Esercizio 1

How can a rational agent choose the best next action to perform in an environment? Despite its ability to select the best move, why does this not necessarily drive success?

Un agente razionale seleziona la sua prossima azione con l'obiettivo di massimizzare il valore atteso della sua misura di prestazione (performance measure), basandosi sulla sequenza di percezioni ricevuta e sulla conoscenza predefinita (dell'ambiente) che possiede. Per modellare un agente razionale è essenziale specificare la sua descrizione PEAS (Performaance, Environment, Actuators, Sensors).
Nonostante l'agente cerchi di agire in modo "ottimo", non è garantito il successo, perché:

- Razionele != Onniscente: non si hanno a disposizione tutte le informazioni sul mondo, la conoscenza dell'agente non è perfetta, le azioni sono scelte in base all'esperenza personale; al contrario se non dovesse considerare le percezioni e le sue azioni ossero basate completamente sulla conoscenza predefinita, l'agente sarebbe privo di autonomia.
- Razionale != Chiaroveggente: non può prevedere il futuro. I risultati delle sue azioni potrebbero non essere quelli attesi, anche se ha scelto l'azione che riteneva migliore.

Quindi, l'agente razionale è definito dalla sua capacità di esplorare l'ambiente, apprendere ed agire con autonomia, piuttosto che dalla garanzia di successo in ogni circostanza.

Esercizio 2

Explain how the iterative deepening search strategy works, reporting its time and space complexity and describing its completeness and optimality.

La strategia di ricerca ad approfondimento iterativo è un metodo di ricerca non informata che combina i vantaggi della ricerca in profondità e quella in ampiezza.
Viene applicata in modo ripetuto la ricerca a profondità limitata, iniziando con un limite di profondità pari a 0 che viene incrementato proggressivamente ad ogni iterazione (0,1,2...) fino a quando non viene trovata una soluzione o non è stato rilevato un fallimento. Durante ogni iterazione la ricerca a profondità limitata esplora i nodi fino al limite di profondità corrente. Quando il limite di profondità viene incrementato per una nuova iterazione, il lavoro svolto nelle iterazioni precedenti viene "concettualmente" scartato, perché la ricerca riparte da zero con il nuovo limite. Nonostante possa sembrare uno spreco, la ri-generazione dei nodi nei livelli superiori ha un impatto limitato sulla complessità coplessiva, la maggior parte dei nodi (in molti spazi degli stati) si trova ai livelli più profondi.
Questa ricerca è completa se il fattore di diramazione "b" è finito, ed è ottimale se tutte le azioni hanno lo stesso costo. La complessità spaziale è pari a O(bd) con "d" profondità della soluzione più superficiale; quella temporale, invece, è pari a O(b^d).
La ricerca ad approfondimento iterativo è il metodo preferito, di ricerca non informata, quando lo spazio di ricerca è troppo ampio per essere mantenuto in memoria e la profondità della soluzione non è nota.

Esercizio 3

Model the Australia map coloring problem as a CSP. Use the forward-checking algorithm after each assignment. Finally, show a possible solution. Allowed colors are red, green, and blue. After, by using AC3, show that this problem is not solvable using only 2 colors.

Il roblema della colorazione della mappa dell'australia pò essere formalmente definito da tre componenti (quindi come un CSP, problema di soddisfacimento di vincoli):

- Variabili: X = {WA, NT, Q, NSW, V, SA, T}
- Domini: Di = {rosso, verde, blu}
- VIncoli: C = {SA != WA, SA != NT, SA != Q, SA != NSW, SA != V, WA != NT, NT != Q, Q != NSW, V != NSW}

L'algoritmo di forward-checking è una forma di inferenza che, dopo aver assegnato un valore ad una variabile "x", stabilisce la consistenza d'arc per essa. Questo significa che per ogni variabile non assegnata "y" collegata ad "x" da un vincolo, il forward-checking elimina dal dominio di "y" ogni valore che non sia consistente con quello scelto per "x".

