1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

Progetto di Programmazione Logica e Funzionale

Sessione invernale 2023/2024

Professore Marco Bernardo

Giulia Costa
N. Matricola: 308558

Bogdan Dragne
N. Matricola: 308193

Indice

1 Specifica del Problema

Scrivere un programma Haskell e un programma Prolog che realizzi il seguente gioco tra due giocatori:

- Nella prima fase, il programma chiede al primo giocatore di inserire un certo numero di parole (in italiano). Per la precisione, il programma inizia chiedendo al primo giocatore il numero di parole che vuole inserire (almeno 10), poi le acquisisce una alla volta verificando che ciascuna sia formata solo dalle 26 lettere minuscole dell'alfabeto (ogni parola errata va introdotta di nuovo finché non è corretta).

- Nella seconda fase, il programma sceglie a caso una parola tra quelle inserite dal primo giocatore, poi chiede al secondo giocatore di indovinarla mettendogli a disposizione un numero di tentativi pari al numero di lettere che costituiscono la parola. La parola deve inizialmente essere visualizzata come una sequenza di asterischi, che vengono poi tramutati nelle corrispondenti lettere man mano che il secondo giocatore le indovina. Una volta indovinata la parola oppure esauriti i tentativi a disposizione, il programma chiede al secondo giocatore se vuole indovinare un'altra parola.

2 Analisi del Problema

2.1 Dati di Ingresso del Problema

Possiamo dividere i dati di ingresso del problema in due fasi rispettivamente fase 1 e fase 2.

- **Fase 1:** numero intero che rappresenta la lunghezza della lista di parole e un elenco di parole ciascuna formata dalle 26 lettere minuscole dell'alfabeto.

- **Fase 2:** parola appartenente alle 26 lettere minuscole dell'alfabeto e una risposta (s/n) per determinare se si desidera indovinare un'altra parola.

2.2 Dati di Uscita del Problema

Il dato d'uscita consiste nella rappresentazione di una maschera che identifica la parola da indovinare, aggiornata ogni qualvolta, l'utente inserendo una parola, indovini le lettere nelle posizioni corrispondenti.

2.3 Relazioni Intercorrenti

L'alfabeto italiano è costituito da 26 lettere minuscole che possiamo formalizzare:
A = $\{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$

Una parola w è una sequenza di caratteri: $w = c_1$,c_2, ...,c_n.
La lunghezza di una stringa la possiamo indicare con $|w|$ secondo la teoria dei linguaggi formali.
Poichè nella lingua italiana esistono anche parole costituite da una sola unità caratteriale, vale la seguente relazione $|w| \geq 1$.

3 Progettazione dell'Algoritmo

3.1 Scelte di Progetto

L'insieme delle parole acquisite, viene rappresentato tramite una struttura dati lineare, ossia una lista.
Le motivazioni sono le seguenti:

- **Pattern Matching:** le liste diventano uno strumento potente grazie all'utilizzo di tale tecnica di lavoro. La loro natura consente una destrutturazione agevole e una manipolazione efficiente, facilitando l'estrazione di informazioni specifiche.

- **Ricorsione:** le liste costituiscono una struttura dati adatta per l'implementazione di soluzioni a problemi ricorsivi.

Nel contesto dell'inserimento delle parole da parte del primo giocatore, è stata imposta la restrizione per cui non è consentito l'inserimento di parole già presenti nella lista.

La maschera di asterischi, utilizzata per nascondere la parola all'utente, è stata trattata come una lista.
Durante la "guessing procedure" i tentativi per indovinare la parola vengono decrementati solo se la parola inserita è valida. Infatti, solo nel caso in cui il secondo giocatore inserisca una parola valida ma di lunghezza diversa rispetto a quella richiesta, il numero di tentativi viene decrementato.

La parola estratta viene quindi eliminata dalla lista delle parole:

- se la lista è vuota, il gioco termina;

- se lista non è vuota viene proposto al giocatore di continuare il gioco.

3.2 Passi dell'Algoritmo

La dinamica del gioco coinvolge due partecipanti, con la prima fase assegnata al giocatore 1 e la seconda fase riservata al giocatore 2.

GIOCATORE 1:

1. Inserimento del numero di parole.
   Il primo giocatore inserisce il numero desiderato di parole da utilizzare nel gioco.

   - Validazione dell'input: l'algoritmo verifica che l'input sia un numero maggiore o uguale a 10. In caso contrario si richiede nuovamente l'inserimento.

2. Acquisizione delle singole parole.
   Le parole inserite vengono validate individualmente, trattate come liste di caratteri.

   - Validazione dell'input: ogni parola deve avere lunghezza maggiore o uguale a 1, deve essere composta da caratteri alfabetici minuscoli e non contenere spazi vuoti. In caso di errore, si richiede nuovamente l'inserimento. Ogni parola corretta viene inserita nella lista delle parole.
   - Caso base: non ci sono più parole da inserire nella lista.
   - Caso generale: si prosegue fino a quando ci sono ancora parole da validare. Viene inoltre riproposto all'utente l'inserimento della parola, ogni qualvolta l'input non fosse corretto. Ogni parola correttamente validata viene poi aggiunta alla lista, decrementando il numero di prarole da processare. In caso di inserimento di una parola già presente, viene mostrato un messaggio di errore e richiesto un nuovo input.

GIOCATORE 2:

1. Selezione di una parola random.
   Viene selezionata casualmente una parola dalla lista delle parole generando un indice casuale nell'intervallo [1, lunghezza lista].

