Università degli studi di urbino

Dipartimento di Scienze Pure e Applicate
Corso di Laurea in informatica applicata

Progetto di Programmazione Logica e Funzionale

di Pecmarkaj Arlind
Matricola 305991
Anno di corso: terzo

Anno Accademico 2022/2023 - Sessione estiva

Docente: Prof. Marco Bernardo

Indice

1 Specifica del problema

Scrivere un programma Haskell e un programma Prolog che acquisiscono dalla tastiera una formula di logica proposizionale in forma normale congiuntiva, stabiliscono se essa è una formula di Horn e, in caso affermativo, stabiliscono se essa è soddisfacibile stampando sullo schermo il minimo assegnamento di verità che la soddisfa.

2 Analisi del problema

2.1 Dati in ingresso del problema

L'unico dato in ingresso del problema è una FBF ϕ della logica proposizionale in forma normale congiuntiva.

2.2 Dati in uscita del problema

Nel caso ϕ fosse formula di Horn e soddisfacibile, il dato di uscita di problema è l'assegnamento di verità minimo (denotato con A) che soddisfa ϕ.

2.3 Relazioni intercorrenti

La FBF ϕ è in formula normale congiuntiva, i.e. ϕ è della forma

$$\bigwedge_{i=i}^{n} \left(\bigvee_{j=1}^{m_i} \lambda_{i,j} \right)$$

con $n \geq 1$ ed $m_i \geq 1$ per ogni $1 \leq i \leq n$, dove $\lambda_{i,j}$ è detto letterale ed è della forma p o $\neg p$ con $p \in Prop$.
Una disgiunzione di letterali è chiamata anche clausola.
Una clausola è detta clausola di Horn se contiene al più un letterale positivo (ossia della forma p).
Se ogni clausola in ϕ è una clausola di Horn, allora ϕ è una formula di Horn.
Il mimimo assegnamento di verità A è l'insieme con cardinalità più piccola appartentente a 2^{Prop} tale che $A \models \phi$.

3 Progettazione dell'algoritmo

3.1 Scelte di progetto

Per via della forma che le formule della logica proposizionale in forma normale congiuntiva assumono, esse si apprestono ad essere rappresentati mediante strutture dati lineari. Infatti possiamo notare dalla sezione 2.3 che effettivamente gli operatori di disgiunzione e di congiunzione non sono necessari per la rappresentazione se vediamo una clausola come un insieme di letterali e una formula come un insieme di clausole: gli operatori diventano dunque impliciti nel nostro caso.
Prendiamo per esempio la seguente FBF:

$$p \wedge (\neg p \vee q)$$

essa può essere vista come

$$\{\{p\}, \{\neg p, q\}\}$$

Se in fase di input chiediamo all'utente di seguire questa convenzione e inserire solo i letterali per ogni clausola, possiamo evitare di verificare la sintassi della formula. Questo ci impone un allocazione dinamica della struttura dati usata, in quanto non sappiamo a priori la lunghezza della formula. Inoltre non vogliamo avere dei limiti rispetto alla specifica del problema.

La convenzione verrà comunicata all'utente e sarà 'obbligato' a usarla per poter usufruire del programma senza anomalie, in tal modo evitiamo di dover implementare una validazione stretta. Formule con clausole ripetute verranno tollerate per il fatto che sono ammesse nella sintassi della forma normale congiuntiva. Le proposizioni potranno essere scritte come singoli caratteri o parole o anche come una sequenza alfanumerica, dando piena libertà all'utente in questo ambito.

Entrando nel dettaglio, implementeremo una formula come una lista di liste di stringhe. Si è deciso di usare la lista come struttura dati in quanto si addice ai linguaggi che verranno usati

3.2 Passi dell'algoritmo

I passi dell'algoritmo sono i seguenti ed esulano dal linguaggio implementativo e dalle strutture dati usate:

1. Acquisire ϕ in forma normale congiuntiva;
2. Per ogni clausola c in ϕ:
   (a) Se c contiene al più un letterale positivo, passa alla clausola successiva;
   (b) Se c contiene più di un letterale positivo, termina il programma e comunica che ϕ non è una formula di Horn;
3. Prendere un assegnamento di verita A e aggiungere ogni proposizione che compare come fatto in ϕ in A.
4. Finché A non è stabile, per ogni regola $p \vee (\bigvee_{i=1}^{n} \neg p_i)$ in cui $p_i \in A$ ma $p \notin A$, aggiungi p ad A.
5. Se esiste un vincolo $\bigvee_{i=1}^{n} \neg p_i$ in cui ogni $p_i \in A$, allora A è insoddisfacibile e lo si comunica. In caso contrario ϕ è soddisfacibile e si stampa A.

In particolare il punto 4, per via della natura dichiarativa dei linguaggi usati, dovrà essere implementato in maniera ricorsiva. La stabilità dell'insieme A si verifica quando dopo aver valutato ogni regola, non sono state aggiunte proposizioni all'assegnamento di verità. Per implementare ciò la funzione dovrà avere come argomento un iteratore che tiene conto di ciò. L'iteratore verrà incrementato in uno e un solo caso: quando anche solo una delle proposizioni dei letterali negativi non è già presente nell'assegnamento. Questo ci impone di tenere conto dell'ordine delle clausole nell'insieme che andranno valutate sequenzialmente e in questo caso, la clausola andrà messa in fondo per essere rivalutata nel caso si aggiungesse qualcosa di nuovo in A. Nel caso valutando una clausola la proposizione del letterale positivo fosse già nell'assegnamento, valutiamo tutte le clausole successive alla corrente. Nel caso una regola aggiungesse un elemento all'assegnamento, dovremo richiamare di nuovo la funzione su tutte le clausole rimanenti con l'iteratore reimpostato ad 1 in quanto ciò implica che l'assegnamento potrebbe ancora non essere stabile.
Il caso base della funzione si verifica quando l'insieme di regole è vuoto: in questo caso l'assegnamento di verità rimane uguale.
Alla luce di tutto la funzione viene formalizzata come segue, con il caso base che sarà:

$$assegnamentoDaRegole(I, A, \emptyset) = A$$

mentre i casi generali saranno (il nome della funzione è stato modificato solo per poter stare dentro i margini della pagina, *assegnamDaReg* fa sempre riferimento a *assegnamentoDaRegole*):

$$assegnamDaReg(I, A, \{c_i, ..., c_{i+k}\}) = \begin{cases} A \text{ se } I > k \\ assegnamDaReg(I, A\{c_{i+1}, ..., c_{i+k}\}) * \\ assegnamDaReg(1, A', \{c_{i+1}, ..., c_{i+k}\}) ** \\ assegnamDaReg(I+1, A, \{c_i, ...c_{i+k}\}) *** \end{cases}$$

dove $I, i, k \in \mathbb{N}$. La prima chiamata della funzione prevede I ed i a 1

* se c_i ha il letterale positivo già in A;

** se c_i ha le proposizioni dei letterali negativi in A e $A' = A \cup \{p\}$ dove p è il letterale positivo

*** se c_i presenta un letterale negativo la cui proposizione non è nell'assegnamento, la clausola valutata al momemento passa in fondo alla lista, ossia c_i diventa c_{i+k} e l'indice delle altre clausole è decrementato di uno.