Assengando D(WA) = {rosso} → D(NT) = {verde, blu} (perché NT != WA) → D(SA) = {verde, blu} (SA != WA); gli altri domini rimangono {rosso, verde, blu}.
Assegnando D(NT) = {verde} → D(SA) = {blu} (SA != NT) → D(Q) = {rosso, blu} (Q != NT); gli altri vincoli rimangono inalterati rispetto ai vincoli attivi.
Assegnando Q = {rosso} → D(NSW) = {verde, blu} (NSW != Q); gli altri domini rimangono inalterati.
Assegnando NSW = {verde} → D(V) = {rosso, blu} (V != NSW); gli altri domini sono inalterati.
Assegnando D(V) = {rosso}; gli altri vincoli rimangono inalterati
Assegna D(SA) = {blu}; tutte le variabili adiacenti hanno già colori assegnati che non entrano in conflitto con il blu.
Assegna D(T) = {verde} (T non ha vincoli con le altre regioni, potrebbe essere colorata con qualsiasi colore nel suo dominio).
Soluzione trovata: {WA = rosso, NT = verde, Q = rosso, NSW = verde, V = rosso, SA = blu, T = verde}.

Utilizzando l'AC-3 ogni variabile è già arco-consistente, quindi la consistenza d'arco per il problema di colorazione della mappa con due colori non serve a nulla. L'algoritmo con solo due colori nel dominio non produrrebbe alcuna riduzione di dominio e non rileverebbe che il problema è irrisorvibile (per ogni valore nel dominio di una variabile xi esiste un valore nel dominio della variabile xj ad essa collegata che soddisfa il vincolo xi != xj). Per rilevare lìinsolvibilità sono necessarie tecniche di consistenza di livello superiore (path consistency o la k-consistency).

Esercizio 4

What is the genetic algorithm? In what kind of search is it used? Please provide a description of how it works.

L'algoritmo genetico è una variante della local beam search stocastica. Si distingue perché, invece di modificare un singolo stato, genera statii successori combinando due stati genitori. L'algoritmo inizia con un insieme di "k" stati generrati casualmente (chiamato popolazione), ogni stato (individuo) è rappresentato da una stringa di simboli (ad esempio, bit o cifre). Ad ogni stato è associata una funzione di itness che valuta quanto è "buona" la sequenza di simboli, assegnando un valore maggiore agli stati migliori. Due coppie di stati (genitori) vengono selezionate casualmente dalla popolazione per la riproduzione, basandosi sulle proprietà di fitness. Viene scelto un punto di crossover casuale all'interno della stringa; i successori (prole) vengono creati combinando le parti separate dai punti di incrocio. Ogni prole può essere soggetta a mutazioni casuali. Il procedimento è iterato associando ad ogni

nuova sequenza una funzione di fitness ed una probabilità, ed il ciclo continua fino a quando non è trovata una soluzione.

Esercizio 5
What is an inference procedure? Describe how it is linked to logical entailment and describe the soundness and completeness of a procedure.

Una procedura di inferenza è un meccanismo algoritmico che permette di derivare nuove formule a partire da una KB (Knowledge Base) esistente. L'AI si occupa di sviluppare strumenti per la rappresentazione e l'elaborazione della conoscenza, e le procedure di inferenza sono fondamentali per il ragionameno automatico (di tipo deduttivo).

Un algoritmo di inferenza "i" può derivare (una formula) alpha (partendo) dalla KB.
Le basi di conoscenza sono insiemi di formule in un linguaggio formale (come logica proposizionale o del primo ordine) che includono un motore di inefrenza, il quale prende le informazioni codificate nella KB e ne deduce qualcosa di nuovo.

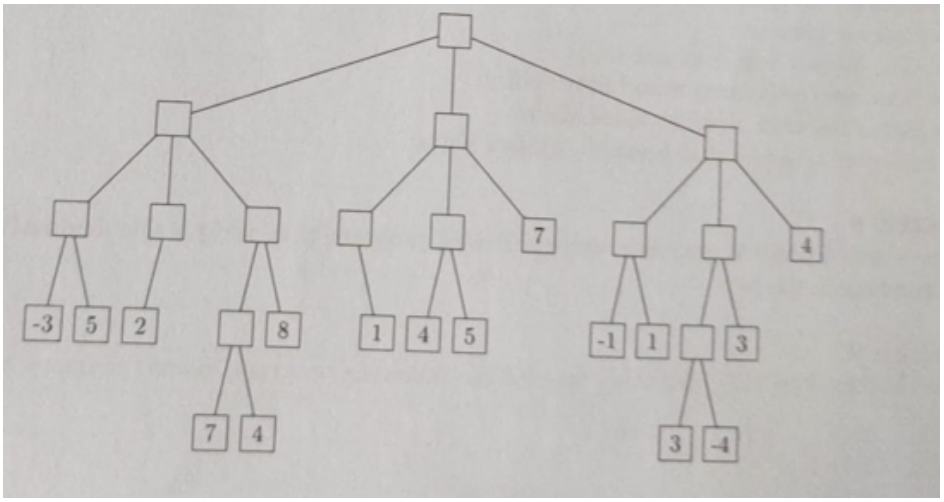
Entailment: l'implicazione logica si indica con KB |= alpha e significa che "in tutti i mondi in cui la base di conoscenza è vera, allora è vera anche alpha" (una segue logicamente l'altra). Matematicamente M(KB) \subset M(alpha); quindi KB è un asserzione più forte di alpha.