2. Inizializzazione della maschera.
   Una lista di asterischi, viene mostrata al secondo giocatore per indovinare la parola nascosta.

   - Caso base: restituisce una lista vuota.
   - Caso generale: crea una lista di asterischi, per nascondere al secondo giocatore la parola da indovinare, utilizzando la ricorsione aggiungendo un asterisco alla volta fino a raggiungere la lunghezza della parola.

3. Acquisizione della parola.
   La parola viene validata allo stesso modo del giocatore 1. Vengono gestiti i tentativi disponibili per indovinare la parola e si confronta la parola inserita con quella da indovinare.

   - Gestione dei tentativi: inizialmente i tentativi vengono inizializzati pari alla lunghezza della parola da indovinare e ad ogni inserimento validato di lunghezza diversa, questi vengono decrementati.
   - Caso base: restituisce la lista aggiornata quando vengono processati tutti gli elementi delle liste che sono passate in input.
   - Caso generale: viene confrontata ciascuna lettera della parola inserita con quella della parola casuale da indovinare. Se una lettera è corretta, viene inclusa nella nuova lista risultante; altrimenti viene inclusa nella nuova lista la lettera corrispondente della maschera fornita.

4. Rimozione dalla lista.
   Se la parola viene indovinata o terminano i tentativi a disposizione, essa viene rimossa dalla lista. Nel caso di lista vuota il gioco termina (non viene incluso in questo caso il punto 5).

5. Proposta di prosecuzione.
   Al giocatore 2 viene proposto di terminare la sessione di gioco o procedere con una nuova parola. L'input è validato accettando solo i valori 's/n'.

   - Caso di risposta affermativa: il turno del secondo giocatore ricomincia.
   - Caso di risposta negativa: il gioco termina.