Si specifica che la funzione è indipendente da qualsiasi linguaggio usato per la sua implementazione. L'implementazione effettiva differirà nella sintassi (ma non nel contenuto e nella semantica) leggermente in Haskell e Prolog per via della sintassi stessa dei due linguaggi in questione.

4 Implementazione dell'algoritmo

4.1 Implementazione in Haskell

Per via dell'editor di testo usato e del linguaggio di marcatura ad esso associato, l'identazione del codice qui presentato presenta minuscole differenze con quello presente nel file sorgente.
Per avere la miglior identazione e leggibilità si prega di consultare il codice originale, contenuto dentro "SoddisfacibilitaFormulaHorn.hs".
Il codice è stato pensato per essere compilato/interpretato usando Glasgow Haskell Compiler.
Il codice inizia nella pagina successiva

```haskell
-- ##########################################################
-- #     Corso di Programmazione Logica e Funzionale        #
-- #   Progetto per la sessione estiva A.A. 2022/2023       #
-- #              di Arlind Pecmarkaj                       #
-- #               Matricola: 305991                        #
-- #             Anno di corso: terzo                       #
-- ##########################################################
{- Specifica: Scrivere un programma Haskell che acquisisce dalla tastiera una
   formula di logica proposizionale in forma normale congiuntiva, stabilisce
   se essa e' una formula di Horn e, in caso affermativo, stabilisce se essa
   e' soddisfacibile stampando sullo schermo il minimo assegnamento di verita
   che la soddisfa. -}

{- Nota sui commenti: i commenti dentro '{- -}' descrivono le funzioni, i loro
   argomenti e i casi base e generali se son ricorsive. I commenti con '--'
   invece spiegano invece l'implementazione tecnica specifica in Haskell. -}
module Main where

-- Caricamento delle funzioni necessarie per le operazioni sulle liste.
import Data.List (isPrefixOf, nub, intersect)

{- Ridenominazione dei tipi di dato usati.
   Anche se Assegnamento e Clausola sono dello stesso tipo, per aumentare la
   leggibilita' si e' deciso avere due denominazioni diverse. -}
type Assegnamento = [String]
type Clausola     = [String]
type Formula      = [Clausola]

{- Funzione che verifica se una FBF della logica proposizionale in forma normale
   congiuntiva e' una formula di Horn o meno, controllando se ogni sua clausola e'
   di Horn:
   - l'argomento e' la FBF in fnc da valutare. -}
formulaHorn :: Formula -> Bool
formulaHorn = all clausolaHorn
    where
        {- Funzione che data una clausola della logica proposizionola, verifica se
           essa e' una clausola di Horn o meno:
           - l'argomento e' la clausola da verificare. -}
        clausolaHorn :: Clausola -> Bool
        -- Una clausola vuota e' banalmente una clausola di Horn.
        clausolaHorn [] = True
        -- Prendiamo dalla clausola i letterali positivi e verifichiamo se ne esiste al piu'
        -- uno solo.
        clausolaHorn c  = length (filter (\x -> head x /= '-') c) <= 1

{- Funzione che restituisce i fatti di una formula di Horn:
   - l'argomento e' la formula di Horn da cui ricabiamo i fatti. -}
fatti :: Formula -> [String]
-- Dalla formula ricaviamo solo le clausole unarie, le uniamo in unica lista,
```

```
-- eliminiamo quelle con la negazione davanti. Eliminiamo i duplicati per gestire
-- i casi in cui abbiamo fatti ripetuti nell'input.
fatti f = nub                          $
          filter (not . isPrefixOf "-") $
          concat                       $
          filter (\x -> length x <= 1) f

{- Funzione che restituisce i vincoli di una formula di Horn:
   - l'argomento e' la formula di Horn. -}
vincoli :: Formula -> Formula
vincoli = filter (all (isPrefixOf "-"))

{- Funzione che restituisce le regole di una formula di Horn:
   - l'argomento e' la formula di Horn -}
regole :: Formula -> Formula
-- Non ci interessano le clausole con un solo letterale (sicuro son fatti o vincoli)
-- e non ci interessano le clausole che sono vincoli di f.
regole f = filter (\c -> not (length c <= 1 || c `elem` vincoli f)) f

{- Funzione che date le regole di una formula di Horn e un assegnamento di verita'
   iniziale pari ai fatti della formula, restituisce l'assegnamento aggiornato in
   base alle regole:
   - il primo argomento e' un iteratore che serve per verificare se l'assegnamento
     e' stabile o meno, chiamando la funzione va messa ad 1;
   - il secondo argomento e' l'assegnamento di verita', ottenuto valutando i fatti
     della formula;
   - il terzo argomento la lista delle regole della formula. -}
assDaRegole :: Int -> Assegnamento -> Formula -> Assegnamento
{- Caso base: una regola vuota non aggiunge propoisizioni all'assegnamento di verita'. -}
assDaRegole _ assIniz []                              = assIniz
{- Casi generali:
   - il primo caso prevede che se abbiamo gia' controllato tutte le regole, verificando che
     l'iteratore sia maggiore del numero di regole visitate senza aggiungere nulla all'assegnamento,
     allora l'assegnamento A e' stabile e si restituisce esso;
   - il secondo caso prevede che se nella clausola corrente il letterale positivo e' gia' nell'as-
     segnamento, allora la clausola non ci interessa e valutiamo quella successiva;
   - il terzo caso prevede che se i letterali negativi hanno la proposizione nell'assegnamento,
