ANNO ACCADEMICO 2024/2025

Intelligenza Artificiale e Laboratorio

Teoria

Altair's Notes

UNIVERSITÀ DI TORINO

DIPARTIMENTO DI INFORMATICA

Premessa

Licenza

Formato utilizzato

Box di "Concetto sbagliato":

Concetto sbagliato 0.1: Testo del concetto sbagliato

Testo contente il concetto giusto.

Box di "Corollario":

Corollario 0.0.1 Nome del corollario

Testo del corollario. Per corollario si intende una definizione minore, legata a un'altra definizione.

Box di "Definizione":

Definizione 0.0.1: Nome delle definizione

Testo della definizione.

Box di "Domanda":

Domanda 0.1

Testo della domanda. Le domande sono spesso utilizzate per far riflettere sulle definizioni o sui concetti.

Box di "Esempio":

Esempio 0.0.1 (Nome dell'esempio)

Testo dell'esempio. Gli esempi sono tratti dalle slides del corso.

Box di "Note":

Note:-

Testo della nota. Le note sono spesso utilizzate per chiarire concetti o per dare informazioni aggiuntive.

Box di "Osservazioni":

Osservazioni 0.0.1

Testo delle osservazioni. Le osservazioni sono spesso utilizzate per chiarire concetti o per dare informazioni aggiuntive. A differenza delle note le osservazioni sono più specifiche.

1 Introduzione

1.1 Il Corso in Breve...

1.1.1 Motivazioni

Definizione 1.1.1: Intelligenza Artificiale

L'intelligenza artficiale (o IA, dalle iniziali delle due parole, in italiano) è una disciplina appartenente all'informatica che studia i fondamenti teorici, le metodologie e le tecniche che consentono la progettazione di sistemi hardware e sistemi di programmi software capaci di fornire all'elaboratore elettronico prestazioni che, a un osservatore comune, sembrerebbero essere di pertinenza esclusiva dell'intelligenza umana.

Note:-

Meh, in realtà l'IA è una disciplina di confine. Però le tematiche sono prettamente informatiche.

IA In breve:

- Area di ricerca dell'informatica.
- Si occupa di tutto ciò che serve per rendere un computer intelligente come un essere umano.
- Interessata a problemi *intelligenti*: problemi per cui non esiste/non è noto un algoritmo di risoluzione[1].

Note:-

Il cubo di Rubik non è un gioco intelligente >:(

Ci sono tante sotto-aree di ricerca:

- Rappresentazione della conoscenza e ragionamento.
- Interpretazione/sintesi del linguaggio naturale.
- Apprendimento automatico.
- Pianificazione.
- Robotica.

[1] Tris, il labirinto, etc.

Si collega a tante discipline, oltre all'informatica:

- Filosofia.
- Fisica.
- Psicologia.

Questo insegnamento ha l'obiettivo di approfondire le conoscenze di Intelligenza Artificiale con particolare riguardo alle capacità di un agente intelligente di fare *inferenze* sulla base di una *rappresentazione esplicita della conoscenza* sul dominio. In questo corso si faranno anche sperimentazione di metodi di ragionamento basati sul paradigma della *programmazione logica*, sull'uso di *formalismi a regole* (CLIPS) e su *reti bayesiane* (ragionamento probabilistico[2]).

Programma:

- Dal punto di vista metodologico saranno a rontate problematiche relative a:
  - Meccanismi di ragionamento per calcolo dei predicati del primo ordine.
  - Programmazione logica.
  - Ragionamento non monotono.
  - Answer set programming.
- Queste metodologie verranno a rontate dal punto di vista sperimentale con l'introduzione dei principali costrutti del *Prolog*, lo sviluppo di strategie di ricerca in Prolog e l'utilizzo dell'ambiente *CLINGO* nella risoluzione di problemi in cui sia necessaria l'applicazione di meccanismi di ragionamento non monotono e del paradigma dell'Answer Set Programming.

Domanda 1.1

E le novità dell'AI che vanno di moda?

Risposta: vengono trattate in altri corsi (TLN, RNDL, AAUT, ELIVA, AGINT).

[2] Odio la probabilità con tutto il mio cuore <3

2 Il PROLOG

Definizione 2.0.1: PROLOG

PROLOG (Programming Logic) è un *linguaggio dichiarativo* basato sul *paradigma logico*:

- Non si descrive cosa fare per risolvere un problema.
- Si descrive la situazione reale con *fatti* e *regole* e si chiede all'interprete di verificare se un *goal* segue oppure no secondo una logica classica.

Note:-

Il PROLOG è equivalente alla logica dei predicati del primordine.

2.1 Le Basi

Definizione 2.1.1: Fatti

Si rappresenta con dei *fatti* un dominio di interesse.

