica del problema

Un alfabeto è un insieme finito di simboli. Un linguaggio finito su un alfabeto è un insieme finito di sequenze di lunghezza finita di simboli dell'alfabeto. Dato l'alfabeto {*a, e, i, o, u*}, scrivere un programma ANSI C che acquisisce da tastiera due linguaggi finiti su quell'alfabeto e poi stampa sullo schermo l'unione dei due linguaggi e la differenza tra il primo linguaggio e il secondo. Almeno una delle due operazioni deve essere calcolata ricorsivamente.

2 – Analisi del problema

Dati di ingresso del problema:

I dati di ingresso del problema sono rappresentati da due linguaggi finiti sull'alfabeto {a, e, i, o, u}.

Dati di uscita del problema:

I dati di uscita del problema sono rappresentati dall'unione e la differenza dei due linguaggi inseriti dall'utente.

Relazioni intercorrenti tra i dati del problema:

Un **alfabeto** è un insieme finito di simboli. Un **linguaggio finito** su un alfabeto è un insieme finito di sequenze di lunghezza finita di simboli dell'alfabeto. In una **sequenza** (o **parola**) i simboli dell'alfabeto possono ripetersi.

L'**unione** è un'operazione binaria commutativa e associativa denotata con il simbolo '∪' , che ha come elemento neutro l'insieme vuoto ∅. L'unione di due insiemi A e B è un insieme formato da tutti e soli gli elementi che appartengono ad A, appartengono a B o appartengono ad entrambi.

$$A \cup B := \{x \in E \mid x \in A \ \vee \ x \in B\}$$

La **differenza** è un'operazione binaria non commutativa denotata con il simbolo '\' o '-' . La differenza di due insiemi A e B è un'insieme formato da tutti e soli gli elementi che appartengono ad A ma non appartengono a B.

$$A - B := \{x \in E \mid x \in A \ \wedge \ x \notin B\}$$

In un insieme sono irrilevanti sia l'**ordine** che la **molteplicità** degli elementi.

Sono validi sia la **sequenza vuota** (ovvero la sequenza ε di lunghezza 0) e il **linguaggio vuoto** (ovvero l'insieme che contiene 0 sequenze).

3 – Progettazione dell’algoritmo

Scelte di progetto:

Per rappresentare i linguaggi verranno utilizzate delle **matrici** (array bidimensionali), dove ogni riga corrisponde ad una parola. Le matrici verranno allocate dinamicamente, poiché non si conosce a priori né il numero delle parole (le righe) né la lunghezza delle parole (le colonne). Queste dimensioni verranno richieste all’utente prima di effettuare l’inserimento.

Le parole verranno inserite **una alla volta**, **un simbolo alla volta**. Se i due linguaggi sono vuoti, non sarà possibile eseguire nessun inserimento.

Per ogni inserimento di un linguaggio bisogna effettuare un **controllo** nel caso in cui l’utente abbia inserito più volte la stessa parola. Inoltre, mentre la differenza garantisce l’unicità degli elementi nell’insieme risultato, ciò non vale per l’unione, poiché potrebbe essere presente la stessa parola nell’insieme A e nell’insieme B. Bisognerà quindi effettuare un ulteriore controllo. Per identificare le parole duplicate e le sequenze vuote, verranno usate delle **lettere contrassegno** (rispettivamente ‘D’ e ‘V’).

Quando si effettua la **ricerca dei duplicati**, è sufficiente confrontare ogni parola con le successive, mentre quando si esegue la **differenza**, ogni parola del primo linguaggio va confrontata con tutte le parole del secondo.

Passi dell’algoritmo:

- Inserimento del numero di parole del primo linguaggio;
- Inserimento del numero di parole del secondo linguaggio;
- Inserimento della lunghezza massima possibile di una parola;
- Allocazione dinamica della memoria dei linguaggi di input e output;
- Inserimento del primo linguaggio, parola per parola:
    - Ad ogni inserimento, viene chiesta la lunghezza della parola che si intende inserire.
    - I simboli vengono inseriti uno alla volta.
    - Una volta inserite tutte le parole, avviene la ricerca delle parole duplicate.
- Inserimento del secondo linguaggio, seguendo lo stesso procedimento del primo;
- Unione dei due linguaggi, calcolata in modo ricorsivo:
    - Inizialmente, tutte le parole del primo linguaggio vengono caricate in modo ricorsivo nel linguaggio output, poi inizia l’unione:
    - *Caso base*: se il secondo insieme è vuoto, non c’è nessun elemento da inserire nell’unione dei linguaggi.
    - *Caso generale*: se il secondo insieme non è vuoto, allora il primo elemento del secondo insieme viene inserito nel linguaggio unione.
    - Una volta terminata l’unione, avviene la ricerca delle parole duplicate.
- Differenza dei due linguaggi;
- Stampa dell’unione e della differenza dei due linguaggi.