     aggiungiamo il letterale positivo all'assegnamento di verita'. Ricominciamo l'iterazione
     visto che questa aggiunta implica che l'assegnamento potrebbe ancora non essere stabile;
   - il quarto caso prevede che se i letterali negativi non hanno la proposizione nell'assegnamen-
     to, allora valutiamo la clausola successiva e mettiamo quella attuale in coda alla lista. -}
assDaRegole iter assIniz (c:cs) | iter > length (c:cs) = assIniz
                                | giaInAssegnamento    = assDaRegole iter assIniz cs
                                | negInAssegnamento    = assDaRegole 1 nuovoAssegnamento cs
                                | not negInAssegnamento = assDaRegole (iter + 1) assIniz (cs ++ [c])
                            where
                                -- Denota il letterale posivivo gia' nell'assegnamento.
                                giaInAssegnamento = all (`elem` assIniz) positiviInRegola
                                -- Denota i letterali negativi la cui proposizione e' nell'assegnamento
                                negInAssegnamento = all ((`elem` assIniz) . tail)
```

```
                                        (filter (isPrefixOf "-") c)
                        -- Denota i letterali positivi nella regola
                        positiviInRegola = filter (not . isPrefixOf "-") c
                        -- Denota l'aggiunta del letterale positivo alla regola.
                        nuovoAssegnamento = positiviInRegola ++ assIniz

{- Funzione che data una formula di Horn della logica proposizionale, calcola se e' soddisfacibile e
   restituisce l'assegnamento di verita:
   - l'argomento e' la formula da valutare.
   * NOTA: se la formula non e' soddisfacibile viene resituito ["Insoddisfacibile"] -}
soddisfacibile :: Formula -> Assegnamento
soddisfacibile []                              = []
soddisfacibile f | vincoli f                  /= [] &&
                   verificaVincoliAssegnamento = ["Insoddisfacibile"]
                 | otherwise                   = assegnamento
                 where
                     -- Denota se esiste un vincolo i cui letterali sono tutti presenti nell'assegna-
                     -- mento. La logica e' quella di prendere i vincoli privati della negazione, fare
                     -- l'intersezione di ognuno di esso con l'assegnamento trovato e vedere se
                     -- esiste un vincolo la cui intersezione equivale a se stesso.
                     verificaVincoliAssegnamento =
                         any (\vinc -> vinc `intersect` assegnamento == vinc) lettVincoli
                     -- Assegnamento di verita' ottenuto.
                     assegnamento                = assDaRegole 1 (fatti f) (regole f)
                     -- Denota i vincoli privati della negazione.
                     lettVincoli                 = map (map (drop 1)) (vincoli f)

{---------------------------- Sezione non funzionale del codice -----------------------}
main :: IO ()
main = do
    print " ------------------------------- Benvenuto -------------------------------- "
    print " Il seguente programma verifica se una FBF in forma normale congiuntiva      "
    print " della logica proposizionale e' una formula di Horn o meno e in caso af-     "
    print " -fermativo stampa il minor assegnamento di verita' che la soddisfa.         "
    print " Inserire i letterali delle clausole, ogni clausola occupa una riga.         "
    print " Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)  "
    print " scriverai nell'input:                                                       "
    print " p <Invio>                                                                   "
    print " -p q <Invio>                                                                "
    print " -p -z <Invio>                                                               "
    print " Premere poi invio nella riga vuota per terminare l'input.                   "
    print " In caso la formula fosse formula di Horn, l'assegnamento di verita' sara'   "
    print " rappresentato da una lista di proposizioni.                                 "
    print " NOTA: [] denota l'insieme vuoto, [\"Insoddisfacibile\"] denota che la formu-"
    print " -la e' una contraddizione.                                                  "
    print " Inserire l'input:                                                           "
    formula <- leggiFormula []
    if formulaHorn formula
        then do
            print " La formula inserita e' una formula di Horn!    "
```

```
            print " L'assegnamento minimo di verita' che la soddisfa e': "
            putStrLn $ show (soddisfacibile formula)
        else
            print " La formula inserita non e' una formula di Horn! "

{- Funzione di supporto richiesta per l'input della formula -}
-- Prendiamo una clausola in ogni linea e ogni letterale e' separato da spazio.
-- In questo modo l'utente non dovra' inserire i simboli '[' e ']' per le liste.
leggiFormula :: [[String]] -> IO [[String]]
leggiFormula f = do
                    line <- getLine
                    if null line
                        then return f
                    else do
                        let list = words line
                        leggiFormula (list : f)
```