4 Implementazione dell'Algoritmo

File sorgente indovina_la_parola.hs:

```
{- Programma Haskell che implementa un gioco per indovinare una parola
   data una lista di parole tra due giocatori. -}

import Text.Read (readMaybe)        -- necessario per verificare se è stato inserito un numero
import Data.List (nub)              -- necessario per rimuovere i duplicati dalla lista
import Data.Char (isLower,isLetter) -- necessario per stabilire se la lettera inserita è minuscola
                                    -- e se il secondo giocatore ha inserito un carattere
import System.Random(randomRIO)     -- necessario per estrarre l'indice casuale
                                    -- nell'intervallo della lista di parole

main :: IO ()
main = do
    putStrLn "---------------- GIOCATORE 1"
    listWords <- first_player
    putStrLn " ---------------- GIOCATORE 2"
    second_player listWords

{- La funzione first_player acquisisce il numero di parole -}
first_player :: IO [String]
first_player = do
  n <- validate_first_input
  listWords <- read_input_words n []
  return listWords

{- La funzione second_player ha come argomento la lista di parole:
   - Estrae dalla lista delle parole una parola randomica
   - La parola estratta viene tramutata in asterischi
   - Viene richiesto se si vuole indovinare un'altra parola -}
second_player :: [String] -> IO ()
second_player listWord = do
    randomStr <- random_word listWord
    maskWord <- init_mask_word randomStr
    let randomWordLength = length randomStr
    check_word randomStr maskWord randomWordLength
    let newList = remove_guess_word randomStr listWord
    continue <- try_again newList
    result newList continue

{- La funzione result ha come primo argomento la lista di parole e
   come secondo argomento la scelta s/n del secondo giocatore.
   Viene verificato se la lista è vuota e in tal caso il gioco termina
   se la lista non è vuota e il giocatore vuole proseguire il gioco continua
   se la lista non è vuota e il giocatore NON vuole proseguire il gioco termina -}
result :: [String] -> Bool -> IO ()
result newList continue
  | is_list_empty newList = putStrLn "Non ci sono più parole da indovinare!"
  | continue              = do
                            second_player newList
  | otherwise             = putStrLn "GAME OVER!"

{- La funzione is_list_empty ha come argomento la lista di parole
   e verifica se la lista è vuota -}
```

```
is_list_empty :: [a] -> Bool
is_list_empty [] = True
is_list_empty _  = False

{- La funzione remove_guess_word rimuove la parola dalla lista delle parole
   primo argomento: parola estratta
   secondo argomento: lista di parole -}
remove_guess_word :: Eq a => a -> [a] -> [a]
remove_guess_word x ys = filter (/= x) ys

{- La funzione try_again chiede se si vuole indovinare un'altra parola
   Nel caso in cui la lista delle parole risulti vuota non viene richiesto
   il messaggio -}
try_again :: [String] -> IO Bool
try_again str = do

  if is_list_empty str
    then return False
  else
    do
    putStrLn "Vuoi indovinare un'altra parola? (s/n)"
    answer <- getLine
    case answer of
      ['s'] -> return True
      ['n'] -> return False
      _ -> do
        try_again str

{- La funzione validate_first_input chede di inserire il numero di parole
   e verifica se è stato inserito un numero intero -}
validate_first_input :: IO Int
validate_first_input = do
    putStrLn "Inserisci un numero (>= 10):"
    n <- getLine

    case readMaybe n of
        Just np -> validate_n np
        Nothing -> do
            validate_first_input

{- La funzione validate_n ha come parametro il numero di parole
   e se il numero è < 10 chiede il suo reinserimento -}
validate_n :: Int -> IO Int
validate_n n
    | n >= 10    = return n
    | otherwise  = do
        validate_first_input

{- La funzione read_input_words permette l'inserimento delle parole nella lista
   e verifica se la la parola è già presente nella lista:
   - primo argomento: numero di parole da inserire nella lista
   - secondo argomento: lista di parole -}
read_input_words :: Int -> [String] -> IO [String]
read_input_words 0 listWords = return listWords
read_input_words n listWords = do
```

```
    putStrLn $ "Inserisci la parola " ++ show (length listWords + 1) ++ ":"
    input <- validate_second_input
    newList <- remove_dup_ins input listWords
    read_input_words (n - 1) newList

{- La funzione remove_dup_ins rende in ingresso una stringa
   e una lista e controlla se è gia presente
   chiedendo l'eventuale reinserimento:
   - primo argomento: parola inserita
   - secondo argomento: lista di parole -}
remove_dup_ins :: String -> [String] -> IO [String]
remove_dup_ins str strList
    | str `elem` nub strList = do
        putStrLn "Parola già inserita."
        newStr <- validate_second_input
        remove_dup_ins newStr strList
    | otherwise = return (strList ++ [str])

{- La funzione validate_second_input ha come argomento la parola
   se la parola ha lunghezza inferiore a 2 oppure
   non contiene lettere minuscole dell'alfabeto
   viene richiesto il reinserimento -}
validate_second_input :: IO String
validate_second_input = do
    input <- getLine
    case () of
        _ | length input >= 1 && is_StringBetween_AZ input &&
                    all isLower input -> return input
        _ -> do
            putStrLn "Inserimento non valido."
            putStrLn "La parola deve contenere lettere minuscole dell'alfabeto."
            validate_second_input

{- La funzione random_word ha come parametro la lista di parole
   ed estrae un indice nell'intervallo della lista -}
random_word :: [String] -> IO String
random_word strings = do
  let lastIndex = length strings - 1
  randomIndex <- randomRIO(0, lastIndex)
  return (strings !! randomIndex)

{- La funzione init_mask_word genera asterischi pari al numero della parola estratta
   ha come parametro la parola da indovinare -}
init_mask_word :: String -> IO String
init_mask_word inputString = return (replicate (length inputString) '*')

{- La funzione check_word controlla se la parola inserita trova corrispondenze
   con la parola da indovinare
   - primo argomento: parola estratta dalla lista delle parole
   - secondo argomento: maschera costituita da asterischi
   - terzo argomento: tentativi
  Si procede in maniera ricorsiva come segue:
  Se ho terminato i tentativi la parola non è stata indovinata.
  Se la maschera aggiornata corrisponde con la parola estratta la parola è stata indovinata.
  Altrimenti viene validato l'input e se la lunghezza della parola inserita coincide con
```

```
  la lunghezza della parola da indovinare viene aggiornata la maschera decrementando
  il numero di tentativi.
  Se l'input non è valido viene richiesto l'inserimento decrementando il numero di tentativi -}
check_word :: String -> String -> Int -> IO ()
check_word randomWord maskWord attempts
  | attempts == 0 = putStrLn "Parola non indovinata, hai terminato i tentativi."
  | randomWord == maskWord = putStrLn "Parola indovinata correttamente!"
  | otherwise = do
      status maskWord attempts

      inputWord <- validate_second_input
      if length inputWord /= length randomWord
         then do
            putStrLn "Inserimento non valido."
            putStrLn "La parola inserita ha lunghezza diversa rispetto a quella da indovinare"
            check_word randomWord maskWord (attempts - 1)
         else do
            let nuovaMaskWord = compare_words randomWord maskWord inputWord
            check_word randomWord nuovaMaskWord (attempts - 1)

{- La funzione compare_words aggiorna la maschera in base alla parola da indovinare
   e restituisce la maschera aggiornata.
   primo argomento: parola da indovinare
   secondo argomento: la maschera
   terzo argomento: la nuova maschera aggiornata -}
compare_words :: String -> String -> String -> String
compare_words [] _ _ = ""
compare_words (p : parola) (m : maskWord) (p2 : parola2)
  | m == '*' && p2 == p = p : compare_words parola maskWord parola2
  | otherwise = m : compare_words parola maskWord parola2

{- La funzione status stampa il numero di tentativi a disposizione e la maschera:
   primo argomento: maschera
   secondo argomento: numero di tentativi -}
status :: String -> Int -> IO ()
status p t = do
    let lengthWord = length p
    putStrLn $ "Hai a disposizione " ++ show t ++ " tentativi"
    putStrLn $ "Inserisci una parola lunga " ++ show lengthWord ++ ": "
    putStrLn p

{- La funzione is_StringBetween_AZ controlla se la stringa contiene
   lettere minuscole -}
is_StringBetween_AZ :: String -> Bool
is_StringBetween_AZ str = all (\c -> c >= 'a' && c <= 'z') str
```

