2024, VOL. 21, N. 44, pp. 138 - 150 ISSN 2279-9001

CONTRIBUTO TEORICO

Coding e logica a scuola: Maturare soft-skills con PROLOG.

Coding and Logic in Schools: Developing Soft Skills with PROLOG.

Maurizio Pattoia, Università degli Studi di Perugia.

ABSTRACT ITALIANO

Il presente lavoro introduce un approccio differente e spesso dimenticato alla programmazione informatica: la programmazione logica. La programmazione informatica è un importante strumento didattico, oltre che strumento tecnico di produzione software. È costantemente in evoluzione e, alla luce dei nuovi strumenti di intelligenza artificiale, in profondo mutamento per ciò che concerne l'intervento umano e la capacità di comprendere i nuovi modelli computazionali sempre più vicini ai modelli di pensiero umano.
Esplorare non solamente i paradigmi classici della programmazione imperativa, ma anche quelli della programmazione dichiarativa e, in particolare della programmazione logica, è diventato un cammino importante per perseguire uno sviluppo non parziale dei cosiddetti ambiti di pensiero computazionale.

ENGLISH ABSTRACT

This paper introduces a different and often forgotten approach to computer programming: Logic Programming.
Computer programming is today an important teaching tool as well as a technical tool for software development. It is constantly evolving and, in the light of new artificial intelligence tools, undergoing deep changes as regards human intervention and the ability to fully understand the new computational models, which are ever closer to human thought patterns.
Exploring, therefore, not only the classical paradigms of Imperative Programming, but also those of Declarative Programming, and particularly Logic Programming, has become an important way to pursue a full, non-selective, development of computational thinking aspects.
.

Introduzione

Nell'ultimo ventennio ha avuto forte impulso la diffusione di modelli di programmazione informatica innovativi e differenti da quelli tradizionalmente e largamente adottati per lo sviluppo dei software.

Uno dei paradigmi per il quale c'è stato via via un crescente interesse è sicuramente quello definito, e poi distinto nelle sue differenti declinazioni, come Programmazione Logica. Questo lavoro intende fornire alcuni riferimenti di base per l'introduzione, nella scuola in generale ed in quella secondaria in particolare, della Programmazione Logica, da qui PL, intesa come strumento per la maturazione di soft-skill inerenti al pensiero computazionale.

Autore per la Corrispondenza: Maurizio Pattoia - Università degli studi di Perugia, Dipartimento di Filosofia, Scienze Sociali, Umane e della Formazione.
E-mail: maurizio.pattoia@unipg.it

Il pensiero computazionale e la PL

La via più seguita nel definire un percorso di sviluppo del pensiero computazionale passa inevitabilmente per una definizione/disamina del cosiddetto "ragionamento logico" o, meglio, "ragionamento logico razionale"; ovvero quel processo cognitivo che permette di giungere a una o più conclusioni, partendo da specifiche premesse poste in forma di proposizioni logiche "(1)" e seguendo un percorso logico-razionale.

Il ragionamento logico "(2)" si basa sulle deduzioni logiche, ovvero sulla considerazione che, se le premesse sono vere, allora anche la relativa conclusione è vera.

Una delle categorizzazioni dei linguaggi di programmazione informatica identifica oggi differenti approcci che prendono origine da distinti modelli di astrazione nella programmazione.

Si riconoscono differenti paradigmi di programmazione (Fig. 1), i quali certamente influenzano l'attività di programmazione e, a monte, l'attività di analisi in astrazione o, se si preferisce, l'evidente impronta per impiego del cosiddetto pensiero computazionale. In sostanza la scelta tra un modello di programmazione o un altro ci spinge a adottare una forma di pensiero computazionale piuttosto che un'altra.

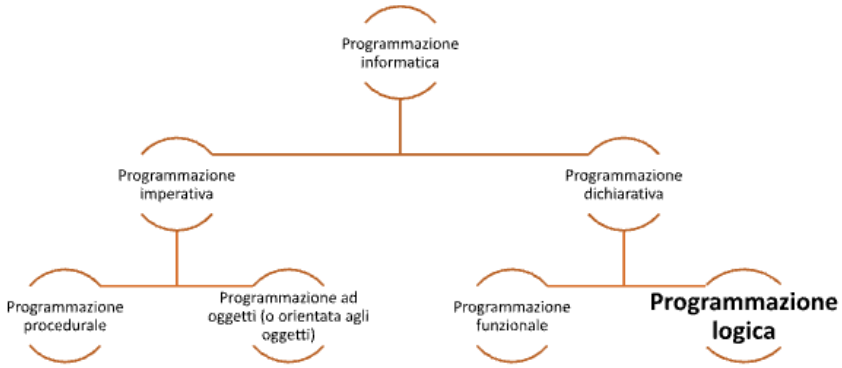

FIG. 1 – PARADIGMI DI PROGRAMMAZIONE INFORMATICA.