4.2 Implementazione in Prolog

Con le stesse motivazioni per l'implementazione in Haskell si fa notare della differenza tra codice presentato qui e quello nel file sorgente. Si prega di consultare "SoddisfacibilitaFormulaHorn.pl" per avere la miglior identazione e leggibilità. Il codice è stato scritto seguendo l'implementazione GNU Prolog e compilato con gprolog 1.5.0. Il codice inizia a pagina successiva.

```
   /* ########################################################
   #      Corso di Programmazione Logica e Funzionale #
   #    Progetto per la sessione estiva A.A. 2022/2023 #
   #               di Arlind Pecmarkaj                #
   #                Matricola: 305991                 #
   #              Anno di corso: terzo                #
   ########################################################
   Specifica: Scrivere un programma Prolog che acquisisce dalla tastiera una
   formula di logica proposizionale in forma normale congiuntiva, stabilisce
   se essa e' una formula di Horn e, in caso affermativo, stabilisce se essa
   e' soddisfacibile stampando sullo schermo il minimo assegnamento di verita
   che la soddisfa. */


/* Predicato principale del programma. */
/* set_prolog_flag e' un predicato di prolog che viene usato per cambiare la rappresentazione
   delle stringhe da lista di caratteri rappresentati dalla loro codifica decimale ASCII a lista
   di caratteri rappresentati dal loro simbolo di carattere. */
main :- set_prolog_flag(double_quotes, chars),
       write('------------------------------- Benvenuto --------------------------------'),nl,
       write('Il seguente programma verifica se una FBF in forma normale congiuntiva'),     nl,
       write('della logica proposizionale e\' una formula di Horn o meno e in caso af-'),   nl,
       write('-fermativo stampa il minor assegnamento di verita\' che la soddisfa.'),       nl,
       write('La formula va inserita come una lista di liste di stringhe dove ogni lette-'),nl,
       write('rale èuna stringa. '),                                                        nl,
       write('Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)'), nl,
       write('scriverai nell\'input: '),                                                    nl,
       write('[["p"],["-p", "q"], ["-p", "-z"]]. <Invio>'),                                 nl,
       write('Si ricorda il punto finale prima dell\'invio e si raccomanda di seguire'),    nl,
       write('questa convenzione per evitare comportamenti anomali del programma.'),        nl,
       write('In caso la formula fosse di Horn, l\'assegnamento di verita\' sara\' rap-'),  nl,
       write('-presentato da una lista di proposizioni.'),                                  nl,
       write('NOTA: [] denota l\'insieme vuoto, [insoddis] denota che la formula e\' una'), nl,
       write('contraddizione. '),                                                           nl,
       write('Inserire la formula: '),
       read(T),
       if_else_stampa(formulaHorn(T), T).

/* Predicato che viene usato per implementare un if-else specifico per il programma.
   Viene usato per differenziare nel caso l'utente scrivesse una formula di Horn o meno:
   - il primo argomento e' la condizione che in questo caso e' la verifica di appartenenza
     alle formule di Horn;
   - il secondo argomento e' la formula, nel primo caso viene usata poi per calcolare
     l'assegnamento minimo di verita'. */
if_else_stampa(Cond, Form) :- Cond,
                              !,
                              write('La formula inserita e\' una formula di Horn!'), nl,
                              soddisfacibile(Form, AssMin),
                              write('L\'assegnamento minimo che la soddisfa e\':'), nl,
```

```
                          write(AssMin).

if_else_stampa(Cond, _) :- \+Cond,
                          !,
                          write('La formula inserita non e\' una formula di Horn!').

/* Fatto che indica che un letterale negativo inizia con '-.'*/
letteraleNegativo([-|_]).

/* Predicato che verifica se una clausola non e' unaria:
   - l'argomento e' la clausola da verificare. */
nonUnaria(Clausola) :- length(Clausola, N), N > 1.

/* Predicato che verifica se una clausola non e' un vincolo:
   - l'argomento e' la clausola da verificare. */
nonVincolo(Clausola) :- numeroPos(Clausola, N), N > 0.

/* Predicato che calcola il numero dei letterali positivi da una clausola:
   - il primo argomento e' la clausola;
   - il secondo argomento e' il numero di letterali positivi. */
numeroPos(Cl, N) :- findall(Lett,
                            (member(Lett, Cl), \+letteraleNegativo(Lett)),
                            Positivi),
                    length(Positivi, N).

/* Predicato che verifica se una FBF della logica proposizione in forma normale congiuntiva
   e' una formula di Horn o meno:
   - l'argomento e' la FBF. */
formulaHorn(Formula) :- forall(member(Clausola, Formula), (numeroPos(Clausola, 0);numeroPos(Clausola, 1))).

/* Predicato che calcola tutti i fatti di una formula di Horn:
   - il primo argomento e' la formula;
   - il secondo argomento e' l'insieme dei fatti. */
fattiFormula(Formula, Fatti) :- findall(Letterale,
                                       (member(Clausola, Formula),
                                        length(Clausola, 1),
                                        member(Letterale, Clausola),
                                        \+letteraleNegativo(Letterale)),
                                       Fatti).

/* Predicato che calcola tutte le regole di una formula di Horn:
   - il primo argomento e' la formula;
   - il secondo argomento e' l'insieme di regole. */
regoleFormula(Formula, Regole) :- findall(Clausola,
                                         (member(Clausola, Formula),
                                          nonUnaria(Clausola),
                                          nonVincolo(Clausola)),
                                         Regole).
```

```
/* Predicato che calcola tutti i vincoli di una formula di Horn:
   - il primo argomento e' la formula;
   - il secondo argomento e' l'insieme di vincoli. */
vincoliFormula(Formula, Vincoli) :- findall(Clausola,
                                            (member(Clausola, Formula), numeroPos(Clausola, 0)),
                                            Vincoli).

/* Predicato che da un insieme di regole di una formula di Horn e un assegnamento iniziale, restituisce
   l'assegnamento aggiornato.
   - il primo argomento e' l'insieme di regole;
   - il secondo argomento èun iteratore che verifica se l'assegnamento e' stabile o meno; quando il
     predicato viene invocato, va messo a uno;
   - il terzo argomento e' l'assegnamento iniziale;
   - il quarto argomento e' l'assegnamento finale aggiornato. */
/* Caso base: un insieme di regole vuoto non cambia l'assegnamento di verita' */
assegnamentoDaRegole([], _, AssIniz, AssIniz).
/* Casi generali:
   - il primo prevede che se abbiamo gia' controllato tutte le regole, verificando che l'iteratore sia maggiore
     del numero di regole visitate senza aggiungere nulla all'assegnamento, allora l'assegnamento A e' stabile
     e si restituisce esso;
   - il secondo prevede che se nella clausola corrente il letterale positivo e' gia' presente nell'assegnamento,
     allora la clausola non ci interessa e valutiamo quella successiva;
   - il terzo caso prevede che se i letterali negativi hanno la proposizione nell'assegnamento aggiungiamo il
     letterale positivo all'assegnamento di verita'. Ricominciamo l'iterazione visto che questa aggiunta implica
     che l'assegnamento potrebbe ancora non essere stabile.
   - il quarto caso prevede che se i letterali negativi non hanno la proposizione nell'assegnamento (non lo
     verifichiamo direttamente, ma vedendo che la chiamata al predicato non unifica con le precedenti), allora
     valutiamo la clausola successiva e mettiamo quella attuale in coda alla lista. */
assegnamentoDaRegole(Regole, Iteratore, AssIniz, AssIniz) :- length(Regole, N), Iteratore > N.

assegnamentoDaRegole([R|Rs], Iteratore, AssIniz, AssFina) :- positiviInAssegnamento(R, AssIniz),
                                                             assegnamentoDaRegole(Rs, Iteratore, AssIniz, AssFina).

assegnamentoDaRegole([R|Rs], _, AssIniz, AssFina)          :- negativiInAssegnamento(R, AssIniz),
                                                              nuovoAssegnamento(R, AssIniz, AssTmp),
                                                              assegnamentoDaRegole(Rs, 1, AssTmp, AssFina).

assegnamentoDaRegole([R|Rs], Iteratore, AssIniz, AssFina)  :- append(Rs, [R], RN),
                                                             assegnamentoDaRegole(RN, Iteratore + 1, AssIniz, AssFina).

/* Predicato che verifica se una regola presenta il letterale positivo nell'assegnamento:
   - il primo argomento e' la regola;
   - il secondo argomento e' l'assegnamento di verita'. */
positiviInAssegnamento([], _).

positiviInAssegnamento([Lett|Letts], Assegnamento) :- letteraleNegativo(Lett),
                                                      positiviInAssegnamento(Letts, Assegnamento).

positiviInAssegnamento([Lett|Letts], Assegnamento) :- member(Lett, Assegnamento),
                                                      positiviInAssegnamento(Letts, Assegnamento).
```