File sorgente indovina_la_parola.pl:

```
/* Programma Prolog che implementa un gioco per indovinare una parola data una lista
   di parole tra due giocatori. */

 main :-
  write('---------------- GIOCATORE 1'), nl,
  first_player(ListWords),
  nl, write('--------------- GIOCATORE 2'),
  nl, write('Indovina la parola:'), nl,
  second_player(ListWords).

  /*********************************************************************************/
  /******************************** PRIMO GIOCATORE ********************************/
  /*********************************************************************************/
  /* Il predicato viene utilizzato per richiamare le funzionalità del giocatore 1.
     Viene passato un solo argomento (ListWords), utilizzato per restituire la lista
     delle parole che il secondo giocatore dovrà indovinare. */
  first_player(ListWords) :-
    repeat,
    (validate_first_input(N)),
    read_input_words(N, [], ListWords, 1).

  /* Il predicato acquisisce e valida il numero di parole inserito (input >= 10).
     In caso di validazione errata si richiede un nuovo inserimento.
     Viene passato un solo argomento (N), utilizzato per restituire il numero di
     parole effettive che il primo giocatore vuole inserire. */
   validate_first_input(N) :-
    (
      nl, write('Inserisci un numero (>= 10):'), nl,
      catch(read(N), _, fail),
      (
        number(N) ->
        (
          integer(N),
          N >= 10 ->
            !
        ;
          fail
        )
      )
    ).

  /* Il predicato acquisisce l'input, validandolo tramite 'validate_second_input',
     e in caso positivo lo aggiunge alla lista delle parole.
     I suoi argomenti sono:
     - N, che indica il numero di parole da inserire nella lista;
     - ListInit, l'insieme da comporre, nella quale vengono aggiunte le parole validate;
     - ListWords, l'insieme risultante delle parole;
     - Index, un indice utilizzato nella stampa a video, incrementato ad ogni input accettato.*/
  read_input_words(0, ListInit, ListInit, _).
  read_input_words(N, ListInit, ListWords, Index) :-
    N > 0,
     (
        nl, format('Inserisci la parola ~w: ~n', [Index]),
```

```
      catch(read(Input), _, fail),
      validate_input_words(Input) ->
        (
          member(Input, ListInit) ->
            nl, write('Parola gia'' inserita.'), nl,
            read_input_words(N, ListInit, ListWords, Index)
        ;
            Index1 is Index + 1,
            N1 is N - 1,
            append(ListInit, [Input], UpdatedList),
            read_input_words(N1, UpdatedList, ListWords, Index1)
        )
   ;
      (
          % Se la validazione fallisce, gestione dell'input
          nl, write('Inserimento non valido.'),
          nl, write('La parola deve contenere lettere minuscole dell''alfabeto.'), nl,
          read_input_words(N, ListInit, ListWords, Index)
      )
   ).

/*****************************************************************************/
/****************************** SECONDO GIOCATORE ****************************/
/*****************************************************************************/
/* Il predicato determina le mosse del secondo giocatore.
   Richiama i predicati per l'estrazione casuale di una parola dalla lista delle
   parole, inizializza la maschera e avvia la "guessing procedure".
   Argomenti in input: lista delle parole (ListWords). */
second_player(ListWords) :-
  random_word(ListWords, RandomWord),
  atom_length(RandomWord, RandomWordLength),
  init_mask_word(RandomWordLength, MaskWord),
  check_word(ListWords, RandomWord, MaskWord, RandomWordLength).

/* Il predicato seleziona causalmente una parola per la "guessing procedure".
   Argomenti:
   - ListWords, lista delle parole dalla quale estrarre la parola casuale;
   - RandomWord, parola casuale estratta.
   L'uso di random consente di generare un indice casuale,
   mentre nth0 di estrarre dalla lista la parola corrispondente all'indice. */
random_word(ListWords, RandomWord) :-
  length(ListWords, ListLenght),
  random(0, ListLenght, Index),
  nth0(Index, ListWords, RandomWord).

/* Il predicato inizializza una maschera di asterischi,
   per nascondere al secondo giocatore la parola da indovinare.
    Argomenti:
    - N, la lunghezza della parola da indovinare;
    - la maschera di asterischi, rappresentata da una lista.*/
init_mask_word(0, []).
init_mask_word(N, ['*' | Resto]) :-
  N > 0,
  N1 is N - 1,
  init_mask_word(N1, Resto).
```

```
/* Il predicato è utilizzato per la "guessed procedure".
   Stampa ogni volta la maschera aggiornata, verifica la disponibilità di tentativi
   rimanenti per indovinare la parola, e gestisce l'input dell'utente, richiamando
   i predicati 'try_again/1' (nel caso di tentativi terminati)
   o 'handle_input/5' (per la gestione dell'input validato).
   Nel caso in cui i tentativi siano terminati, rimuove la parola non indovinata
   dalla lista delle parole, restituendo una lista aggiornata.
   Nel caso in cui la maschera non contenga più asterischi, si chiama il predicato
   'guessed_word/4' per la gestione della parola indovinata.
   Argomenti:
   - ListWords, lista delle parole, passata per eliminare la parola se terminati i
     tentativi o indovinata la parola;
   - RandomWord, parola randoma da indovinare;
   - MaskWord, maschera di asterischi da aggiornare;
   - Attempts, numero di tentativi a disposizione.*/
check_word(ListWords, RandomWord, MaskWord, Attempts) :-
  list_to_string(MaskWord, ListToString),
  (
    Attempts =:= 0 ->
      nl, write('Parola non indovinata, hai terminato i tentativi.'), nl,
      remove_guess_word(ListWords, RandomWord, NewLists),
      length(NewLists, Length),
      (
        Length > 0 ->
          try_again(NewLists)
      ;
          nl, write('Non ci sono piu'' parole da indovinare!'),
          true
        )
  ;
      % Se ci sono ancora tentativi si procede con il controllo della maschera.
      (
        member('*', MaskWord) ->
          handle_input(ListWords, RandomWord, MaskWord, Attempts, ListToString)
      ;
          guessed_word(RandomWord, ListToString, ListWords)
      )
  ).

/* Predicato ausiliario per gestire l'input dell'utente, se valido.
   Nel caso in cui l'input sia valido, viene verificata la lunghezza della parola.
   Se la lunghezza dell'input è diversa da quella richiesta, il numero di tentativi
   viene decrementato.
   Argomenti:
   - ListWords, lista delle parole;
   - RandomWord, parola random;
   - MaskWord, maschera da aggiornare;
   - Attempts, numero di tentavi a disposizione,
   - ListToString, lista della maschera aggiornata, passata per la gestione dello status.*/
handle_input(ListWords, RandomWord, MaskWord, Attempts, ListToString) :-
  (