Esempio 2.1.1 (Fatto)

Fatto per descrivere che un alimento contiene più calorie di un altro:

- piuCalorico(wurstel, banana).
- Rappresenta il fatto che il würstel è un alimento maggiormente calorico rispetto alla banana.

Definizione 2.1.2: Regole

Si rappresentano le possibili inferenze con delle *regole*:

```
head := subgoal1, subgoal2, ..., subgoaln
```

Esempio 2.1.2 (Regola)

```
felino(X) := gatto(X)
```

Rappresenta la regola che permette di concludere che i gatti sono felini.

Idee di base del PROLOG:

- Regole ricorsive.
- L'interprete analizza i fatti e le regole nell'ordine in cui si trovano nel programma.
- Meccanismo di pattern matching per uni care variabili e termini.
- L'interprete, dato un programma, cerca di dimostrare un goal considerando fatti e applicando regole, nel secondo caso generando sotto-goal.

Definizione 2.1.3: Clausole

Le clausole sono i fatti o le regole. Contengono:

- Atomi:
  - Costanti.
  - Numeri.
- Variabili.
- Termini Composti, ottenuti applicando funtori a termini.

Note:-

Un programma PROLOG è un insieme di clausole.

Osservazioni 2.1.1

- L'estensione dei file PROLOG è 'pl'.
- In PROLOG le variabili hanno l'iniziale maiuscola.
- L'unica struttura dati nativa è la lista.
- Per eseguire swi: swipl.
- Per compilare: ['nomefile.pl'].
- Il comando ';' indica possibili alternative.
- Il comando 'trace.' consente un esecuzione passo per passo.
- L'ordine è importante perché PROLOG "legge" dall'alto verso il basso.

Qualche predicato *built-in*:

- `var(X)`: indica se `X` è una variabile.
- `ground(X)`: indica se `X` è istanziata.
- `atom(X)`: indica se `X` è atomica.

2.1.1 Liste

Definizione 2.1.4: Lista

La *lista* è la struttura dati principale in PROLOG. Una lista è caratterizzata da una testa e da una coda:

- Testa: primo termine (a sinistra) della lista.
- Coda: la lista dei termini dal secondo (incluso) in poi.

Note:-

Rappresentata come [Head | Tail].

```
?- [1,2,3,4,5] = [Head | Tail].
Head = 1
Tail = [2,3,4,5] = [Head | Tail]
Yes
```

```
?- [a, ciao, [], 2, [1, saluti]] = [Head | Tail].
Head = a
Tail = [ciao, [], 2, [1, saluti]]
Yes
```

Figure 2.1: Le liste in PROLOG.

Predicati *built-in*:

- `length(Lista, N)`: ha successo se la `Lista` contiene `N` elementi.
- `member(Elemento,Lista)`: ha successo se la `Lista` contiene il termine `Elemento`.
- `select(Elemento,Lista,Rimanenti)`: rimuove `Elemento` da `Lista` e restituisce `Rimanenti`.

2.2 Interprete PROLOG

Domanda 2.1

Come avviene l'esecuzione di programmi PROLOG?

- Esecuzione mediante *backward chaining* in profondità.
- Si parte dal *goal* che si vuole derivare:
  - *Goal* = congiunzione di formule atomiche $G_1, G_2, \ldots, G_n$.
  - Si vuole dimostrare, mediante risoluzione, che il goal segua logicamente dal programma.
- Una regola $A :- B_1, B_2, \ldots, B_m$ è applicabile a G_i se:
  - Le variabili vengono rinominate.
  - A e G_i unificano.

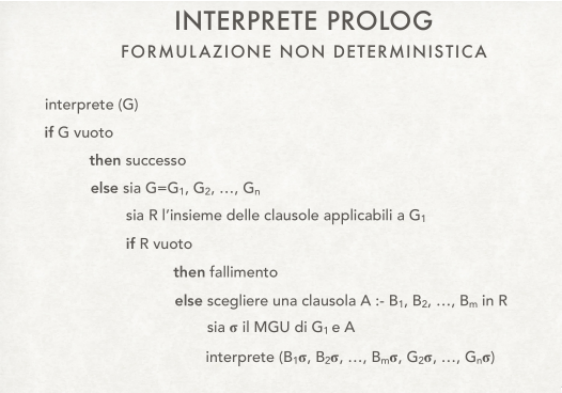


Figure 2.2: Una formulazione non deterministica di come funziona l'interprete PROLOG.

Note:-

MGU è il Most General Unifier: minimo sforzo per rendere uguali due variabili (il fatto e il goal).