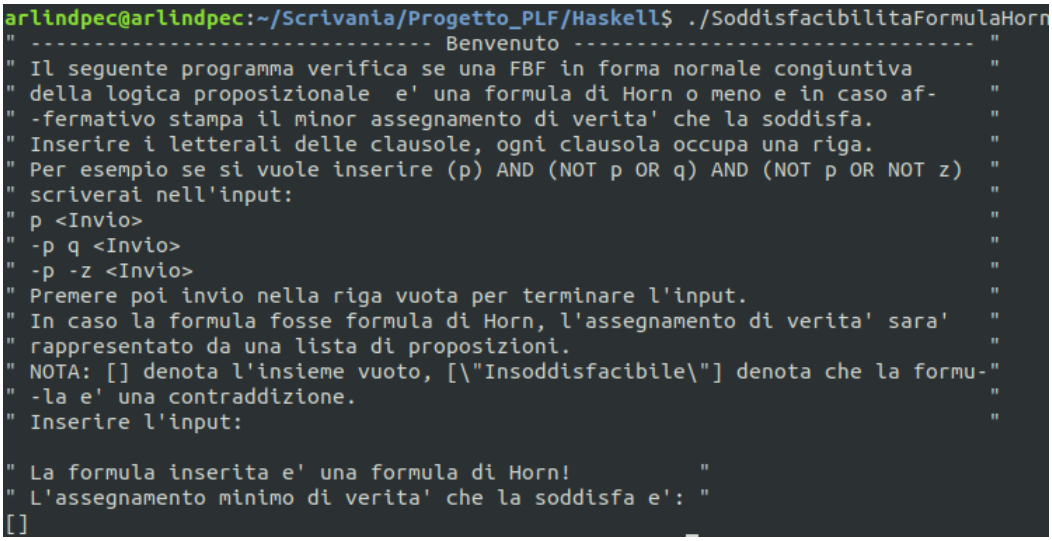
```
/* Predicato che verifica se le proposizioni dei letterali negativi di una regola sono nell'assegnamento:
   - il primo argomento e' la regola;
   - il secondo argomento e' l'assegnamento di verita'. */
negativiInAssegnamento([], _).

negativiInAssegnamento([Lett|Letts], Assegnamento) :- letteraleNegativo(Lett),
                                                      filtraNegazione(Lett, Prop),
                                                      member(Prop, Assegnamento),
                                                      negativiInAssegnamento(Letts, Assegnamento).

negativiInAssegnamento([Lett|Letts], Assegnamento) :- \+letteraleNegativo(Lett),
                                                      negativiInAssegnamento(Letts, Assegnamento).

/* Predicato che aggiorna un assegnamento di àverit aggiungendo il letterale positivo di una regola
   che ha tutte le proposizioni del letterali negativi nell'assegnamento:
   - il primo argomento e' la regola;
   - il secondo argomento e' l'assegnamento iniziale;
   - il terzo argomento e' l'assegnamento aggiornato. */
nuovoAssegnamento([], Ass, Ass).

nuovoAssegnamento([Lett|Letts], Ass, AssFin) :- \+letteraleNegativo(Lett),
                                                \+member(Lett, Ass),
                                                append(Ass, [Lett], AssTmp),
                                                nuovoAssegnamento(Letts, AssTmp, AssFin).

nuovoAssegnamento([Lett|Letts], Ass, AssFin) :- letteraleNegativo(Lett),
                                                nuovoAssegnamento(Letts, Ass, AssFin).

/* Fatto che rimuove il simbolo di negazione da un letterale negativo. */
filtraNegazione([-|Prop], Prop).

/* Predicato che verifica se una formula di Horn e' soddisfacibile e se lo fosse calcola
   l'assegnamento minimo di verita':
   - il primo argomento e' la formula di Horn;
   - il secondo argomento e' l'assegnamento minimo di verita'. Nel caso la formula fosse
     insoddisfacibile, equivale a [insoddis]. */
soddisfacibile(Formula, AssegnMin) :- fattiFormula(Formula, Assegn),
                                      regoleFormula(Formula, Regole),
                                      vincoliFormula(Formula, Vincoli),
                                      assegnamentoDaRegole(Regole, 1, Assegn, AssegnMin),
                                     vincoliNonInAssegnamento(Vincoli, AssegnMin).
/* Nel caso esistesse un vincolo che ha le proposizioni dei letterali nell'assegnamento,
   allora il predicato sopra non unifica. Per come abbiamo impostato la logica, questo
   e' uno dei pochi casi in cui un fatto va dopo una regola. */
soddisfacibile(_, ['insoddis']).
```

```
/* Predicato che verifica se non esiste nessun vincolo le cui proposizioni dei letterali
   siano tutte nell'assegnamento di verita':
   - il primo argomento e' l'insieme di vincoli di una formula;
   - il secondo argomento e' l'assegnamento di verita'. */
vincoliNonInAssegnamento([], _).

vincoliNonInAssegnamento([V|Vs], Assegnamento) :- vincoloNonInAssegnamento(V, Assegnamento),
                                                  vincoliNonInAssegnamento(Vs, Assegnamento).

/* Predicato che verifica se un vincolo non presenta le proposizioni dei letterali
   nell'assegnamento di verita':
   - il primo argomento e' il vincolo;
   - il secondo argomento e' l'assegnamento di verita' */
vincoloNonInAssegnamento([N], Assegnamento)    :- filtraNegazione(N, Lett),
                                                  \+member(Lett, Assegnamento).

vincoloNonInAssegnamento([N|Ns], Assegnamento) :- filtraNegazione(N, Lett),
                                                  \+member(Lett, Assegnamento);
                                                  vincoloNonInAssegnamento(Ns, Assegnamento).
```

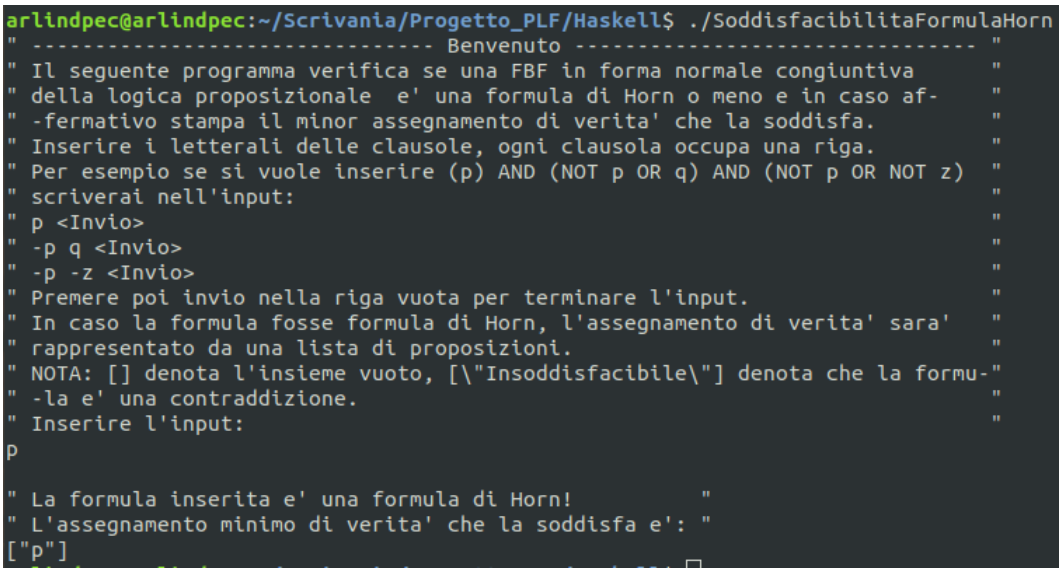
5 Testing

5.1 Testing del programma Haskell

Test nr. 1



Test nr. 2



Test nr. 3

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Haskell$ ./SoddisfacibilitaFormulaHorn
" ------------------------------- Benvenuto -------------------------------- "
" Il seguente programma verifica se una FBF in forma normale congiuntiva      "
" della logica proposizionale  e' una formula di Horn o meno e in caso af-    "
" -fermativo stampa il minor assegnamento di verita' che la soddisfa.         "
" Inserire i letterali delle clausole, ogni clausola occupa una riga.         "
" Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)  "
" scriverai nell'input:                                                       "
" p <Invio>                                                                   "
" -p q <Invio>                                                                "
" -p -z <Invio>                                                               "
" Premere poi invio nella riga vuota per terminare l'input.                   "
" In caso la formula fosse formula di Horn, l'assegnamento di verita' sara'   "
" rappresentato da una lista di proposizioni.                                 "
" NOTA: [] denota l'insieme vuoto, [\"Insoddisfacibile\"] denota che la formu-"
" -la e' una contraddizione.                                                 "
" Inserire l'input:                                                           "
p
-p q
-p -z

