

Paradigma Logico

Risoluzione e Prolog

Introduzione al paradigma logico

- Fa parte del paradigma dichiarativo insieme al paradigma funzionale
- Basato sulla logica formale (logica del primo ordine)
- PROLOG è il linguaggio logico più noto e utilizzato
- Computazione interpretata come deduzione logica (Kowalski)
- Usa la regola inferenziale della **risoluzione**
- Programmatore specifica solo la logica, il controllo è implicito
- Linguaggi logici: puri e con aspetti imperativi (come PROLOG)
- Adatto per problemi complessi, ricerca, IA, vincoli

Deduzione come computazione

- Motto di Kowalski: **Algoritmo = Logica + Controllo**
- Specifica logica (**cosa** fare) distinta dalla specifica del controllo (**come** fare)
- Nei linguaggi logici, il programmatore fornisce principalmente la logica
- Il controllo della computazione viene delegato alla macchina astratta
- La computazione corrisponde alla ricerca di dimostrazioni
- Uso della regola di risoluzione per la dimostrazione automatica
- PROLOG permette di influenzare il controllo con costrutti specifici
- Potenziale inefficienza del calcolo basato esclusivamente sulla logica

Cenni storici sul paradigma logico

- Anni '30: Lavori pionieristici di Gödel ed Herbrand
- Herbrand: prime idee sull'**unificazione**
- Anni '60: definizione formale della risoluzione (Alan Robinson)
- Anni '70: realizzazione del linguaggio PROLOG (Colmerauer, Kowalski)
- Da PROLOG derivano numerose versioni e dialetti
- Introduzione dei linguaggi logici con vincoli (Constraint Logic Programming)
- Importanza attuale in intelligenza artificiale e teoria della computazione

Linguaggi Logici e le loro Caratteristiche

- PROLOG è il linguaggio più diffuso e implementato
- Altri linguaggi: Datalog (database), Mercury (ottimizzazione statica), CLP (vincoli)
- Linguaggi logici si fondano su inferenza e rappresentazione della conoscenza
- Progettati per essere altamente espressivi, ma con controllo del flusso limitato

Esempio: Problema delle Triplette Numeriche

- Obiettivo: trovare una lista di 27 numeri che rispetti vincoli precisi
- Ogni numero i ($1 \leq i \leq 9$) appare esattamente 3 volte
- Tra ogni due occorrenze di i ci devono essere esattamente i elementi
- Esempio di come la logica può esprimere il problema in modo compatto

Parte di un Programma PROLOG

```
sol(Ls):-  
    list_of_27(Ls),  
    sublist([1,_,1,_,1], Ls),  
    sublist([2,_,_,2,_,_,2], Ls),  
    sublist([3,_,_,_,3,_,_,_,3], Ls),  
    ...  
    sublist([9,...,9,...,9], Ls).
```

Deduzione come Computazione

- In PROLOG, specifiche sono automaticamente interpretabili
- L'interprete cerca una prova logica per soddisfare un goal
- Il meccanismo è guidato dalla risoluzione e backtracking
- Il programma può produrre più soluzioni tramite ricerca

Sintassi della logica del prim'ordine: Alfabeto

- Linguaggi basati sulla logica del primo ordine (predicate calculus)
- **Simboli logici** (fissi per tutti i linguaggi):
 - Connettivi logici: \wedge , \vee , \neg , \rightarrow , \leftrightarrow
 - Quantificatori: \forall (forall), \exists (exists)
 - Costanti logiche: true, false
 - Variabili: lettere maiuscole (es: X, Y, Z)
 - Punteggiatura: parentesi (), virgole ,
- **Simboli non-logici** (specifici per applicazioni):
 - Predicati (es: padre(X,Y) significa "X è padre di Y")
 - Funzioni (es: f(a,b))
 - Costanti (es: a, b, mario, luca)

Sintassi (2): Termini

- Termini: elementi base della sintassi
- Variabile singola è un termine (es: X , Y)
- Una costante è un termine (es: a , b , $mario$)
- Applicazione di una funzione ai termini è un termine (es: $f(a,b)$)
- Esempi pratici di termini composti:
 - $f(X, Y)$
 - $g(a, X)$
 - $h(f(a), g(Y,b))$
- Termini ground: senza variabili (es: $f(a,b)$, $g(a,a)$)
- Termini utilizzati come strutture dati (liste, alberi)
- Un singolo tipo per tutti i termini
- Esempio lista in PROLOG: $[a, b, c]$ o $[X|T]$

Sintassi (3): Formule ben formate (Formulæ)