    status(Attempts, ListToString),
    catch(read(Input), _, fail),
    validate_input_words(Input) ->
```

```
      atom_chars(Input, ListInput),
      length(ListInput, InputLength),
      atom_chars(RandomWord, ListRandomWord),
      length(ListRandomWord, RandomWordLength),
        (

          InputLength =:= RandomWordLength ->
            compare_words(ListInput, ListRandomWord, MaskWord, [], UpdateMaskWord),
            NewAttempts is Attempts - 1,
            check_word(ListWords, RandomWord, UpdateMaskWord, NewAttempts)
        ;
            nl, write('Inserimento non valido.'),
            nl, write('La parola inserita ha lunghezza diversa rispetto a quella da indovinare.'), nl,
            NewAttempts is Attempts - 1,
            check_word(ListWords, RandomWord, MaskWord, NewAttempts)
        )
  ;
      nl, write('Inserimento non valido.'),
      nl, write('La parola deve contenere lettere minuscole dell''alfabeto.'), nl,
      handle_input(ListWords, RandomWord, MaskWord, Attempts, ListToString)
  ).


/* Il predicato converte una lista di caratteri in una lista di stringhe.
   CASO BASE: restituisce la lista aggiornata se la lista di caratteri è vuota.
   CASO GENERALE: genera una lista di caratteri, utilizzando la ricorsione aggiungendo
   un carattere alla volta, preso dalla lista passatagli.
   La lista Update viene utilizzata come 'accumulatore', nella quale aggiungere caratteri,
   grazie all'utilizzo di 'append', reiterando fino a che la maschera che gli viene
   passata è vuota. Viene restituita la lista aggiornata "UpdateList".
   Argomenti:
   - MaskWord, maschera da stampare;
   - UpdateList, lista aggiornata.
   - Update, lista da aggiornare. */
list_to_string(MaskWord, UpdateList) :-
  list_to_string(MaskWord, [], StringList),
  atom_chars(UpdateList, StringList).
list_to_string([], Update, Update).
list_to_string([Char|Tail], Update, UpdateList) :-
  append(Update, [Char], NewUpdate),
  list_to_string(Tail, NewUpdate, UpdateList).

/* Il redicato serve per visualizzare lo stato del gioco.
   Argomenti:
   - Attempts, numero di tentativi a disposizione;
   - ListToString, maschera aggiornata trasformata in lista. */
status(Attempts, ListToString) :-
  write(ListToString), nl,
  atom_length(ListToString, LenghList),
  nl, format('Hai a disposizione ~w tentativi.', [Attempts]),
  nl, format('Inserisci una parola lunga ~w: ', [LenghList]).

/* Il predicato mette a confronto ogni lettera della parola in input con quella
   da indovinare, andando ad aggiornare in tal modo la maschera da mostrare all'utente.
   Durante questo confronto, se una lettera è corretta, viene inclusa nella nuova
```

```
   lista risultante.
   Altrimenti, viene inclusa nella nuova
   lista la lettera corrispondente della maschera fornita.
   Argomenti:
   - Il primo input rappresenta la lista dei caratteri che formano la parola in input dell'utente;
   - Il secondo input rappresenta la lista dei caratteri che formano la parola random;
   - Il terzo input rappresenta la lista dei caratteri che formano la maschera attuale;
   - Update rappresenta la maschera di appoggio da aggiornare;
   - UpdateMask rappresenta la maschera risultante aggiornata. */
compare_words([], [], [], Update, Update).
compare_words([X | RestInput], [Y | RestRandomWord], [Mask | RestMask], Update, UpdateMask) :-
  (
    X \= Y ->
      append(Update, [Mask], NewUpdate)
  ;
      append(Update, [X], NewUpdate)
  ),
  compare_words(RestInput, RestRandomWord, RestMask, NewUpdate, UpdateMask).


/* Il predicato controlla che venga indovinata la parola, rimuovendola dalla lista.
   Controlla inoltre se la lista è vuota terminando il programma,
   o in caso alternativo chiama il predicato try_again.
   Argomenti:
   - RandomWord, parola random da confrontare con la parola creata;
   - InputWord, che rappresenta la parola creata dalle varie lettere indovinate;
   - ListWords, lista delle parole, per rimuovere la parola. */
guessed_word(RandomWord, InputWord, ListWords) :-
  (
    InputWord == RandomWord ->
      write(InputWord), nl,
      nl, write('Parola indovinata correttamente!'), nl,
      remove_guess_word(ListWords, RandomWord, NewLists),
      length(NewLists, Length),
      (
        Length > 0 ->
          try_again(NewLists)
      ;
          nl, write('Non ci sono piu'' parole da indovinare!'),
          true
        )
  ).

/* Predicato usato per rimuovere la parola indovinata dalla lista.
   Argomenti:
   - ListWords, lista delle parole dalla quale rimuovere la parola.
   - RandomWord, parola random da eliminare dalla lista;
   - NewListWords, lista aggiornata dopo la rimozione. */
remove_guess_word(ListWords, RandomWord, NewListWords) :-
  select(RandomWord, ListWords, NewListWords).

/* Predicato utilizzato per gestire la risposta dell'utente alla richiesta
   di indovinare una nuova parola nel caso in cui abbia indovinato la precedente.
   - Nel caso di risposta positiva viene richiamato il predicato utilizzato per
     l'intera routine che parte dalla scelta della parola casuale in poi.
```

```
   - Nel caso di risposta negativa il programma termina con 'True'.
   - Nel caso di risposta non valida, viene riproposta la domanda.
   L'unico argomento è ListWords, ovvero la lista delle parole aggiornata,
   che viene passata come argomento per fare in modo che ricominci il turno
   del secondo giocatore in caso di risposta positiva. */
try_again(ListWords) :-
  nl, write('Vuoi indovinare un''altra parola? (s/n)'), nl,
  read(Response),
  (
    Response == 's' ->
      second_player(ListWords)
  ;
    Response == 'n' ->
      nl, write('GAME OVER!'), nl,
      true
  ;
    write('Inserimento non valido. Riprova.'), nl,
    try_again(ListWords)
  ).

/* Predicato utilizzato per validare l'input.
   L'input deve essere una lettera minuscola appartenente all'alfabeto.
   Viene controllato quindi che non sia un numero, che non sia composto da lettere
   maiuscole e che appartenga alle lettere dell'alfabeto
   (e quindi non sia un simbolo o segno di punteggiatura). */
validate_input_words(Input) :-
  (
    maplist(is_upper, Input) ->
      false
  ;
    number(Input) ->
      false
  ;
      atom_chars(Input, Chars),
      length(Chars, InputLength),
      (
        InputLength > 0 ->
          maplist(is_alpha_character, Chars)
      )
  ).

/* Predicato utilizzato per verificare che ogni carattere della parola sia composta
   da lettere minuscole dell'alfabeto, validando l'input in ingresso (Char). */
is_alpha_character(Char) :-
  Char @>= 'a', Char @=< 'z'.

/* Il predicato verifica la presenza di lettere maiuscole in ogni carattere
   dell'input che gli viene passato (Char). */
is_upper(Char) :-
  Char @>= 'A', Char @=< 'Z'.
```