4 – Implementazione dell’algoritmo

File sorgente `linguaggi.c`

```
/* programma per l'unione e la differenza dei linguaggi */
/* inclusione delle librerie */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* definizione delle costanti simboliche */
#define DOPPIONE       'D' /* contrassegno per le parole duplicate */
#define SEQUENZA_VUOTA 'V' /* contrassegno per le sequenze vuote */

/* dichiarazione delle funzioni */
int    inserisci_dim(char*);
char** allocazione_dinamica(int,
                            int);
char** inserisci_linguaggio(int,
                            int,
                            char*,
                            char**);
char** ricerca_duplicati(int,
                         char**);
char** unione_linguaggi(int,
                        int,
                        int,
                        int,
                        char**,
                        char**,
                        char**);
char** differenza_linguaggi(int,
                            int,
                           char**,
                           char**,
                           char**);
void   stampa_linguaggio(int,
                         char*,
                         char**);

/* definizione delle funzioni */
/* definizione della funzione main */
int main(void)
{
    char **linguaggio1,     /* input: primo linguaggio */
         **linguaggio2,     /* input: secondo linguaggio */
         **unione,          /* output: unione dei due linguaggi */
         **differenza;      /* output: differenza dei due linguaggi */
    int  num_parole1,       /* lavoro: numero di parole del primo linguaggio */
         num_parole2,       /* lavoro: numero di parole del secondo linguaggio */
         lunghezza_max = 0; /* lavoro: lunghezza massima delle parole */

    /* inserimento del numero di parole dei due linguaggi input */
    num_parole1 = inserisci_dim("il numero di parole del primo linguaggio");
    num_parole2 = inserisci_dim("il numero di parole del secondo linguaggio");

    /* inserimento della lunghezza massima possibile delle parole */
    if ((num_parole1 == 0) && (num_parole2 == 0))
     printf("Entrambi i linguaggi sono vuoti, non è possibile eseguire nessun inserimento.\n");
```

```
    else
     lunghezza_max = inserisci_dim("la lunghezza massima possibile di una parola. Le
                                    sequenze vuote hanno lunghezza 0");
    /* allocazione dinamica dei linguaggi input e output */
    linguaggio1 = allocazione_dinamica(num_parole1,
                                       lunghezza_max);
    linguaggio2 = allocazione_dinamica(num_parole2,
                                       lunghezza_max);
    unione = allocazione_dinamica((num_parole1 + num_parole2),
                                  lunghezza_max);
    differenza = allocazione_dinamica(num_parole1,
                                      lunghezza_max);

    /* promemoria dei simboli validi */
    printf("\nSimboli validi: {a, e, i, o, u}. Digitare '0' per inserire una sequenza
            vuota.\n");
    /* inserimento dei due linguaggi */
    if (num_parole1 != 0)
     linguaggio1 = inserisci_linguaggio(num_parole1,
                                        lunghezza_max,
                                        "primo",
                                        linguaggio1);
    else
     printf("\nIl primo linguaggio è vuoto.\n");

    if (num_parole2 != 0)
     linguaggio2 = inserisci_linguaggio(num_parole2,
                                        lunghezza_max,
                                        "secondo",
                                        linguaggio2);
    else
     printf("\nIl secondo linguaggio è vuoto.\n");

    /* unione dei due linguaggi */
    unione = unione_linguaggi(num_parole1,
                              num_parole2,
                              0,
                              0,
                              linguaggio1,
                              linguaggio2,
                              unione);
    /* differenza dei due linguaggi */
    differenza = differenza_linguaggi(num_parole1,
                                      num_parole2,
                                      linguaggio1,
                                      linguaggio2,
                                      differenza);

    /* stampa dell'unione dei due linguaggi */
    stampa_linguaggio((num_parole1 + num_parole2),
                      "Unione",
                      unione);
    /* stampa della differenza dei due linguaggi */
    stampa_linguaggio(num_parole1,
                      "Differenza",
                      differenza);