Il paradigma cosiddetto classico, più largamente diffuso, è quello della programmazione imperativa che prevede la puntuale definizione dei passi che devono essere svolti, e in quale sequenza o ordine. Tale modello si riferisce alla forma di astrazione o di pensiero computazionale nota come pensiero algoritmico, e poi vede almeno due differenti approcci specifici che sono la programmazione procedurale e la programmazione ad oggetti.

La programmazione imperativa è fortemente utilizzata in molti linguaggi di programmazione popolari come Basic, C, C++, Java, Python e molti altri. Questo modello è considerato molto utile per la risoluzione di problemi che richiedono la modifica dello stato del sistema e la manipolazione dei dati.

Il paradigma al quale si riferisce il presente lavoro è invece quello denominato programmazione dichiarativa, il quale prevede la "semplice" descrizione del risultato che s'intende ottenere e non dei passaggi necessari per ottenerlo. Ad esempio, i linguaggi di

formato come l'HTML o il PDF descrivono il contenuto di un documento ma non descrivono cosa fare per visualizzarlo o stamparlo. In sintesi, si può affermare che la programmazione dichiarativa è orientata a descrivere il "cosa" piuttosto che il "come".

In tale paradigma si riconoscono, poi, due differenti sottotipi: la programmazione funzionale e la programmazione logica anche se queste distinzioni non sono tassonomicamente definibili, ma piuttosto indicative.

Al di là dei pro e dei contro, dei vantaggi e degli svantaggi, ascrivibili al paradigma di programmazione imperativa o a quello di programmazione dichiarativa, occorre osservare come, dal punto di vista pedagogico e didattico, siano entrambi utilizzabili per permettere di maturare differenti forme di pensiero computazionale.

Ormai è diffusamente promosso, anche a livello scolastico, il modello di pensiero computazionale che sottende al paradigma imperativo, ovvero quello di ragionamento per passi verso l'obiettivo. Non è altrettanto perseguita la promozione del pensiero computazionale che riguarda il definire uno stato finale o un risultato, svincolandolo dal processo (es. dall'algoritmo) di soluzione.

Probabilmente il paradigma di programmazione imperativa è quello più diffuso e promosso in quanto storicamente più vecchio e più immediato nella comprensione se lo si pensa nella sua forma algoritmica, ovvero come insieme, sequenza, di comandi semplici dati ad un esecutore per svolgere un compito.

Nella realtà molte attività dell'uomo si svolgono in questo modo o si sono conformate nel tempo a tale modello. Esempio ne sono le catene di montaggio industriali, i processi in produzione in generale o i processi didattici nelle forme di stampo comportamentista-positivista. Quante cosiddette Unità di Apprendimento o altri progetti didattici si basano sul "fai questo, poi fai quello, poi quell'altro, fine!"?

Le nuove sfide che ci pone la moderna complessità, rilevabile a tutti i livelli, ci impongono, però, di maturare ed allenare anche altre forme di pensiero computazionale che non siano incentrate sui processi ma sui prodotti (o risultati); insomma si possono definire chiaramente le caratteristiche di un oggetto senza per forza doverne definire i processi di realizzazione. Qual è il vantaggio di tale approccio? Uno sicuramente è quello di imparare a riconoscere e descrivere, in sostanza definire, i risultati; ovvero cosa ci serve esattamente, senza preoccuparci di come realizzarli.

Ad esempio io posso dichiarare che voglio un asse di legno di faggio rettangolare delle dimensioni di 30x20cm e di 4cm di spessore, senza dover comandare ad un esecutore di individuare un bosco di faggi, prendere l'auto, andare in quel bosco, individuare un albero adatto, ecc..

Ovviamente la forma dichiarativa è poco operativa perché lascia fuori tutti gli importanti elementi esecutivi che dovranno essere oggetto di ulteriore riflessione; ma allo stesso tempo, svincolando il cosa dal come, lascia poi la possibilità di riflettere creativamente ed in molteplici modi sul come.