Se consideriamo KB come un pagliaio e alpha come un ago, la conseguenza logica dice che l'ago è nel pagliaio mentre l'inferenza lo trova.

Correttezza (Soundness): una procedura di infereza è corretta se deriva solo formule che sono consguenze logiche (esso preserva la verità).

Completezza (Completeness): una procedura di inferenza è completa se può derivare ogni formula che è conseguenza logica.

Esercizio 6



Consider the following game tree, where the scores are represented from the point of view of the first player. Suppose that the first player is MAX and use the min-max search with alpha-beta pruning to show the steps of the game.

alpha = valore della scelta migliore per MAX (valore più alto trovato finora)
beta = valore della scelta migliore per MIN (valore più basso trovato finora)
Potatura:
MAX si pota quando alpha >= beta;
MIN si pota quando beta <= alpha.

Entriamo con alpha = -inf e beta = +inf. Nel sottoalbero sinistro.
Il primo figlio di MAX ha foglie (sinistra) di valore -3 e 5, dato che tocca a max sceliere, sarà preso il valore più alto, quindi viene aggiornato beta = min(inf,5) = 5.
La successiva foglia è signola e di valore 2 → MIN aggiorna beta = min(5,2) = 2.
Il terzo figlio di MAX ha a sinistra un blocco di scelta MIN che porta a valori 7 e 4 → sarà scelto il 4; ed a destra una foglia di valore 8 → il massimo tra 4 ed 8 è 8 quindi beta = min(2,8) = 2 (rimane invariato).
Abbiamo finito il sottoalbero di sinistra che porta 2 alla radice → MAX aggiorna alpha = max(-inf,2) = 2.
Nel sottoalbero centrale si entra con alpha = 2 e beta = +inf.
Primo figlio di MAX ha una sola foglia con valore 1. MIN aggiorna beta = min(inf,1) = 1. Dato che beta <= alpha è effettuatua la potatura dei restanti due figli di questo MIN. Il sottoalbero centrale restituisce 1 → alpha rimane invariato.
Entriamo nel sottoalbero destro con alpha = 2 e beta = +inf.
Il primo figlio (MAX) ha foglie -1,1; sarà scelto il valore 1 → beta = 1 (aggiornato) → beta <= alpha allora si effettua la potatura dei restanti figli di questo MIN → il sottoalbero destro restituisce 1. alpha = max(2,1) = 2 (rimane invariato).

Dal sottoalbero di destra abbiamo il le prime foglie di valore -1,1 → MAX sceglierà 1 → MIN aggiorna beta = 1 → beta <= alpha → PRUNE the remaining 2 children of this MIN → il sottoalbero di destra ritorna 1 → alpha rimane = 2.

Risultato:
MAX scelgie il sottoalbero di sinistra, MIN sceglie la foglia nel mezzo (figlio con valore 2)
MAX ha una sola mossa (forzata) → esito: 2.
MAX può forzare un utilità pari a 2 (scegliendo il subtree di sinistra).

Vantaggio alpha-beta: Abbiamo potato 2 rami nel sottoalbero centrale ed in quello destro (perché beta è sceso sotto il valore di alpha).

Esercizio 7
Assume the following facts:

- *eva owns a dog*
- *kitty is a cat*
- *dogs are animals*
- *cats are animals*
- *who owns a dog loves animals*
- *who loves an animal would never kill it*
- *either susan or eva has killed kitty*

Use resolution to prove that Susan has killed Kitty.