    printf("\n");
    return(0);
}
```

```
/* definizione della funzione per inserire le dimensioni del linguaggio */
int inserisci_dim(char* messaggio) /* input: messaggio da stampare */
{
    int dim,     /* output: dimensione da inserire */
       esito;   /* lavoro: validazione degli input */


printf("Inserisci %s.\n",
       messaggio);
    /* lettura e validazione delle dimensioni */
    do
    {
     esito = scanf("%d",
                   &dim);
     if (esito != 1 || dim < 0)
      printf("Valore non accettabile!\n");
     while (getchar() != '\n');
    }
    while (esito != 1 || dim < 0);

    return(dim);
}

/* definizione della funzione per l'allocazione dinamica della memoria */
char** allocazione_dinamica(int righe,   /* input: righe della matrice */
                            int colonne) /* input: colonne della matrice */
{
    char** matrice; /* output: variabile per l'allocazione dinamica */
    int    i;       /* lavoro: indice per l'allocazione */

    /* allocazione dinamica */
    matrice = (char  **)malloc(righe * sizeof(char *));
     for (i = 0;
          i < righe;
          i++)
      matrice[i] = (char  *)malloc(colonne * sizeof(char));

    return(matrice);
}

/* definizione della funzione per l'inserimento dei linguaggi */
char** inserisci_linguaggio(int    parole,        /* input: numero di parole  */
                            int    lunghezza_max, /* input: lunghezza massima delle
                                                     parole */
                            char*  messaggio,     /* input: messaggio da stampare */
                            char** matrice)       /* input: matrice allocata
                                                     dinamicamente */
{
    int lunghezza_parola, /* input: lunghezza della prossima parola */
       i,                 /* lavoro: indice per la lunghezza delle parole */
       j,                 /* lavoro: indice per l'inserimento dei simboli */
       esito;             /* lavoro: validazione degli input */

    printf("\nInserimento del %s linguaggio.\n",
           messaggio);

    for (i = 0;
         i < parole;
         i++)
     {
      /* lettura e validazione della lunghezza della prossima parola */
```

```
      printf("Inserisci la lunghezza della prossima parola.\n");
      do
       {
        esito = scanf("%d",
                      &lunghezza_parola);
        if (esito != 1 || lunghezza_parola < 0 || lunghezza_parola > lunghezza_max)
         printf("Valore non accettabile!\n");
        while (getchar() != '\n');
       }
      while (esito != 1 || lunghezza_parola < 0 || lunghezza_parola > lunghezza_max);

/* inserimento di un contrassegno per la SEQUENZA VUOTA */
     if (lunghezza_parola == 0)
      matrice[i][0] = SEQUENZA_VUOTA;
     else
      /* inserimento della parola, simbolo per simbolo */
      for (j = 0;
           j < lunghezza_parola;
           j++)
      {
       /* lettura e validazione dei simboli */
       printf("Inserisci il prossimo simbolo.\n");
       do
       {
        esito = scanf("%c",
                      &matrice[i][j]);
       if (esito != 1 || ((matrice[i][j] != 'a') && (matrice[i][j] != 'e') &&
           (matrice[i][j] != 'i') && (matrice[i][j] != 'o') && (matrice[i][j] != 'u')))
        printf("Simbolo non accettabile!\n");
       while (getchar() != '\n');
       }
   while (esito != 1 || ((matrice[i][j] != 'a') && (matrice[i][j] != 'e') &&
          (matrice[i][j] != 'i') && (matrice[i][j] != 'o') && (matrice[i][j] != 'u')));
      }
    }
    /* ricerca di parole duplicate */
    matrice = ricerca_duplicati(parole,
                                matrice);

    return(matrice);
}

/* definizione della funzione per la ricerca di parole duplicate */
char** ricerca_duplicati(int    parole, /* input: numero di parole del linguaggio */
                         char** matrice)/* input: linguaggio da cui rimuovere i duplicati */
{
    int i,         /* lavoro: indice per le righe */
        j,         /* lavoro: indice per il confronto */
        duplicato; /* lavoro: variabile per la ricerca di parole duplicate */

    /* confronto tra righe in ricerca delle parole duplicate */
    for (i = 0;
         i < parole;
         i++)
     {
      /* è sufficiente confrontare ogni i-esima parola con le parole a partire dalla
        posizione i + 1 */
      for (j = (i + 1);
           j < parole;
           j++)
       {
```

```
        /* se due parole sono uguali, sostituiamo la seconda parola con DOPPIONE */
        duplicato = (strcmp(matrice[i], matrice[j]));
        if (duplicato == 0)