5 Testing del Programma

5.1 Testing del programma Haskell

Test Haskell 1
Validazione del primo input per il giocatore 1. Il valore numerico deve essere un numero intero ≥ 10.

– – – – – – – – – – – – – – – – – GIOCATORE 1
Inserisci un numero (≥ 10):
8.
Inserisci un numero (≥ 10):
a.
Inserisci un numero (≥ 10):
!.
Inserisci un numero (≥ 10):
8à.
Inserisci un numero (≥ 10):
10.

Inserisci la parola 1:

Test Haskell 2
Inserimento di un input > 10.

– – – – – – – – – – – – – – – – – – GIOCATORE 1
Inserisci un numero (≥ 10):
12.

Inserisci la parola 1:

Test Haskell 3
Validazione del secondo input per il giocatore 1.
L'input deve essere una parola di lunghezza > 0, composta da lettere minuscole dell'alfabeto.

Inserisci la parola 1:
0
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.

Inserisci la parola 1:
e

Inserisci la parola 2:
o8
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.

Inserisci la parola 2:
clichè
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.

Inserisci la parola 2:
casa mia
Inserimento non valido.

La parola deve contenere lettere minuscole dell'alfabeto.

Inserisci la parola 2:
Finestra
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.

Inserisci la parola 2:
finestra

Inserisci la parola 3:
casa

Test Haskell 4

Inserimento di una parola già inserita precedentemente.

Inserisci la parola 3:
farfalla

Inserisci la parola 4:
farfalla
Parola gia' inserita.

Inserisci la parola 4:
fiore

Test Haskell 5
Validazione delle lettere inserite dal giocatore 2 per indovinare la parola nascosta: l'input deve essere una parola composta da lettere minuscole dell'alfabeto.
I tentativi vengono decrementati solo se l'input viene validato.
La parola nascosta non viene indovinata, perchè il giocatore ha terminato i tentativi a disposizione.

– – – – – – – – – – – – – – – – GIOCATORE 2
Indovina la parola:
****

Hai a disposizione 7 tentativi.
Inserisci una parola lunga 7: 99
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.
****

Hai a disposizione 7 tentativi.
Inserisci una parola lunga 7: ventitrè
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.
****

Hai a disposizione 7 tentativi.
Inserisci una parola lunga 7: prova t
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.
****

Hai a disposizione 7 tentativi.
Inserisci una parola lunga 7: giochi?
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.
****

Hai a disposizione 7 tentativi.
Inserisci una parola lunga 7: palla
Inserimento non valido.
La parola inserita ha lunghezza diversa rispetto a quella da indovinare.
****

Hai a disposizione 6 tentativi.
Inserisci una parola lunga 7: farfall.
Inserimento non valido.
La parola inserita ha lunghezza diversa rispetto a quella da indovinare.
****

Hai a disposizione 5 tentativi.
Inserisci una parola lunga 7: stazione
Inserimento non valido.
La parola inserita ha lunghezza diversa rispetto a quella da indovinare.
******

Hai a disposizione 4 tentativi.
Inserisci una parola lunga 7: bancone
*a*****

Hai a disposizione 3 tentativi.
Inserisci una parola lunga 7: sambuco
*a*****

Hai a disposizione 2 tentativi.
Inserisci una parola lunga 7: tappeto
*a**e**

Hai a disposizione 1 tentativo.
Inserisci una parola lunga 7: rosario

Parola non indovinata, hai terminato i tentativi.