- La computazione ha successo se esiste una computazione che termina con successo.
- Non determinismo: non è specificata la regola scelta in R.
- Ma l'interprete PROLOG si comporta in modo *deterministico*:
  - Le clausole vengono considerate nell'ordine in cui sono scritte nel programma.
  - Viene fatto backtracking all'ultimo punto di scelta ogni volta che la computazione fallisce.
- In caso di successo, l'interprete restituisce una sostituzione per le variabili che compaiono nel goal.

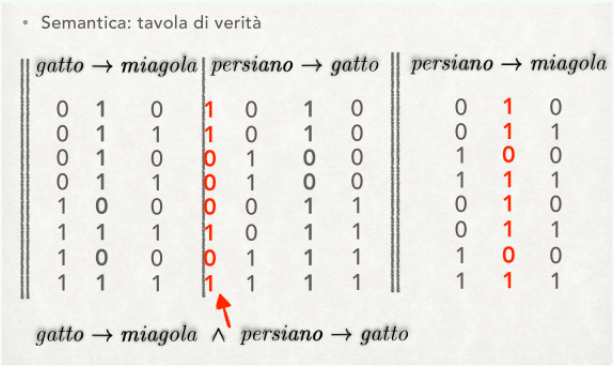
2.2.1 Breve Ripasso di Logica

Definizione 2.2.1: Logica Classica

Conseguenza logica definita semanticamente: dato una teoria e una formula, diciamo che la formula segue dalla teoria se essa è vera in tutti i modelli della teoria.

Esempio 2.2.1 (Gatti)

- I gatti miagolano: gatto $\rightarrow$ miagola.
- I persiani sono gatti: persiano $\rightarrow$ gatto.
- Si vuole dimostrare che i persiani miagolano: k $\vDash$ persiano $\rightarrow$ miagola.

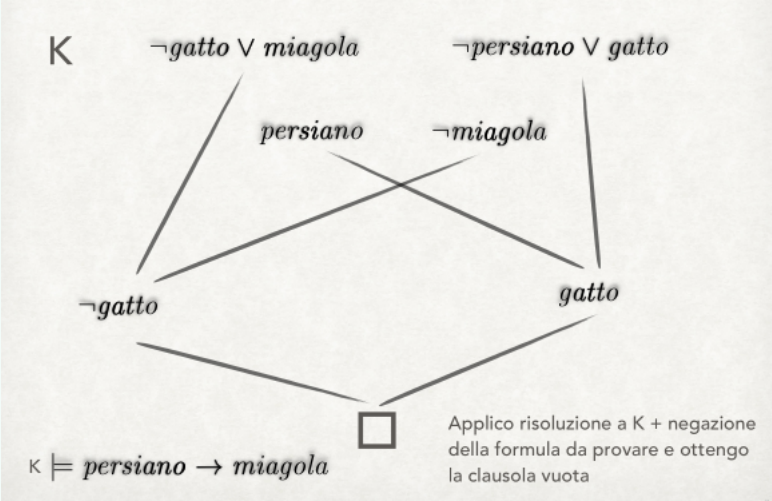


- Tuttavia il processo è molto laborioso già con poche formule e basi di conoscenza piccole.
- Metodo di prova: procedura/algoritmo che calcola/dimostra se una formula è conseguenza logica della teoria.
  - *Corretto*: se l'algoritmo dimostra F da K, allora F è conseguenza logica di K.
  - *Completo*: se F è conseguenza logica di K, allora l'algoritmo dimostra F da K.

Risoluzione:

- Si applica a formule in forma di *clausole* (disgiunzioni di letterali[1]).
- Si basa su un'unica regola di inferenza:
  - Date due clausole $C_1 = A_1 \vee \cdots \vee A_n$ e $C_2 = B_1 \vee \cdots \vee B_m$.
  - Se ci sono due letterali A_i e B_j tali che $A_i = \neg B_j$, allora posso derivare la clausola *risolvente* $A_1 \vee \ldots A_{i-1} \vee A_{i+1} \vee \cdots \vee A_n \vee B_1 \vee \ldots B_{j-1} \vee B_{j+1} \vee \cdots \vee B_m$.
  - Il risolvente è conseguenza logica di $C_1 \cup C_2$
- Data una teoria (insieme di formule) K e una formula F, dimostro che F è conseguenza logica di K per refutazione (dimostrare che $K \cup \neg F$ è inconsistente).
- Si parte dalle clausole $K \cup \neg F$, risolvendo a ogni passo due clausole e aggiungendo il risolvente all'insieme di clausole.
- Si conclude quando si ottiene la clausola vuota.