         matrice[j][0] = DOPPIONE;
       }
      }

    return(matrice);
}

/* definizione della funzione ricorsiva per l'unione dei due linguaggi */
char** unione_linguaggi(int    num_parole1,  /* input: numero parole del primo
                                                linguaggio */
                        int    num_parole2,  /* input: numero parole del secondo
                                               linguaggio */
                        int    i,            /* input: indice linguaggio1 e linguaggio
                                               unione */
                        int    j,            /* input: indice linguaggio2 */
                        char** linguaggio1,  /* input: primo linguaggio */
                        char** linguaggio2,  /* input: secondo linguaggio */
                        char** unione)       /* input: matrice risultato allocata
                                               dinamicamente */
{
    /* prima fase dell'unione: vengono caricate tutte le parole del primo linguaggio */
    if (num_parole1 != 0)
    {
     strcpy(unione[i], linguaggio1[i]);
     unione_linguaggi((num_parole1 - 1),
                      num_parole2,
                      (i + 1),
                      j,
                      linguaggio1,
                     linguaggio2,
                      unione);
    }
     /* seconda fase dell'unione: viene eseguita l'unione dei due linguaggi */
     /* se il secondo insieme non ha più elementi, l'unione termina */
     else if (num_parole2 != 0)
     {
      strcpy(unione[i], linguaggio2[j]);
      unione_linguaggi(num_parole1,
                      (num_parole2 - 1),
                      (i + 1),
                      (j + 1),
                      linguaggio1,
                       linguaggio2,
                      unione);
    }
    /* ricerca di parole duplicate nel linguaggio unione */
    unione = ricerca_duplicati((num_parole1 + num_parole2),
                               unione);

    return(unione);
}

/* definizione della funzione per la differenza dei due linguaggi */
       char** differenza_linguaggi(int    num_parole1,  /* input: numero parole del primo
                                                           linguaggio */
                                   int    num_parole2,   /* input: numero parole del secondo
                                                           linguaggio */
                                   char** linguaggio1, /* input: primo linguaggio */
```

```
                                char** linguaggio2, /* input: secondo linguaggio */
                                char** differenza)  /* input: matrice risultato allocata
                                                       dinamicamente */
{
    int i,    /* lavoro: indice linguaggio1 */
        j,    /* lavoro: indice linguaggio2 */
        k,    /* lavoro: indice linguaggio differenza */
        diff; /* lavoro: variabile per la differenza */

    /* differenza dei linguaggi */
    for (i = 0, k = 0;
         i < num_parole1;
         i++)
     {
      for (j = 0, diff = 1;
           j < num_parole2;
           j++)
       /* confronto tra la i-esima parola del primo linguaggio e tutte le parole del
          secondo linguaggio */
       diff *= (strcmp(linguaggio1[i], linguaggio2[j]));
      if (diff != 0)
      {
       /* inserimento delle parole nel linguaggio differenza */
       strcpy(differenza[k], linguaggio1[i]);
       k++;
      }
     }

    return(differenza);
}

/* definizione della funzione per la stampa dei due linguaggi */
void stampa_linguaggio(int    parole,    /* input: numero di parole */
                       char*  messaggio, /* input: messaggio da stampare */
                       char** matrice)   /* output: matrice da stampare */
{
    int i,              /* lavoro: indice per la stampa */
        insieme_vuoto;  /* lavoro: stampa dell'insieme vuoto */

    printf("\n%s dei due linguaggi:\n\n",
           messaggio);

    /* stampa delle parole */
    for (i = 0, insieme_vuoto = 0;
         i < parole;
         i++)
      /* stampa della sequenza vuota */
      if (matrice[i][0] == SEQUENZA_VUOTA)
      {
       printf(" *sequenza vuota*\n");
       insieme_vuoto++;
      }
      /* se il programma legge DOPPIONE, la parola non verrà stampata */
      else if ((matrice[i][0] != DOPPIONE) && (matrice[i][0] != '\0'))
      {
       insieme_vuoto++;
       printf(" %s\n",
              matrice[i]);
      }
    /* se il programma non ha stampato nulla, allora il linguaggio è vuoto */
    if (insieme_vuoto == 0)
     printf(" Insieme vuoto\n");
}
```