Vuoi indovinare un'altra parola? (s/n)

Test Haskell 6
Il giocatore 2 ha indovinato la parola.

Indovina la parola:
*****

Hai a disposizione 5 tentativi.
Inserisci una parola lunga 5: sacca
****a

Hai a disposizione 4 tentativi.
Inserisci una parola lunga 5: panna

p*nna

Hai a disposizione 3 tentativi.
Inserisci una parola lunga 5: penna
penna

Parola indovinata correttamente!

Vuoi indovinare un'altra parola? (s/n)

Test Haskell 7
Il giocatore ha indovinato l'ultima parola della lista. La lista risulta vuota.

Indovina la parola:
****

Hai a disposizione 8 tentativi.
Inserisci una parola lunga 8: gazzella
*a***lla

Hai a disposizione 7 tentativi.
Inserisci una parola lunga 8: forcella
far**lla

Hai a disposizione 6 tentativi.
Inserisci una parola lunga 8: farfalla
farfalla

Parola indovinata correttamente!

Non ci sono piu' parole da indovinare!

Test Haskell 8
Il giocatore non ha indovinato l'ultima parola della lista. La lista risulta vuota.

Indovina la parola:
****

Hai a disposizione 4 tentativi.
Inserisci una parola lunga 4: diga
***a

Hai a disposizione 3 tentativi.
Inserisci una parola lunga 4: lira
***a

Hai a disposizione 2 tentativi.
Inserisci una parola lunga 4: uova
***a

Hai a disposizione 1 tentativo.
Inserisci una parola lunga 4: tata
*a*a

Parola non indovinata, hai terminato i tentativi.

Non ci sono piu' parole da indovinare!

Test Haskell 9
Proposta di prosecuzione, con validazione dell'input ed esito positivo.

Hai a disposizione 2 tentativi.
Inserisci una parola lunga 5: nonna
*a**a

Hai a disposizione 1 tentativo.
Inserisci una parola lunga 5: sacca

Parola non indovinata, hai terminato i tentativi.

Vuoi indovinare un'altra parola? (s/n)
S
Inserimento non valido. Riprova.

Vuoi indovinare un'altra parola? (s/n)
a
Inserimento non valido. Riprova.

Vuoi indovinare un'altra parola? (s/n)
8
Inserimento non valido. Riprova.

Vuoi indovinare un'altra parola? (s/n)
N
Inserimento non valido. Riprova.

Vuoi indovinare un'altra parola? (s/n)
s

Indovina la parola:
****

Hai a disposizione 4 tentativi.
Inserisci una parola lunga 4: sole
*o**

Hai a disposizione 3 tentativi.
Inserisci una parola lunga 4: noto
*o*o

Hai a disposizione 2 tentativi.
Inserisci una parola lunga 4: foto
foto

Parola indovinata correttamente!

Vuoi indovinare un'altra parola? (s/n)

Test Haskell 10
Proposta di prosecuzione, con esito negativo e terminazione del gioco.

Indovina la parola:
*****

Hai a disposizione 5 tentativi.
Inserisci una parola lunga 5: cuore.
**o*e

Hai a disposizione 4 tentativi.
Inserisci una parola lunga 5: torre.
**ore

Hai a disposizione 3 tentativi.
Inserisci una parola lunga 4: fiore.
fiore

Parola indovinata correttamente!

Vuoi indovinare un'altra parola? (s/n)
n.

GAME OVER!

5.2 Testing del programma Prolog

Test Prolog 1
Validazione del primo input per il giocatore 1. Il valore numerico deve essere un numero ≥ 10.

– – – – – – – – – – – – – – – – – GIOCATORE 1
Inserisci un numero (≥ 10):
8.
Inserisci un numero (≥ 10):
a.
Inserisci un numero (≥ 10):
!.
Inserisci un numero (≥ 10):
8à.
Inserisci un numero (≥ 10):
10.

Inserisci la parola 1:

Test Prolog 2
Inserimento di un input > 10.

– – – – – – – – – – – – – – – – – GIOCATORE 1
Inserisci un numero (≥ 10):
12.

Inserisci la parola 1:

Test Prolog 3
Validazione del secondo input per il giocatore 1.
L'input deve essere una parola di lunghezza > 0, composta da lettere minuscole dell'alfabeto.

Inserisci la parola 1:
0.
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.

Inserisci la parola 1:
e.

Inserisci la parola 2:
o8.
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.

Inserisci la parola 2:
clichè.
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.

Inserisci la parola 2:
casa mia.
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.

Inserisci la parola 2:
Finestra.
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.

Inserisci la parola 2:
finestra.

Inserisci la parola 3:
casa.

Test Prolog 4
Inserimento di una parola già inserita precedentemente.

Inserisci la parola 3:
farfalla.

Inserisci la parola 4:
farfalla.
Parola gia' inserita.

Inserisci la parola 4:
fiore.