Andando in dettaglio, in un contesto pedagogico-didattico, la programmazione dichiarativa può essere proficuamente esplorata in forma di linguaggi d'interrogazione, ovvero i cosiddetti query languages, oppure in forma di linguaggi di PL.

Oggi numerosi, relativamente standardizzati e diffusi, sono i linguaggi d'interrogazione i quali si rifanno quasi tutti al modello chiamato Structured Query Language (SQL).

Pur rilevando l'importanza sia funzionale che didattica di tale categoria di linguaggi di programmazione dichiarativa, in questa occasione si è scelto di rivolgere l'attenzione al meno noto e diffuso modello di programmazione dichiarativa noto come PL, in particolare osservandone funzionalità e prospettive attraverso il linguaggio di programmazione PROLOG.

Dalla logica alla PL

Come già accennato per PL s'intende quella tipologia di programmazione dichiarativa che permette di rappresentare ed elaborare le informazioni attraverso la loro struttura logica o logico-matematica.

La programmazione dichiarativa ha avuto origine negli anni '50 e '60 con lo sviluppo della logica matematica e della teoria della computabilità. In questo periodo, i ricercatori hanno cominciato a studiare la logica come una forma di rappresentazione delle conoscenze e delle informazioni, che poteva essere utilizzata per la risoluzione di problemi complessi.

Nel corso degli anni, la programmazione dichiarativa ha continuato a evolversi e ad essere utilizzata in una vasta gamma di applicazioni, come la risoluzione dei problemi di intelligenza artificiale, la manipolazione delle basi di conoscenza e la modellizzazione dei sistemi complessi.

Oggi, la programmazione dichiarativa è un importante sottoinsieme dell'intelligenza artificiale e della teoria della computazione, e viene utilizzata in una vasta gamma di applicazioni, dalla scienza dei dati alla modellizzazione delle relazioni tra i dati.

Probabilmente il più importante teorico della PL è Robert A. Kowalsky il quale, a partire dalla fine degli anni '60, iniziò ad occuparsi di logica computazionale, ovvero quella logica piegata alla dimostrazione automatizzata di teoremi; e successivamente pose le basi teoriche della moderna PL.

Sintesi chiave del lavoro di Kowalsky fu il passaggio dall'equazione di Wirth sulla programmazione:

$$Algoritmi + Strutture\ dei\ dati = Programmi$$

che evidenzia come nella classica forma i programmi siano l'intima fusione del cosa fare e del come farlo; all'equazione nota proprio come equazione di Kowalsky:

$$Algoritmo = Logica + Controllo$$

svincolando sostanzialmente il senso del Cosa dal Come.

L'equazione di Kowalsky ci indica come ogni problema possa essere affrontato distinguendo la soluzione in parte logica, ovvero definendo direttamente con i dati la struttura del problema e gli elementi necessari per risolverlo; e parte di controllo, la quale identifica le strutture in cui i dati vengono memorizzati e gestiti.

> Nei sistemi basati sulla logica, l'operazione fondamentale del pensiero è la deduzione logica, ma dal punto di vista dei sistemi basati su regole, l'operazione fondamentale del pensiero è la ricerca (Kowalsky, 1979).

Approfondendo il discorso, Kowalsky (Kowalsky, 2011) ci fa notare come il pensiero logico sia utilizzato prevalentemente quale "deduzione in avanti" nella forma top-down o procedurale; e non come "ragionamento all'indietro" o bottom-up. Ma, ricorda sempre Kowalsky, è nel ragionamento all'indietro che, partendo direttamente dall'obiettivo che s'intende raggiungere e ragionando a ritroso, si può trovare il maggior beneficio e avere la maggior efficienza dal ragionamento logico.

Partendo da tali assunti e tenendo conto che lo scopo fondamentale e sempre quello di stimolare lo sviluppo cognitivo nei discenti, si può pensare di sviluppare un percorso di approccio al pensiero computazionale già dai gradi di scuola più bassi che, partendo da semplici elementi di base di logica, porti ad un normale impiego della stessa nei ragionamenti e crei dimestichezza nell'utilizzo di concetti propri di questa disciplina da subito, per poter poi lavorare successivamente in approfondimento sulla cosiddetta PL.

Questo iniziale percorso oggi è indicato e definito come educazione logica, a seguire anche EL.