Makefile

```
#
# Makefile per la compilazione del file sorgente "linguaggi.c"
#
linguaggi: linguaggi.c Makefile
     gcc -ansi -Wall -O linguaggi.c -o linguaggi
pulisci:
     rm -f linguaggi.o
pulisci_tutto:
     rm -f linguaggi linguaggi.o
```

5 – Testing del programma

I test effettuati rivelano che il programma si comporta in modo corretto se vengono inseriti degli input errati. In particolare il programma non avanza se:

- Viene inserito un numero di parole negativo o non viene inserito un numero,
- Viene inserita una lunghezza massima negativa o non viene inserito un numero,
- Viene inserita una lunghezza per la prossima parola maggiore della lunghezza massima inserita precedentemente, negativa o non viene inserito un numero,
- Vengono inseriti simboli diversi da quelli dell'alfabeto stabilito.

Di seguito, vengono riportati invece i test significativi:

Test 1 (entrambi i linguaggi sono vuoti)

Linguaggio 1 (n° parole: 0)**:** Il primo linguaggio è vuoto.
Linguaggio 2 (n° parole: 0)**:** Il secondo linguaggio è vuoto.
Lunghezza massima: Entrambi i linguaggi sono vuoti, non è possibile eseguire nessun inserimento.
Unione: ∅
Differenza: ∅

Test 2 (il secondo linguaggio è vuoto)

Lunghezza massima: 1
Linguaggio 1 (n° parole: 1)**:** { a }
Linguaggio 2 (n° parole: 0)**:** Il secondo linguaggio è vuoto.
Unione: { a }
Differenza: { a }

Test 3 (il primo linguaggio è vuoto)

Lunghezza massima: 1
Linguaggio 1 (n° parole: 0)**:** Il primo linguaggio è vuoto.
Linguaggio 2 (n° parole: 1)**:** { a }
Unione: { a }
Differenza: ∅

Test 4 (2 parole per linguaggio, di cui una è un doppione)

Lunghezza massima: 2
Linguaggio 1 (n° parole: 2)**:** { aa, ea }
Linguaggio 2 (n° parole: 2)**:** { aa, ue }
Unione: { aa, ea, ue }
Differenza: { ea }

Test 5 (3 parole per linguaggio, di cui una è un doppione)

Lunghezza massima: 3
Linguaggio 1 (n° parole: 3)**:** { aei, *sequenza vuota*, uu }
Linguaggio 2 (n° parole: 3)**:** { uu, a, ee }
Unione: { aei, *sequenza vuota*, uu, a, ee }
Differenza: { aei, *sequenza vuota* }

Test 6 (sequenza vuota in entrambi i linguaggi)

Lunghezza massima: 3
Linguaggio 1 (n° parole: 3)**:** { aei, *sequenza vuota*, uu }
Linguaggio 2 (n° parole: 3)**:** { uu, *sequenza vuota*, ee }
Unione: { aei, *sequenza vuota*, uu, ee }
Differenza: { aei }

Test 7 (linguaggi con numero diverso di parole)

Lunghezza massima: 1
Linguaggio 1 (n° parole: 5)**:** { a, e, i, o, u }
Linguaggio 2 (n° parole: 3)**:** { *sequenza vuota*, a, e }
Unione: { a, e, i, o, u, *sequenza vuota* }
Differenza: { i, o, u }

Test 8

Lunghezza massima: 5
Linguaggio 1 (n° parole: 4)**:** { aaeei, uii, o, aiuoa }
Linguaggio 2 (n° parole: 2)**:** { uii, o }
Unione: { aaeei, uii, o, aiuoa }
Differenza: { aaeei, aiuoa }