Esempio 2.2.2 (Risoluzione gatti)



Inoltre:

- Se le due clausole $C_1 = A_1 \vee \cdots \vee A_n$ e $C_2 = B_1 \vee \cdots \vee B_m$ contengono variabili, i due letterali A_i e B_j devono essere tali che si possa fare l'*unificazione* tra i due:
  - Unificazione: sostituzione α di variabili con termini o uguaglianza di variabili affinché $A_i = \neg B_j$.
  - Clausola risolvente $[A_1 \vee \ldots A_{i-1} \vee A_{i+1} \vee \cdots \vee A_n \vee B_1 \vee \ldots B_{j-1} \vee B_{j+1} \vee \cdots \vee B_m]\alpha$.
  - Le sostituzioni di α sono applicate a $A_1 \vee \ldots A_{i-1} \vee A_{i+1} \vee \cdots \vee A_n \vee B_1 \vee \ldots B_{j-1} \vee B_{j+1} \vee \cdots \vee B_m$.

[1] Formule atomiche o negazione di formule atomiche.

	costante c_2	variabile x_2	composto s_2
costante c_1	unificano se $c_1 = c_2$	unificano con x_2/c_1	non unificano
variabile x_1	unificano con x_1/c_2	unificano con x_1/x_2	unificano con x_1/s_2
composto s_1	non unificano	unificano con x_2/s_1	unificano se il funtore in s_1 e s_2 è lo stesso e gli argomenti unificano

Figure 2.3: Unificazione di due termini.

Note:-

Per ragioni d'efficienza, PROLOG non fa *occur check*, ossia una variabile X unifica con f(X).

2.2.2 Risoluzione SLD

Per arrivare a un linguaggio di programmazione PROLOG si vuole una strategia efficiente.

Definizione 2.2.2: Risoluzione SLD

Linear resolution with Selection function for Definite clauses:

- K con clausole *definite*:
  - Clausole di Horn: al più un letterale non negato.
  - Strategia linear input: a ogni passo di risoluzione, una *variante* di una clausola è sempre scelta nella K di partenza (programma) mentre l'altra è sempre il risolvente del passo precedente (goal, la negazione di F al primo passo).
  - Variante: clausola con variabili rinominate.

Note:-

NON LSD.

Domanda 2.2

Ma perché ci si limita alle clausole di Horn?

Risposta: si rimuove la parte "intuitiva" che non può essere implementata nel PROLOG. Inoltre le clausole di Horn garantiscono la completezza.

Derivazione SLD per un goal G_0 da un insieme di clausole K è:

- Una sequenza di clausole goal $G_0, G_1, \ldots, G_n$.
- Una sequenza di varianti di clausole di $KC_1, C_2, \ldots, C_n$.
- Una sequenza di MGU $\alpha_1, \alpha_2, \ldots, \alpha_n$, tali che G_{i+1} è derivato da G_i e da C_{i+1} attraverso la sostituzione α_{i+1},

Tre possibili tipi di derivazioni:

- Successo se G_n è vuoto (`true`).
- Fallimento finito, se non è possibile derivare da G_n alcun risolvente e G_n non è vuoto (`false`).
- Fallimento infinito, se è sempre possibile derivare nuovi risolventi (loop infinito).

Due forme di non determinismo:

- Regola di calcolo per selezionare a ogni passo l'atomo B_i del goal da unificare con una clausola.
- Scelta di quale clausola utilizzare a ogni passo di risoluzione.

Definizione 2.2.3: Regola di calcolo

Funzione che ha come dominio l'insieme dei goal e per ogni goal seleziona un suo atomo.

Note:-

La regola di calcolo non influenza correttezza e completezza del metodo di prova.

Domanda 2.3

Come si costruisce l'albero SLD?

Data una regola di calcolo, è possibile rappresentare tutte le derivazioni con un albero SLD:

- Nodo: goal.
- Radice: goal iniziale G_0.
- Ogni nodo $\leftarrow A_1, \ldots, A_m, \ldots, A_k$, dove A_m è l'atomo selezionato dalla regola di calcolo, ha un figlio per ogni clausola A $\leftarrow B_1, \ldots, B_k$ tale che A e A_m sono unificabili con MGU α. Il nodo figlio è etichettato con il goal $\leftarrow [A_1, \ldots, A_{m-1}, B_1, \ldots, B_k, A_{m+1}, \ldots, A_k]\alpha$. Il ramo dal padre al figlio è etichettato con α e con la clausola selezionata.

Scelte per rendere la strategia deterministica:

- Regola di computazione: *leftmost* (viene sempre scelto il sottogoal più a sinistra).
- Clausole considerate nell'*ordine in cui sono scritte nel programma*.
- Strategia di ricerca: *in profondità con backtracking*.
  - Non è completa perché se una computazione che porta al successo si trova a destra di un ramo infinito l'interprete non la trova, perché entra, senza mai uscirne, nel ramo infinito.

Note:-

Cercare di mettere a destra le computazioni che possano produrre eventuali casini.