L'EL si lega intimamente a quanto proposto come significato di sviluppo del cosiddetto pensiero computazionale e amplia il concetto rispetto a quanto rilevabile, ad esempio, nelle iniziative ministeriali del progetto "Programma il Futuro".

In tale quadro la PL non è più solamente una modalità di approccio alla programmazione informatica, ma diventa un framework curriculare per portare chiaramente i discenti a sviluppare dapprima il senso del pensiero logico in generale e del linguaggio formale di tale pensiero; e poi a collegare tali elementi alla programmazione informatica per renderli significativi e operanti.

Tutto questo percorso richiede un'iniziale alfabetizzazione inerente al linguaggio matematico-logico e la conseguente riflessione sulle differenze formali e semantiche esistenti tra tale linguaggio e gli altri linguaggi comunemente usati.

Un buon punto di partenza per l'EL potrebbe essere la condivisione della definizione degli elementi costitutivi del ragionamento logico, ovvero la proposizione (logica) e l'enunciato.

Per iniziare, poi, un vero percorso di EL, ed in particolare l'apprendimento delle basi del linguaggio della logica cosiddetta del Primo Ordine "(3)", ci si potrebbe avvalere di strumenti software appositi come il Tarski's world.

La PL e il linguaggio PROLOG

Nella programmazione dichiarativa, e quindi anche in PL, i programmatori codificano esplicitamente le informazioni di dominio rilevanti e gli obiettivi/risultati del programma, ma non specificano i dettagli dell'elaborazione interna, lasciando ai sistemi esecutori, che sono chiamati ad eseguire quei programmi, la decisione in merito a tali dettagli.

Per comprendere pienamente questa distinzione, si consideri il compito di programmare un veicolo per spostarsi da un punto ad un altro della città. Un tipico programma imperativo indicherebbe progressivamente al veicolo in questione di avanzare per un tot di metri, poi di girare, poi di avanzare ancora e così via fino a quando il veicolo non sarà giunto nel punto considerato come punto di arrivo.

Differentemente, un programma dichiarativo sarebbe costituito da una dettagliata mappa della città, compresi eventuali vincoli, come l'impossibilità di attraversare un muro o la necessità di rimanere in una corsia della strada, e da un'indicazione dei punti di partenza e di arrivo sulla mappa stessa, lasciando al veicolo/esecutore la gestione del processo esecutivo, ovvero come e dove procedere.

Tali principi di base sono, nel corso del tempo, confluiti in differenti linguaggi di programmazione capaci di rendere in forma applicativa gli elementi teorici della PL.

Uno di questi linguaggi di programmazione, sicuramente uno dei più diffusi ed evoluti, è il PROLOG.

Coding a Scuola e PROLOG

L'introduzione del coding a scuola è diventata una questione sempre più rilevante negli ultimi anni poiché l'economia mondiale diventa sempre più digitale e la tecnologia gioca un ruolo sempre più significativo nella vita quotidiana. In questo contesto l'apprendimento del coding e lo sviluppo del pensiero computazionale sono diventati una parte importante dell'educazione scolastica e l'utilizzo di linguaggi di programmazione come PROLOG può offrire molti vantaggi.

Mentre un approccio all'EL come disciplina trasversale sarebbe auspicabile già a partire dalla scuola primaria e, addirittura, con interventi già nella scuola dell'infanzia magari legati ad attività di coding preferibilmente in forma unplugged, la programmazione PROLOG potrebbe essere introdotta a partire dal 3° o 4° anno di scuola primaria, per trovare poi sviluppo, sempre nel framework di riferimento delle cosiddette competenze chiave, nella scuola secondaria di primo grado. Infine, è auspicabile un forte approfondimento di tali competenze, che oggi potrebbero sfociare anche nella didattica dell'Intelligenza Artificiale, nella scuola secondaria di secondo grado.

Innanzitutto, PROLOG è un linguaggio di programmazione basato sulla PL, che offre un modo diverso di pensare e di risolvere problemi rispetto ai linguaggi di programmazione imperativi più comuni. Questo può aiutare gli studenti a sviluppare una maggiore comprensione del pensiero computazionale e ad acquisire competenze trasferibili in altre aree.

Inoltre, PROLOG è un linguaggio di alto livello che rende facile per gli studenti comprendere e scrivere codice, anche se non hanno esperienza di programmazione. Ciò significa che gli studenti possono concentrarsi sul problema e sulla logica, piuttosto che sulle complesse strutture di codice, e questo può aiutare a sviluppare le loro capacità di pensiero critico e di problem-solving.

