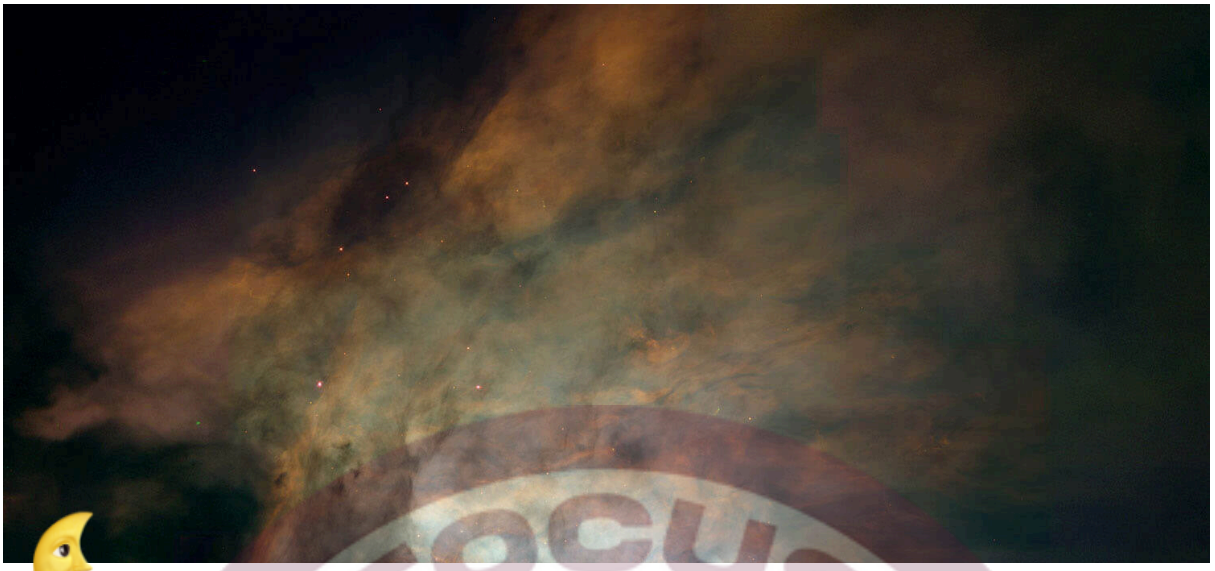

1. Introduzione e contesto

1.1 Storia e motivazioni dell'IA logica (Dartmouth, McCarthy, ragionamento non monotono)

Contesto storico

Nel 1956, John McCarthy conia il termine “Intelligenza Artificiale” durante il Dartmouth Workshop. L'obiettivo era ambizioso: sviluppare macchine intelligenti capaci di ragionamento umano. In particolare, McCarthy notò che:

- I sistemi intelligenti devono ragionare sul mondo usando **conoscenza incompleta**.
- Il **ragionamento comune** è spesso **non-monotono**, cioè nuove informazioni possono invalidare conclusioni precedenti.

Esempio 1 – Ragionamento classico (monotono)

```
flies(X) :- bird(X).
```

Se aggiungiamo:

```
bird(tweety).
```

allora `flies(tweety)` è vero. Ma se poi aggiungiamo:

```
penguin(tweety).
```

la regola iniziale continua ad applicarsi erroneamente, anche se sappiamo che i pinguini non volano.

Esempio 2 – Ragionamento non-monotono (ASP)

```
flies(X) :- bird(X), not abnormal(X).
abnormal(X) :- penguin(X).
bird(tweety).
penguin(tweety).
```

Qui `flies(tweety)` non è più derivabile, perché la presenza di `penguin(tweety)` attiva `abnormal(tweety)`, impedendo il volo.

Problematiche

- I sistemi classici (es. Prolog) non supportano nativamente la negazione non-monotona (`not`) in maniera stratificata.
- Il mondo reale non è completamente conosciuto e quindi serve **una logica che può cambiare idea**.

1.2 Differenza tra programmazione procedurale e dichiarativa

Procedurale

- Descrive **come** risolvere un problema, passo dopo passo.
- Esempio in pseudocodice:

```
for node in graph:
    for color in [red, green, blue]:
        if no_conflict(node, color):
            assign(node, color)
```

Pro:

- Controllo completo del flusso.

  Contro:

- Complessità alta per problemi combinatori (esplosione dello spazio di stati).

Dichiarativa (ASP)

- Descrive **cosa** deve essere vero in una soluzione, non come trovarla.

```
colore(X, rosso) | colore(X, verde) | colore(X, blu) :- nodo(X).
:- arco(X,Y), colore(X,C), colore(Y,C).
```

Questa rappresentazione:

- "Assegna a ogni nodo un colore."
- "Non permettere lo stesso colore ai nodi adiacenti."

Il sistema trova automaticamente le soluzioni valide (i cosiddetti “answer set”).

Esempio 1 – ASP

```
nodo(a). nodo(b). nodo(c).
arco(a,b). arco(b,c).

colore(X, rosso) | colore(X, verde) | colore(X, blu) :- nodo(X).
:- arco(X,Y), colore(X,C), colore(Y,C).
```

Esempio 2 – Procedurale equivalente (Python-like)

```
def is_valid(coloring, edges):
    for x, y in edges:
        if coloring[x] == coloring[y]:
            return False
    return True
```

Vantaggi della dichiaratività

- Meno codice.
- Separazione netta tra **modellazione** e **soluzione**.
- Maggiore espressività per problemi NP (Vertex Cover, 3-Coloring, Scheduling...).

Limitazioni

- Debug più difficile: non si controlla il processo, solo il risultato.
- Ottimizzazione: può richiedere aggiunte come "weak constraints" o strategie di propagazione specifiche.