- Atomi: applicazione di un predicato ai termini
 - Es: `fratello(X,Y)`, `maggiore_di(3,2)`
- Formule complesse ottenute dai connettivi logici:
 - Es: `fratello(X,Y) ∧ fratello(Y,Z)`
- Quantificazioni con variabili:
 - Es: $\forall X \exists Y \text{ padre}(Y,X)$
- Esempio proprietà transitiva:
 - $>(X,Y) \wedge >(Y,Z) \rightarrow >(X,Z)$
- true e false formule costanti
- Uso di parentesi per chiarezza sintattica (es: $(A \wedge B) \rightarrow C$)

Sintassi (4): Clausole e programmi logici

- **Clausola definita:** $H :- A_1, A_2, \dots, A_n$ con $n \geq 0$
- Testa (H) e corpo (A_1, \dots, A_n), con corpo vuoto per i fatti
- Interpretazione logica $A_1 \wedge A_2 \wedge \dots \wedge A_n \rightarrow H$
- Esempio di clausola definita:

`genitore(X,Y) :- padre(X,Y)`

`antenato(X,Y) :- genitore(X,Y)`

`antenato(X,Y) :- genitore(X,Z), antenato(Z,Y)`

- **Fatti:** clausole con corpo vuoto

`padre(luca,mario)`

`padre(mario,giulia)`

- Programma logico: insieme di clausole definite
- Goal (query): sequenza di atomi da dimostrare

Esempi di query (goal)

```
genitore(X,Y) :- padre(X,Y)
antenato(X,Y) :- genitore(X,Y)
antenato(X,Y) :- genitore(X,Z), antenato(Z,Y)
padre(luca,mario)
padre(mario,giulia)
```

Esempi di goal:

- ?- genitore(X, mario)
- ?- genitore(giulia, Y)
- ?- antenato(luca, Y)
- ?- antenato(X, giulia)

Variabili logiche e unificazione

- Per applicare una clausola, necessario unificare la query con l'head della clausola
 - istanziando variabili logiche con termini
- Variabile logica può essere istanziata una sola volta
- Può essere associata a termini non ground
- Binding bidirezionale (simmetrico) con unificazione
- Unificazione: soluzione di equazioni tra termini
- Esempio di unificazione:
 - Equazione: $f(X, b) = f(a, Y)$
 - Soluzione: $\{X/a, Y/b\}$
- Meccanismo alla base del calcolo nei linguaggi logici

Algoritmo di Martelli-Montanari

- Serve per unificare **termini** logici.
- Determina se un insieme di equazioni tra termini è risolvibile.
- Produce una **forma risolta** da cui si ottiene la **most general unifier (MGU)**.
- Obiettivo dell'algoritmo
 - Dato un insieme di equazioni $\{s_1 = t_1, \dots, s_n = t_n\}$:
 - Restituire la MGU θ tale che $s_i\theta = t_i\theta$ per ogni i , oppure
 - Segnalare **fallimento** se non unificabili.
 - MGU: la sostituzione **più generale** tra tutte quelle che unificano i termini.

Definizione di Sostituzione

Una **sostituzione** (in logica e programmazione logica) è una funzione che associa **variabili** a **termini**.

Formalmente, una sostituzione si scrive come:

$$\theta = \{ X_1/t_1, X_2/t_2, \dots, X_n/t_n \}$$

dove:

- X_1, \dots, X_n sono **variabili** distinte.
- t_1, \dots, t_n sono **termini** (che possono essere costanti, variabili o funzioni composte).

Esempio

Sia:

$$\theta = \{ X/a, Y/f(b), Z/Y \}$$

Allora:

- $\theta(X) = a$
- $\theta(Y) = f(b)$
- $\theta(Z) = Y$ (e se si continua: $\theta(Z) = f(b)$)

Applicazione di una sostituzione

Dato un termine t e una sostituzione θ , si denota con $t\theta$ il termine ottenuto **sostituendo tutte le occorrenze** delle variabili di t secondo θ .

Esempio:

- Termine: $p(X, g(Y))$
- Sostituzione: $\theta = \{ X/a, Y/b \}$
- Risultato: $p(X, g(Y))\theta = p(a, g(b))$

Composizione di sostituzioni

Dati due sostituzioni:

$$\theta = \{ X/a \}, \quad \sigma = \{ Y/X \}$$

La **composizione** $\theta \circ \sigma$ è:

$$\{ Y/a, X/a \}$$

La sostituzione σ è più generale di $\theta \circ \sigma$

- Forma dell'output:

$$\{ X_1 = r_1, \dots, X_n = r_n \}$$

- Con X_i variabili **distinte**, e r_i termini che **non contengono** X_i .
- L'output definisce la sostituzione:

$$\{ X_1/r_1, \dots, X_n/r_n \}$$

Regole dell'algoritmo

Per ogni equazione $s = t$, si applica una delle seguenti:

- 1 $f(s_1, \dots, s_n) = f(t_1, \dots, t_n) \rightarrow$ decomposizione:
 $\{s_1 = t_1, \dots, s_n = t_n\}$
- 2 $f(\dots) = g(\dots)$ con $f \neq g \rightarrow$ **fallimento**
- 3 $X = X \rightarrow$ eliminazione
- 4 $X = t$ con X non in $t \rightarrow$ sostituzione:
 - Applica $\{X/t\}$ a tutto l'insieme
- 5 $X = t$ con X in $t \rightarrow$ **fallimento** (occurs-check)
- 6 $t = X$ con t non variabile \rightarrow riscrivi come $X = t$

Esempio: Unificazione semplice

Insieme iniziale:

$$E = \{ f(X, b) = f(g(Y), W), h(X, Y) = h(Z, W) \}$$

Passo 1: selezionando l'equazione

$$f(X, b) = f(g(Y), W)$$

otteniamo:

$$X = g(Y), b = W$$

nuovo insieme di equazioni:

$$E = \{ X = g(Y), b = W, h(X, Y) = h(Z, W) \}$$

Passi successivi: applica sostituzioni

Da

$$E = \{ X = g(Y), b = W, h(X, Y) = h(Z, W) \}$$

applicando le sostituzioni

- $X \rightarrow g(Y)$
- $W \rightarrow b$

Otteniamo:

$$E = \{ X = g(Y), W = b, h(g(Y), Y) = h(Z, b) \}$$

Selezionando:

$$h(g(Y), Y) = h(Z, b)$$

otteniamo:

$$E = \{ X = g(Y), W = b, g(Y) = Z, Y = b \}$$

Da

$$E = \{ X = g(Y), W = b, g(Y) = Z, Y = b \}$$

applicando le sostituzioni:

$$E = \{ X = g(b), W = b, Z = g(b), Y = b \}$$

Formula risolta e **MGU finale**:

$$\{ X/g(b), W/b, Z/g(b), Y/b \}$$

Considerazioni

- Algoritmo **incrementale**: ogni passo riduce la complessità dei termini coinvolti nelle equazioni.
- Permette valori **parziali** che vengono **raffinati**.
- Unificatore ottenuto è il più generale possibile.
- È alla base del **motore di inferenza** dei linguaggi logici.

Esempio:

$X = f(X)$

- X appare **dentro** al termine a cui è assegnata.
- Porta a **loop infinito** → fallimento.
- Alcune implementazioni (come PROLOG) **saltano** questo controllo per efficienza.

Risoluzione SLD

- Metodo inferenziale base della programmazione logica
- “Selection rule-driven Linear resolution for Definite clauses”
- Parte da goal iniziale e programma
- Risolve goal scegliendo clausole appropriate (con unificazione)
- Sequenza di applicazioni produce dimostrazione (SLD refutation)
- Dimostrazione vuota indica successo
- Sostituzione finale (answer substitution) è il risultato
- Dimostrazione per backtracking: cerca alternative se fallisce

Esempio di programma PROLOG

- Definizione semplice di “antenato”:

```
antenato(X,Y):- genitore(X,Y)
```

```
antenato(X,Y):- genitore(X,Z), antenato(Z,Y)
```

- Fatti definiti:

```
genitore(luca, mario)
```

```
genitore(mario, giulia)
```

- Query di esempio:

```
?- antenato(luca, giulia)
```

- Risposta attesa: true

Universo di Herbrand

- L'universo di Herbrand: insieme di tutti i termini costruibili da un insieme di simboli
- I termini costruiti solo a partire da costanti e funzioni (senza variabili) sono **ground**
- Esempio semplice:
 - Costanti: a
 - Funzioni: f
 - Universo: $\{a, f(a), f(f(a)), f(f(f(a))), \dots\}$
- Termini rappresentano dati e strutture dati nei linguaggi logici

Interpretazione dichiarativa e procedurale

- Doppia interpretazione dei programmi logici:
 - **Dichiarativa**: significato logico del programma
 - **Procedurale**: esecuzione del programma (chiamate procedurali)
- Clausola logica dichiarativa:
 - `antenato(X,Y) :- genitore(X,Y)`
 - Significa: “Se X è genitore di Y, allora X è antenato di Y”
- Interpretazione procedurale:
 - Per trovare `antenato(luca,Y)` cerca `genitore(luca,Y)`