" La formula inserita e' una formula di Horn!          "
" L'assegnamento minimo di verita' che la soddisfa e': "
["q","p"]
```

Test nr. 4

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Haskell$ ./SoddisfacibilitaFormulaHorn
" ------------------------------- Benvenuto -------------------------------- "
" Il seguente programma verifica se una FBF in forma normale congiuntiva      "
" della logica proposizionale  e' una formula di Horn o meno e in caso af-    "
" -fermativo stampa il minor assegnamento di verita' che la soddisfa.         "
" Inserire i letterali delle clausole, ogni clausola occupa una riga.         "
" Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)  "
" scriverai nell'input:                                                       "
" p <Invio>                                                                   "
" -p q <Invio>                                                                "
" -p -z <Invio>                                                               "
" Premere poi invio nella riga vuota per terminare l'input.                   "
" In caso la formula fosse formula di Horn, l'assegnamento di verita' sara'   "
" rappresentato da una lista di proposizioni.                                 "
" NOTA: [] denota l'insieme vuoto, [\"Insoddisfacibile\"] denota che la formu-"
" -la e' una contraddizione.                                                 "
" Inserire l'input:                                                           "
p
p q
-p -q

" La formula inserita non e' una formula di Horn! "
```

Test nr. 5

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Haskell$ ./SoddisfacibilitaFormulaHorn
" ------------------------------ Benvenuto ------------------------------- "
" Il seguente programma verifica se una FBF in forma normale congiuntiva      "
" della logica proposizionale  e' una formula di Horn o meno e in caso af-    "
" -fermativo stampa il minor assegnamento di verita' che la soddisfa.         "
" Inserire i letterali delle clausole, ogni clausola occupa una riga.         "
" Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)  "
" scriverai nell'input:                                                       "
" p <Invio>                                                                   "
" -p q <Invio>                                                                "
" -p -z <Invio>                                                               "
" Premere poi invio nella riga vuota per terminare l'input.                   "
" In caso la formula fosse formula di Horn, l'assegnamento di verita' sara'   "
" rappresentato da una lista di proposizioni.                                 "
" NOTA: [] denota l'insieme vuoto, [\"Insoddisfacibile\"] denota che la formu-"
" -la e' una contraddizione.                                                 "
" Inserire l'input:                                                           "
p
-p

" La formula inserita e' una formula di Horn!          "
" L'assegnamento minimo di verita' che la soddisfa e': "
["Insoddisfacibile"]
```

Test nr. 6

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Haskell$ ./SoddisfacibilitaFormulaHorn
" ------------------------------ Benvenuto ------------------------------- "
" Il seguente programma verifica se una FBF in forma normale congiuntiva      "
" della logica proposizionale  e' una formula di Horn o meno e in caso af-    "
" -fermativo stampa il minor assegnamento di verita' che la soddisfa.         "
" Inserire i letterali delle clausole, ogni clausola occupa una riga.         "
" Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)  "
" scriverai nell'input:                                                       "
" p <Invio>                                                                   "
" -p q <Invio>                                                                "
" -p -z <Invio>                                                               "
" Premere poi invio nella riga vuota per terminare l'input.                   "
" In caso la formula fosse formula di Horn, l'assegnamento di verita' sara'   "
" rappresentato da una lista di proposizioni.                                 "
" NOTA: [] denota l'insieme vuoto, [\"Insoddisfacibile\"] denota che la formu-"
" -la e' una contraddizione.                                                 "
" Inserire l'input:                                                           "
p
q
-p -q z
-z b -p

" La formula inserita e' una formula di Horn!          "
" L'assegnamento minimo di verita' che la soddisfa e': "
["b","z","q","p"]
```

Test nr. 7

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Haskell$ ./SoddisfacibilitaFormulaHorn
" ------------------------------ Benvenuto ------------------------------- "
" Il seguente programma verifica se una FBF in forma normale congiuntiva      "
" della logica proposizionale  e' una formula di Horn o meno e in caso af-    "
" -fermativo stampa il minor assegnamento di verita' che la soddisfa.         "
" Inserire i letterali delle clausole, ogni clausola occupa una riga.         "
" Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)  "
" scriverai nell'input:                                                       "
" p <Invio>                                                                   "
" -p q <Invio>                                                                "
" -p -z <Invio>                                                               "
" Premere poi invio nella riga vuota per terminare l'input.                   "
" In caso la formula fosse formula di Horn, l'assegnamento di verita' sara'   "
" rappresentato da una lista di proposizioni.                                 "
" NOTA: [] denota l'insieme vuoto, [\"Insoddisfacibile\"] denota che la formu-"
" -la e' una contraddizione.                                                 "
" Inserire l'input:                                                           "
p
-p q
-p -q

" La formula inserita e' una formula di Horn!          "
" L'assegnamento minimo di verita' che la soddisfa e': "
["Insoddisfacibile"]
```

Test nr. 8

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Haskell$ ./SoddisfacibilitaFormulaHorn
" ------------------------------ Benvenuto ------------------------------- "
" Il seguente programma verifica se una FBF in forma normale congiuntiva      "
" della logica proposizionale  e' una formula di Horn o meno e in caso af-    "
" -fermativo stampa il minor assegnamento di verita' che la soddisfa.         "
" Inserire i letterali delle clausole, ogni clausola occupa una riga.         "
" Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)  "
" scriverai nell'input:                                                       "
" p <Invio>                                                                   "
" -p q <Invio>                                                                "
" -p -z <Invio>                                                               "
" Premere poi invio nella riga vuota per terminare l'input.                   "
" In caso la formula fosse formula di Horn, l'assegnamento di verita' sara'   "
" rappresentato da una lista di proposizioni.                                 "
" NOTA: [] denota l'insieme vuoto, [\"Insoddisfacibile\"] denota che la formu-"
" -la e' una contraddizione.                                                 "
" Inserire l'input:                                                           "
p -q

" La formula inserita e' una formula di Horn!          "
" L'assegnamento minimo di verita' che la soddisfa e': "
[]
```

Test nr. 9

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Haskell$ ./SoddisfacibilitaFormulaHorn
" ------------------------------ Benvenuto ------------------------------- "
" Il seguente programma verifica se una FBF in forma normale congiuntiva      "
" della logica proposizionale  e' una formula di Horn o meno e in caso af-    "
" -fermativo stampa il minor assegnamento di verita' che la soddisfa.         "
" Inserire i letterali delle clausole, ogni clausola occupa una riga.         "
" Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)  "
" scriverai nell'input:                                                       "
" p <Invio>                                                                   "
" -p q <Invio>                                                                "
" -p -z <Invio>                                                               "
" Premere poi invio nella riga vuota per terminare l'input.                   "
" In caso la formula fosse formula di Horn, l'assegnamento di verita' sara'   "
" rappresentato da una lista di proposizioni.                                 "
" NOTA: [] denota l'insieme vuoto, [\"Insoddisfacibile\"] denota che la formu-"
" -la e' una contraddizione.                                                 "
" Inserire l'input:                                                           "
-a -b c
a
b
-z -q
-c