Traduco in FOL:

1. \exist x Owns(EVA, x) \and Dog(x)
2. Cat(KITTY)
3. \forall x (Dog(x) → Animal(x))
4. \forall x (Cat(x) → Animal(x))
5. \forall x,y (Owns(y,x) \and Dog(x) → LovesAnimals(y))
6. \forall x,y (LovesAnimals(y) \and Animal(x) → - Kills(y,x)
7. Kills(SUSAN, KITTY) \or Kills(EVA, KITTY)

Q: Kills(SUSAN, KITTY)

Conversione in CNF:

1: Sostituiamo la x con una costante di Skolem per rimuovere il quantificatore esistenziale, ed otteniamo

- Owns(EVA, C); (1)
- Dog(C); (2)

2: Rimane invariata; (3)

3: dopo aver rimosso l'implicazione è equivalente a \forall x (- Dog(x) \or Animal(x)); rimuovendo anche il quantificatore universale

- - Dog(x) \or Animal(x); (4)

4: - Cat(x) \or Animal(x); (5)

5: \forall x,y (- (Owns(y,x) \and Dog(x)) \or LovesAnimals(y)); che diventa applicando De Morgan

- - Owns(y,x) \or - Dog(x) \or LovesAnimals(y); (6)

6: - LovesAnimals(y) \or - Animal(x) \or - Kills(y,x); (7)

7: rimane invariata; (8)

Negazione della Query: - Kills(SUSAN, KITTY); (9).

Risolvendo (2) e (4) con la sostituzione x = C:

(10) Animal(x)

Risolvendo (1) e (6), con la sostituzione y = EVA e x = C:

(11) - Dog(C) \or LovesAnimals(EVA)

Risolvendo (2) con (11):

(12) LovesAnimals(EVA)

Risolvendo (3) e (5) con l'unificazione x = KITTY:

(13) Animal(KITTY)

Risolvendo (12) e (7) con l'unificazione y = EVA:

(14) - Animal(x) \or - Kills(EVA, x)

Risolvendo (13) e (14) con la sostituzione x = KITTY

(15) - Kills(EVA, KITY)

Risolvendo la (8) e (9):

(16) Kills(EVA, KITTY)

Risolvendo (15) e (16) trovimo l'insieme vuoto, quindi la query è dimostrata.

Esercizio 8

Given a list of positive numbers, write a Prolog program that returns the summation of all the odd numbers in the list.

% predicato per verificare se un numari è dispari e positivo

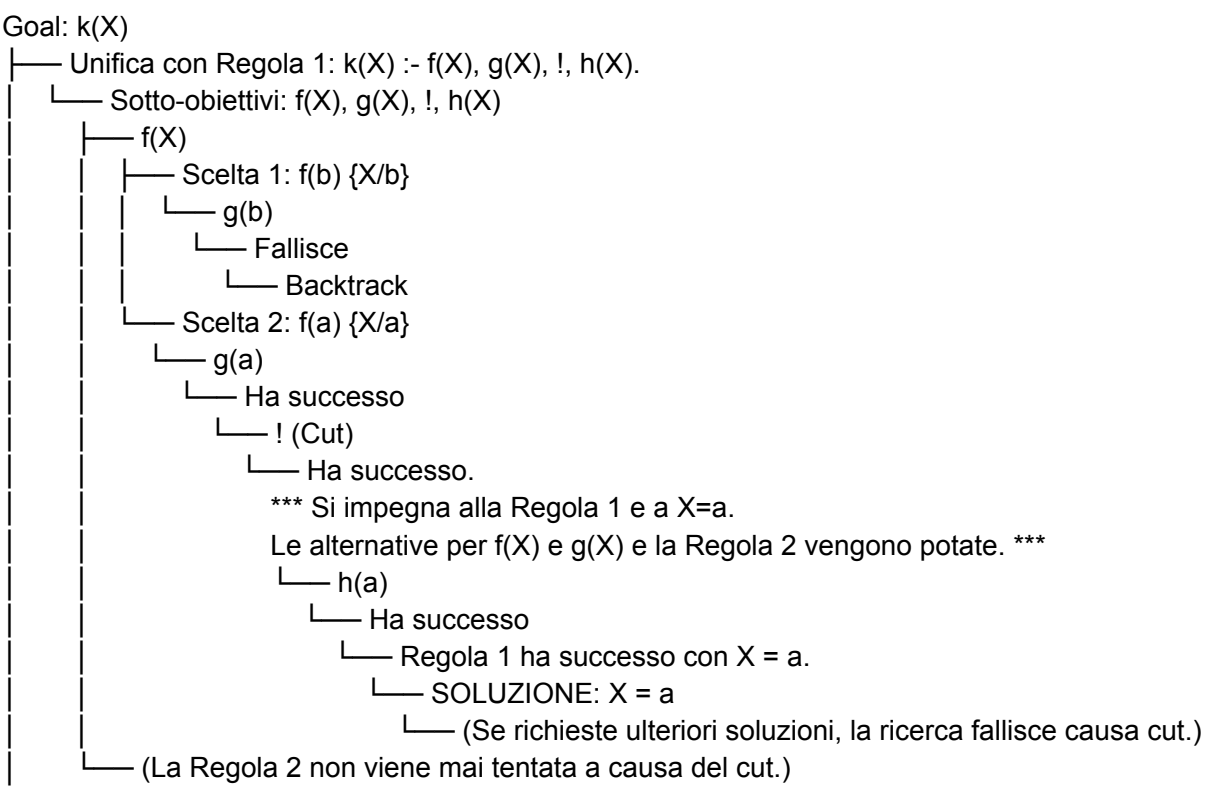
```
is_odd(X) :-
        X > 0, % verifichiamo che il numero sia positivo
        Y is X mod 2, % calcola il resto della divisione per 2
        Y =:= 1. % unifica con true se il resto è 1.

% predicato per sommare i numeri dispari nella lista (3 casi)
%CasoBase
sum_odd_numbers([], 0). % somma di tutti i numero dispari in una lista vuota è zero
%Regola1: se l'elemento in testa (H) è dispari, includilo nella somma.
sum_odd_numbers([H|T], Sum) :-
        is_odd(H),
        sum_odd_numbers(T, RestSum), % calcola la somma dei dispari nella coda
        Sum is H+RestSum. % aggiungi H alla somma della coda
%Regola2: se l'elemento in testa non è dispari, ignoralo.
sum_odd_numbers([H|T], Sum) :-
        \+ is_odd(H),
        sum_odd_numbers(T, Sum). % calcola la somma dei dispari nella coda (ignora H)
```