Chiamate procedurali e unificazione

- Chiamata procedurale è una richiesta di dimostrazione di un goal
- Clausola: $p(X) :- q(X)$ Goal: $p(a)$ Si valuta $q(a)$ (passaggio di parametri tramite unificazione)
- Se esiste una clausola compatibile, continua la computazione
- Se non esiste clausola compatibile, la chiamata fallisce (fallimento)

- Regola inferenziale centrale per la computazione logica
- Derivazione SLD produce sequenza di goal sempre più semplici
- Refutazione SLD termina con goal vuoto (successo)
- Risultato: sostituzione (answer substitution) dei valori ai parametri

Meccanismo di backtracking

- Computazione logica non deterministica
- In caso di fallimento, torna indietro a esplorare alternative
- Consente ricerca esaustiva delle soluzioni
- Potenziale problema: inefficienza dovuta all'esplosione combinatoria

Sintassi avanzata: liste in PROLOG

- Sintassi standard: [a,b,c]
- Sintassi ricorsiva: [Head | Tail]
- Esempio di manipolazione liste:

```
membro(X, [X|_])
```

```
membro(X, [_|T]) :- membro(X,T)
```

- Uso tipico nelle applicazioni pratiche (parsing, strutture dati)

Variabili locali e globali

- Variabili locali: appaiono solo nel corpo delle clausole
- Variabili globali: appaiono anche nella testa della clausola
- Esempio:

`antenato(X,Y) :- genitore(X,Z), antenato(Z,Y)`

(Z è variabile locale, X e Y globali)

Clausole definite e clausole generali

- Clausole definite (solo atomi positivi):
 - Forma: $H :- A_1, \dots, A_n$
- Clausole generali (logica classica):
 - Forma con eventuali letterali negativi: $A \vee \neg B \vee \neg C$ (che è equivalente a $A :- B, C$)
- PROLOG tratta principalmente clausole definite

- Algoritmo di Martelli-Montanari:
 - Risolve sistemi di equazioni tra termini
 - Produce la forma risolta $\{X_1=t_1, \dots, X_n=t_n\}$
 - Genera il Most General Unifier (MGU)
- Non deterministico: ordini differenti producono risultati equivalenti

Forma risolta dell'unificazione

- Esempio di unificazione:

$$\{f(X,b)=f(g(Y),W), h(X,Y)=h(Z,W)\}$$

- Forma risolta (MGU):

$$\{X=g(b), Y=b, W=b, Z=g(b)\}$$

- Risultato ottenuto applicando progressivamente l'algoritmo

Aspetti avanzati del controllo

- Non determinismo intrinseco:
 - Nessun ordine obbligato per scegliere le clausole
- In PROLOG, ordine delle clausole e degli atomi è significativo
- Uso di “cut” (!) in PROLOG per ridurre backtracking
- Influenza fortemente efficienza e semantica operativa

Funzioni predefinite in PROLOG

- Operazioni aritmetiche: `is`, `+`, `-`, `*`, `/`

`X is 2+3`

- Predicati speciali: `write`, `read`, `assert`, `retract`
- Funzioni per input/output e modifica dinamica dei programmi

- Eleganza e semplicità dichiarativa contro inefficienza computazionale
- Ottimizzazioni possibili:
 - Uso intelligente di clausole
 - Riduzione dello spazio di ricerca tramite cut
 - Uso di predicati built-in ottimizzati

Introduzione alla programmazione con vincoli (CLP)

- Estensione della programmazione logica
- Gestione esplicita di vincoli:
 - Numerici (CLP(R), CLP(Q))
 - Su insiemi finiti (CLP(FD))

- Esempio:

$X + Y \# = 10, X \# > 0, Y \# > 0$

- Maggiore efficienza per certi tipi di problemi

Applicazioni pratiche del paradigma logico

- Intelligenza artificiale (pianificazione, ragionamento automatico)
- Linguaggi naturali (parsing e generazione)
- Database deduttivi
- Sistemi esperti basati su regole
- Problemi di ottimizzazione e scheduling

Vantaggi e svantaggi del paradigma logico

Vantaggi:

- Programmi chiari, semplici e dichiarativi
- Adatto a problemi di ricerca e IA
- Separazione di logica e controllo

Svantaggi:

- Potenziale inefficienza
- Gestione del controllo complessa senza esperienza
- Difficoltà di debugging rispetto ai linguaggi imperativi

Conclusioni generali

- Il paradigma logico offre un approccio alternativo alla programmazione
- È utile per modellare problemi complessi in modo dichiarativo
- Non sostituisce altri paradigmi, ma li integra in situazioni adatte
- Importante acquisire competenze anche su aspetti procedurali e di efficienza