" La formula inserita e' una formula di Horn!          "
" L'assegnamento minimo di verita' che la soddisfa e': "
["Insoddisfacibile"]
```

Test nr. 10

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Haskell$ ./SoddisfacibilitaFormulaHorn
" ------------------------------ Benvenuto ------------------------------- "
" Il seguente programma verifica se una FBF in forma normale congiuntiva      "
" della logica proposizionale  e' una formula di Horn o meno e in caso af-    "
" -fermativo stampa il minor assegnamento di verita' che la soddisfa.         "
" Inserire i letterali delle clausole, ogni clausola occupa una riga.         "
" Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)  "
" scriverai nell'input:                                                       "
" p <Invio>                                                                   "
" -p q <Invio>                                                                "
" -p -z <Invio>                                                               "
" Premere poi invio nella riga vuota per terminare l'input.                   "
" In caso la formula fosse formula di Horn, l'assegnamento di verita' sara'   "
" rappresentato da una lista di proposizioni.                                 "
" NOTA: [] denota l'insieme vuoto, [\"Insoddisfacibile\"] denota che la formu-"
" -la e' una contraddizione.                                                 "
" Inserire l'input:                                                           "
p
-p q
-p q

" La formula inserita e' una formula di Horn!          "
" L'assegnamento minimo di verita' che la soddisfa e': "
["q","p"]
```

5.2 Testing del programma Prolog

Test nr. 1

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Prolog$ ./SoddisfacibilitaFormulaHorn
GNU Prolog 1.5.0 (64 bits)
Compiled May 24 2023, 19:59:04 with gcc
Copyright (C) 1999-2023 Daniel Diaz

| ?- main.
-------------------------------- Benvenuto --------------------------------
Il seguente programma verifica se una FBF in forma normale congiuntiva
della logica proposizionale e' una formula di Horn o meno e in caso af-
-fermativo stampa il minor assegnamento di verita' che la soddisfa.
La formula va inserita come una lista di liste di stringhe dove ogni lette-
rale è una stringa.
Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)
scriverai nell'input:
[["p"],["-p", "q"], ["-p", "-z"]]. <Invio>
Si ricorda il punto finale prima dell'invio e si raccomanda di seguire
questa convenzione per evitare comportamenti anomali del programma.
In caso la formula fosse di Horn, l'assegnamento di verita' sara' rap-
-presentato da una lista di proposizioni.
NOTA: [] denota l'insieme vuoto, [insoddis] denota che la formula e' una
contraddizione.
Inserire la formula: [["p"]].
La formula inserita e' una formula di Horn!
L'assegnamento minimo che la soddisfa e':
[[p]]

true ?

(1 ms) yes
```

Test nr. 2

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Prolog$ ./SoddisfacibilitaFormulaHorn
GNU Prolog 1.5.0 (64 bits)
Compiled May 24 2023, 19:59:04 with gcc
Copyright (C) 1999-2023 Daniel Diaz

| ?- main.
-------------------------------- Benvenuto --------------------------------
Il seguente programma verifica se una FBF in forma normale congiuntiva
della logica proposizionale e' una formula di Horn o meno e in caso af-
-fermativo stampa il minor assegnamento di verita' che la soddisfa.
La formula va inserita come una lista di liste di stringhe dove ogni lette-
rale è una stringa.
Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)
scriverai nell'input:
[["p"],["-p", "q"], ["-p", "-z"]]. <Invio>
Si ricorda il punto finale prima dell'invio e si raccomanda di seguire
questa convenzione per evitare comportamenti anomali del programma.
In caso la formula fosse di Horn, l'assegnamento di verita' sara' rap-
-presentato da una lista di proposizioni.
NOTA: [] denota l'insieme vuoto, [insoddis] denota che la formula e' una
contraddizione.
Inserire la formula: [["p"],["-p"]].
La formula inserita e' una formula di Horn!
L'assegnamento minimo che la soddisfa e':
[insoddis]

(1 ms) yes
```

Test nr. 3

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Prolog$ ./SoddisfacibilitaFormulaHorn
GNU Prolog 1.5.0 (64 bits)
Compiled May 24 2023, 19:59:04 with gcc
Copyright (C) 1999-2023 Daniel Diaz

| ?- main.
------------------------------- Benvenuto --------------------------------
Il seguente programma verifica se una FBF in forma normale congiuntiva
della logica proposizionale e' una formula di Horn o meno e in caso af-
-fermativo stampa il minor assegnamento di verita' che la soddisfa.
La formula va inserita come una lista di liste di stringhe dove ogni lette-
rale è una stringa.
Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)
scriverai nell'input:
[["p"],["-p", "q"], ["-p", "-z"]]. <Invio>
Si ricorda il punto finale prima dell'invio e si raccomanda di seguire
questa convenzione per evitare comportamenti anomali del programma.
In caso la formula fosse di Horn, l'assegnamento di verita' sara' rap-
-presentato da una lista di proposizioni.
NOTA: [] denota l'insieme vuoto, [insoddis] denota che la formula e' una
contraddizione.
Inserire la formula: [["p"],["-p","c","-a"],["a"]].
La formula inserita e' una formula di Horn!
L'assegnamento minimo che la soddisfa e':
[[p],[a],[c]]

true ?

(1 ms) yes
```

Test nr. 4

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Prolog$ ./SoddisfacibilitaFormulaHorn
GNU Prolog 1.5.0 (64 bits)
Compiled May 24 2023, 19:59:04 with gcc
Copyright (C) 1999-2023 Daniel Diaz

| ?- main.
------------------------------- Benvenuto --------------------------------
Il seguente programma verifica se una FBF in forma normale congiuntiva
della logica proposizionale e' una formula di Horn o meno e in caso af-
-fermativo stampa il minor assegnamento di verita' che la soddisfa.
La formula va inserita come una lista di liste di stringhe dove ogni lette-
rale è una stringa.
Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)
scriverai nell'input:
[["p"],["-p", "q"], ["-p", "-z"]]. <Invio>
Si ricorda il punto finale prima dell'invio e si raccomanda di seguire
questa convenzione per evitare comportamenti anomali del programma.
In caso la formula fosse di Horn, l'assegnamento di verita' sara' rap-
-presentato da una lista di proposizioni.
NOTA: [] denota l'insieme vuoto, [insoddis] denota che la formula e' una
contraddizione.
Inserire la formula: [["-p","-q","-c","-d"]].
La formula inserita e' una formula di Horn!
L'assegnamento minimo che la soddisfa e':
[]

true ?
```

Test nr. 5

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Prolog$ ./SoddisfacibilitaFormulaHorn
GNU Prolog 1.5.0 (64 bits)
Compiled May 24 2023, 19:59:04 with gcc
Copyright (C) 1999-2023 Daniel Diaz

| ?- main.
------------------------------- Benvenuto -------------------------------
Il seguente programma verifica se una FBF in forma normale congiuntiva
della logica proposizionale e' una formula di Horn o meno e in caso af-
-fermativo stampa il minor assegnamento di verita' che la soddisfa.
La formula va inserita come una lista di liste di stringhe dove ogni lette-
rale è una stringa.
Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)
scriverai nell'input:
[["p"],["-p", "q"], ["-p", "-z"]]. <Invio>
Si ricorda il punto finale prima dell'invio e si raccomanda di seguire
questa convenzione per evitare comportamenti anomali del programma.
In caso la formula fosse di Horn, l'assegnamento di verita' sara' rap-
-presentato da una lista di proposizioni.
NOTA: [] denota l'insieme vuoto, [insoddis] denota che la formula e' una
contraddizione.
Inserire la formula: [["p"],["-p","c","a"],["a"]].
La formula inserita non e' una formula di Horn!