Test Prolog 5
Validazione delle lettere inserite dal giocatore 2 per indovinare la parola nascosta: l'input deve essere una parola composta da lettere minuscole dell'alfabeto.
I tentativi vengono decrementati solo se l'input viene validato.
La parola nascosta non viene indovinata, perchè il giocatore ha terminato i tentativi a disposizione.

– – – – – – – – – – – – – – – – – GIOCATORE 2
Indovina la parola:
****

Hai a disposizione 7 tentativi.
Inserisci una parola lunga 7: 99.
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.
****

Hai a disposizione 7 tentativi.
Inserisci una parola lunga 7: ventitrè.
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.
****

Hai a disposizione 7 tentativi.
Inserisci una parola lunga 7: prova t.
Inserimento non valido.
La parola deve contenere lettere minuscole dell'alfabeto.
****

Hai a disposizione 7 tentativi.
Inserisci una parola lunga 7: giochi?.
Inserimento non valido.

La parola deve contenere lettere minuscole dell'alfabeto.
****

Hai a disposizione 7 tentativi.
Inserisci una parola lunga 7: palla.
Inserimento non valido.
La parola inserita ha lunghezza diversa rispetto a quella da indovinare.
****

Hai a disposizione 6 tentativi.
Inserisci una parola lunga 7: farfalla.
Inserimento non valido.
La parola inserita ha lunghezza diversa rispetto a quella da indovinare.
****

Hai a disposizione 5 tentativi.
Inserisci una parola lunga 7: stazione.
Inserimento non valido.
La parola inserita ha lunghezza diversa rispetto a quella da indovinare.
****

Hai a disposizione 4 tentativi.
Inserisci una parola lunga 7: bancone.
*a*****

Hai a disposizione 3 tentativi.
Inserisci una parola lunga 7: sambuco.
*a*****

Hai a disposizione 2 tentativi.
Inserisci una parola lunga 7: tappeto.
*a**e**

Hai a disposizione 1 tentativo.
Inserisci una parola lunga 7: rosario.

Parola non indovinata, hai terminato i tentativi.

Vuoi indovinare un'altra parola? (s/n)

Test Prolog 6
Il giocatore 2 ha indovinato la parola.

Indovina la parola:
*****

Hai a disposizione 5 tentativi.
Inserisci una parola lunga 5: sacca.
****a

Hai a disposizione 4 tentativi.
Inserisci una parola lunga 5: panna.
p*nna

Hai a disposizione 3 tentativi.

Inserisci una parola lunga 5: penna.
penna

Parola indovinata correttamente!

Vuoi indovinare un'altra parola? (s/n)

Test Prolog 7
Il giocatore ha indovinato l'ultima parola della lista. La lista risulta vuota.

Indovina la parola:
****

Hai a disposizione 8 tentativi.
Inserisci una parola lunga 8: gazzella.
*a***lla

Hai a disposizione 7 tentativi.
Inserisci una parola lunga 8: forcella.
far**lla

Hai a disposizione 6 tentativi.
Inserisci una parola lunga 8: farfalla.
farfalla

Parola indovinata correttamente!

Non ci sono piu' parole da indovinare!

Test Prolog 8
Il giocatore non ha indovinato l'ultima parola della lista. La lista risulta vuota.

Indovina la parola:
****

Hai a disposizione 4 tentativi.
Inserisci una parola lunga 4: diga.
***a

Hai a disposizione 3 tentativi.
Inserisci una parola lunga 4: lira.
***a

Hai a disposizione 2 tentativi.
Inserisci una parola lunga 4: uova.
***a

Hai a disposizione 1 tentativo.
Inserisci una parola lunga 4: tata.
*a*a

Parola non indovinata, hai terminato i tentativi.

Non ci sono piu' parole da indovinare!

Test Prolog 9
Proposta di prosecuzione, con validazione dell'input ed esito positivo.

Hai a disposizione 2 tentativi.
Inserisci una parola lunga 5: nonna.
*a**a

Hai a disposizione 1 tentativo.
Inserisci una parola lunga 5: sacca.

Parola non indovinata, hai terminato i tentativi.

Vuoi indovinare un'altra parola? (s/n)
S.
Inserimento non valido. Riprova.

Vuoi indovinare un'altra parola? (s/n)
a.
Inserimento non valido. Riprova.

Vuoi indovinare un'altra parola? (s/n)
8.
Inserimento non valido. Riprova.

Vuoi indovinare un'altra parola? (s/n)
N.
Inserimento non valido. Riprova.

Vuoi indovinare un'altra parola? (s/n)
s.

Indovina la parola:
****

Hai a disposizione 4 tentativi.
Inserisci una parola lunga 4: sole.
*o**

Hai a disposizione 3 tentativi.
Inserisci una parola lunga 4: noto.
*o*o

Hai a disposizione 2 tentativi.
Inserisci una parola lunga 4: foto.
foto

Parola indovinata correttamente!

Vuoi indovinare un'altra parola? (s/n)

Test Prolog 10
Proposta di prosecuzione, con esito negativo e terminazione del gioco.

Indovina la parola:
*****

Hai a disposizione 5 tentativi.
Inserisci una parola lunga 5: cuore.
**o*e

Hai a disposizione 4 tentativi.
Inserisci una parola lunga 5: torre.
**ore

Hai a disposizione 3 tentativi.
Inserisci una parola lunga 4: fiore.
fiore

Parola indovinata correttamente!

Vuoi indovinare un'altra parola? (s/n)
n.

GAME OVER!