Esercizio 9

Show the complete SLD resolution tree for the following program, given the query k(X).

```
f(b).    f(a).    f(c).    h(b).
h(a).    h(c).
g(a).    g(d).
k(X) :- f(X), g(X), !, h(X).
k(X) :- g(X).
```



Esercizio 10
Suppose to have the following knowledge base KB:
AvB → C
A → B
A
explain what is modus ponens and how we can use it both in forward chaining and backward chaining to prove that KB |= B^C.

Il Modus Pones è una delle regole di inferenza fondamentali nella logica (noto anche come “eliminazione delle implicazioni”). Essa stabilisce che avendo una proposizione condizionale “a → b” e sapendo che “a” è vera, allora possiamo dedurre che anche “b” sia vera.

Il forward chaining è una strategia di ragionamento “data-driven”. Partendo dai fatti noti nella base di conoscenza attiva le regole le cui premesse sono soddisfatte. Le conclusioni di queste regole vengono aggiunte alla KB come nuovi fati ed il processo si ripete.
Nell’esempio:
Valutiamo la regola2 A → B, premessa A è un fatto noto nella KB, applicando il modus ponens possiamo dedurre B.
Valutiamo la regola1 A v B → C, A v B è vero quindi possiamo dedurre C.
Avendo dedotto sia B che C, la congiunzione B ^ C è vera.

Il backward chaining è una strategia di inferenza che parte dall’obiettivo (la query) e lavora a ritroso per trovare i fatti e le regole che supportano tale obiettivo. Ragionamento “query-driven” o “goal-driven”. L’algoritmo cerca implicazione nella KB che hanno l’obiettivo come conclusione. Se ne trova uno, tenta di dimostrare vere tutte le preesse di tale implicazione. Processo ricorsivo.
Nell’esempio:
dobbiamo dimostrare B e poi C.
primo sotto-obiettivo, dimostrare B. Troviamo la regola2 che ha B come conclusioni. A è un fatto dichiarato esplicitamente nella KB (ed è vero), quindi aplicando il modus ponens B è dimostrato.
secondo sotto-obiettivo, dimostrare C. Troviamo la regola1 che ha C come conclusione. Per dimostrare A v B è sufficiente dimostrare o A o B (questa disgiunzione è vera), applicando il modus ponens C è dimostrato.
Anche in questo caso abbiamo dimostrato che *KB |= B^C.*