(2 ms) yes
```

Test nr. 6

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Prolog$ ./SoddisfacibilitaFormulaHorn
GNU Prolog 1.5.0 (64 bits)
Compiled May 24 2023, 19:59:04 with gcc
Copyright (C) 1999-2023 Daniel Diaz

| ?- main.
------------------------------- Benvenuto -------------------------------
Il seguente programma verifica se una FBF in forma normale congiuntiva
della logica proposizionale e' una formula di Horn o meno e in caso af-
-fermativo stampa il minor assegnamento di verita' che la soddisfa.
La formula va inserita come una lista di liste di stringhe dove ogni lette-
rale è una stringa.
Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)
scriverai nell'input:
[["p"],["-p", "q"], ["-p", "-z"]]. <Invio>
Si ricorda il punto finale prima dell'invio e si raccomanda di seguire
questa convenzione per evitare comportamenti anomali del programma.
In caso la formula fosse di Horn, l'assegnamento di verita' sara' rap-
-presentato da una lista di proposizioni.
NOTA: [] denota l'insieme vuoto, [insoddis] denota che la formula e' una
contraddizione.
Inserire la formula: [["p"],["-p","q"], ["-p", "-z"]].
La formula inserita e' una formula di Horn!
L'assegnamento minimo che la soddisfa e':
[[p],[q]]

true ?

(1 ms) yes
```

Test nr. 7

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Prolog$ ./SoddisfacibilitaFormulaHorn
GNU Prolog 1.5.0 (64 bits)
Compiled May 24 2023, 19:59:04 with gcc
Copyright (C) 1999-2023 Daniel Diaz

| ?- main.
------------------------------- Benvenuto -------------------------------
Il seguente programma verifica se una FBF in forma normale congiuntiva
della logica proposizionale e' una formula di Horn o meno e in caso af-
-fermativo stampa il minor assegnamento di verita' che la soddisfa.
La formula va inserita come una lista di liste di stringhe dove ogni lette-
rale è una stringa.
Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)
scriverai nell'input:
[["p"],["-p", "q"], ["-p", "-z"]]. <Invio>
Si ricorda il punto finale prima dell'invio e si raccomanda di seguire
questa convenzione per evitare comportamenti anomali del programma.
In caso la formula fosse di Horn, l'assegnamento di verita' sara' rap-
-presentato da una lista di proposizioni.
NOTA: [] denota l'insieme vuoto, [insoddis] denota che la formula e' una
contraddizione.
Inserire la formula: [["-p"],["-d"],["p"],["c"]].
La formula inserita e' una formula di Horn!
L'assegnamento minimo che la soddisfa e':
[insoddis]

(2 ms) yes
```

Test nr. 8

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Prolog$ ./SoddisfacibilitaFormulaHorn
GNU Prolog 1.5.0 (64 bits)
Compiled May 24 2023, 19:59:04 with gcc
Copyright (C) 1999-2023 Daniel Diaz

| ?- main.
------------------------------- Benvenuto -------------------------------
Il seguente programma verifica se una FBF in forma normale congiuntiva
della logica proposizionale e' una formula di Horn o meno e in caso af-
-fermativo stampa il minor assegnamento di verita' che la soddisfa.
La formula va inserita come una lista di liste di stringhe dove ogni lette-
rale è una stringa.
Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)
scriverai nell'input:
[["p"],["-p", "q"], ["-p", "-z"]]. <Invio>
Si ricorda il punto finale prima dell'invio e si raccomanda di seguire
questa convenzione per evitare comportamenti anomali del programma.
In caso la formula fosse di Horn, l'assegnamento di verita' sara' rap-
-presentato da una lista di proposizioni.
NOTA: [] denota l'insieme vuoto, [insoddis] denota che la formula e' una
contraddizione.
Inserire la formula: [["a"],["-b","-c","a"],["-z","-a","c"],["z"]].
La formula inserita e' una formula di Horn!
L'assegnamento minimo che la soddisfa e':
[[a],[z],[c]]

true ?

(1 ms) yes
```

Test nr. 9

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Prolog$ ./SoddisfacibilitaFormulaHorn
GNU Prolog 1.5.0 (64 bits)
Compiled May 24 2023, 19:59:04 with gcc
Copyright (C) 1999-2023 Daniel Diaz

| ?- main.
------------------------------- Benvenuto -------------------------------
Il seguente programma verifica se una FBF in forma normale congiuntiva
della logica proposizionale e' una formula di Horn o meno e in caso af-
-fermativo stampa il minor assegnamento di verita' che la soddisfa.
La formula va inserita come una lista di liste di stringhe dove ogni lette-
rale è una stringa.
Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)
scriverai nell'input:
[["p"],["-p", "q"], ["-p", "-z"]]. <Invio>
Si ricorda il punto finale prima dell'invio e si raccomanda di seguire
questa convenzione per evitare comportamenti anomali del programma.
In caso la formula fosse di Horn, l'assegnamento di verita' sara' rap-
-presentato da una lista di proposizioni.
NOTA: [] denota l'insieme vuoto, [insoddis] denota che la formula e' una
contraddizione.
Inserire la formula: [["a"],["-b","-c","f"],["-a","p","-b"],["b"]].
La formula inserita e' una formula di Horn!
L'assegnamento minimo che la soddisfa e':
[[a],[b],[p]]

true ?

(3 ms) yes
```

Test nr. 10

```
arlindpec@arlindpec:~/Scrivania/Progetto_PLF/Prolog$ ./SoddisfacibilitaFormulaHorn
GNU Prolog 1.5.0 (64 bits)
Compiled May 24 2023, 19:59:04 with gcc
Copyright (C) 1999-2023 Daniel Diaz

| ?- main.
------------------------------- Benvenuto -------------------------------
Il seguente programma verifica se una FBF in forma normale congiuntiva
della logica proposizionale e' una formula di Horn o meno e in caso af-
-fermativo stampa il minor assegnamento di verita' che la soddisfa.
La formula va inserita come una lista di liste di stringhe dove ogni lette-
rale è una stringa.
Per esempio se si vuole inserire (p) AND (NOT p OR q) AND (NOT p OR NOT z)
scriverai nell'input:
[["p"],["-p", "q"], ["-p", "-z"]]. <Invio>
Si ricorda il punto finale prima dell'invio e si raccomanda di seguire
questa convenzione per evitare comportamenti anomali del programma.
In caso la formula fosse di Horn, l'assegnamento di verita' sara' rap-
-presentato da una lista di proposizioni.
NOTA: [] denota l'insieme vuoto, [insoddis] denota che la formula e' una
contraddizione.
Inserire la formula: [].
La formula inserita e' una formula di Horn!
L'assegnamento minimo che la soddisfa e':
[]

true ?

yes
```