Test 9

Lunghezza massima: 8
Linguaggio 1 (n° parole: 7)**:** { uioaeioa, aaa, uu, o, oioi, aaaaa, *sequenza vuota* }
Linguaggio 2 (n° parole: 4)**:** {uu, aaa, i, *sequenza vuota* }
Unione: { uioaeioa, aaa, uu, o, oioi, aaaaa, *sequenza vuota*, i }
Differenza: { uioaeioa, o, oioi, aaaaa }

Test 10

Lunghezza massima: 3
Linguaggio 1 (n° parole: 3)**:** { aaa, eee, iii }
Linguaggio 2 (n° parole: 3)**:** { ooo, uuu, *sequenza vuota* }
Unione: { aaa, eee, iii, ooo, uuu, *sequenza vuota* }
Differenza: { aaa, eee, iii }

6 – Verifica del programma

Brano di codice scelto:

```
void unione_linguaggi(int num_parole1, /* input: numero parole del primo linguaggio */
                      int num_parole2, /* input: numero parole del secondo linguaggio */
                      int    i,   /* input: contatore linguaggio1 e linguaggio unione */
                      int    j,            /* input: contatore linguaggio2 */
                      char** linguaggio1,  /* input: primo linguaggio */
                      char** linguaggio2,  /* input: secondo linguaggio */
                      char** unione)       /* output: unione dei linguaggi */
{
    /* prima fase dell'unione: vengono caricate tutte le parole del primo linguaggio */
    if (num_parole1 != 0)
    {
     strcpy(unione[i], linguaggio1[i]);
     unione_linguaggi((num_parole1 - 1),
                      num_parole2,
                      (i + 1),
                      j,
                     linguaggio1,
                      linguaggio2,
                      unione);
    }
     /* seconda fase dell'unione: viene eseguita l'unione dei due linguaggi */
     /* se il secondo insieme non ha più elementi, l'unione termina */
     else if (num_parole2 != 0)
     {
      strcpy(unione[i], linguaggio2[j]);
      unione_linguaggi(num_parole1,
                       (num_parole2 - 1),
                       (i + 1),
                       (j + 1),
                       linguaggio1,
                      linguaggio2,
                       unione);
    }
}
```

Proprietà da verificare:

Se consideriamo:

- `strcpy(s1, s2)` la funzione che copia il contenuto del secondo argomento nel primo, quindi la funzione che esegue l'unione;
- `unione*` è l'insieme `unione` più il primo elemento di `linguaggio2`;
- `linguaggio1'` è l'insieme `linguaggio1` meno il suo primo elemento;
- `linguaggio2'` è l'insieme `linguaggio2` meno il suo primo elemento.

La proprietà che si vuole dimostrare è che la funzione calcoli l'unione dei due linguaggi:

$$\texttt{unione} = \texttt{linguaggio1} \cup \texttt{linguaggio2}$$

Svolgimento:

Poiché la funzione è ricorsiva, dimostriamo la validità della proprietà procedendo per induzione con n, m ∈ N - rispettivamente `num_parole1` e `num_parole2` - senza necessità di ricorrere alle triple di Hoare:

- Se **n = 0** e **m = 0**, cioè se entrambi i linguaggi sono vuoti:

  `linguaggio1` ∪ `linguaggio2` = Ø ∪ Ø = Ø = `unione`

- Se **n > 0** e **m = 0**, cioè se il secondo linguaggio è vuoto:

  `linguaggio1` ∪ `linguaggio2` = `linguaggio1` ∪ Ø = `linguaggio1` = `unione`

  Poiché l'unione gode della proprietà commutativa, questa dimostrazione racchiude anche il caso in cui **n = 0** e **m > 0**, ovvero se il primo linguaggio è vuoto.

- Se **n > 0** e **m > 0**, abbiamo il caso generale:

  Supponiamo che la proprietà sia vera per **n-1** e **m-1** (**ipotesi induttiva**) e che le parole di `linguaggio1` siano state caricate in `unione`, allora:

  `linguaggio1` ∪ `linguaggio2` = `unione` ∪ `linguaggio2` →
  `strcpy(unione[i], linguaggio2[j])` →
  `unione*` ∪ `linguaggio2'` =
  `unione` ∪ `unione[i + 1]` ∪ `linguaggio2'` =
  `unione'` ∪ `unione[i + 1]` ∪ `unione[i]` ∪ `linguaggio2'`.

  In virtù del principio di induzione, possiamo dunque concludere che la funzione considerata calcola l'unione dei due linguaggi.