L'utilizzo di PROLOG può aiutare gli studenti a comprendere meglio la logica e il ragionamento, poiché il linguaggio è basato sulla rappresentazione della conoscenza come un insieme di fatti e regole. Ciò aiuta gli studenti a sviluppare una maggiore

comprensione delle relazioni tra i concetti e a imparare a pensare in modo più sistematico e rigoroso.

Aldilà dell'aspetto meramente tecnico-informatico, l'utilizzo di Prolog può offrire un'opportunità unica per lo sviluppo di soft-skills quali:

- *Comunicazione:* La natura logica di Prolog richiede ai programmatori di pensare in modo chiaro ai problemi da risolvere. Scrivendo codice in Prolog si impara a descrivere idee e soluzioni in modo preciso e conciso, soft-skill importante in molte altre aree professionali.
- *Pensiero critico*: La programmazione in Prolog richiede di formulare soluzioni logiche complesse a problemi specifici. Questo processo richiede una valutazione critica dei diversi approcci e la selezione della soluzione più adatta. Questa pratica aiuta a sviluppare la capacità di pensare criticamente.
- *Problem solving*: La natura logica di Prolog e la sua capacità di gestire conoscenze complesse fanno sì che i programmatori Prolog siano esperti nella risoluzione dei problemi. Imparare a formulare soluzioni logiche a problemi specifici e a testare la loro validità attraverso la programmazione in Prolog può aiutare a sviluppare la capacità di risolvere problemi in modo efficiente.
- *Lavoro di gruppo*: La programmazione in Prolog spesso richiede la collaborazione con altri esperti di intelligenza artificiale e conoscenza. Questa collaborazione può aiutare a sviluppare le soft-skill di team-work, a rispettare le opinioni degli altri e a lavorare verso un obiettivo comune.

Infine, PROLOG è ancora utilizzato in molte applicazioni, soprattutto nell'intelligenza artificiale e nella ricerca, il che significa che gli studenti che imparano a utilizzare questo linguaggio saranno preparati a raccogliere molte future opportunità.

Iniziare a lavorare con il PROLOG a Scuola

Per iniziare a programmare in PROLOG a scuola, è importante seguire i seguenti passaggi:

1. *Comprendere i concetti di base:* prima di iniziare a scrivere codice, è importante comprendere i concetti di base della programmazione logica e di come funziona PROLOG. Questo include comprendere le regole di base della programmazione, come le variabili, le espressioni, i predicati e le regole.
2. *Apprendere l'interfaccia di programmazione*: una volta compresi i concetti di base, è importante imparare l'interfaccia di programmazione utilizzata per scrivere codice in PROLOG. Questo include imparare a utilizzare un ambiente di sviluppo integrato (IDE).
3. *Scrivere codice semplice:* una volta che si ha familiarità con l'interfaccia di programmazione, è importante iniziare a scrivere codice semplice. Questo potrebbe includere l'implementazione di semplici algoritmi o l'utilizzo di regole per rappresentare conoscenza.
4. *Imparare attraverso esempi:* un modo efficace per imparare a programmare in PROLOG è esplorare e analizzare esempi di codice esistenti.

5. *Sviluppare progetti più complessi*: una volta che si è acquisita familiarità, è possibile iniziare a sviluppare progetti più complessi, che possono aiutare a consolidare le competenze acquisite e a sviluppare una maggiore comprensione del pensiero computazionale e della programmazione logica.

I concetti di base del PROLOG sono fondamentali per comprendere e utilizzare questo linguaggio di programmazione. Ecco alcuni dei concetti chiave da insegnare agli studenti:

- *Predicati:* i predicati sono affermazioni che descrivono relazioni o fatti. Ad esempio, "il gatto è un animale domestico" è un predicato che descrive una relazione tra il gatto e l'animale domestico.
- *Regole:* le regole sono utilizzate per dedurre nuove affermazioni a partire da quelle esistenti. Ad esempio, se sappiamo che "il gatto è un animale domestico" e che "gli animali domestici hanno bisogno di cibo e acqua", possiamo dedurre che "il gatto ha bisogno di cibo e acqua".
- *Variabili*: le variabili sono utilizzate per rappresentare elementi sconosciuti o mutabili nei predicati e nelle regole. Ad esempio, una variabile potrebbe rappresentare il nome di un animale domestico e potrebbe essere utilizzata in un predicato come "X è un animale domestico".
- *Espressioni*: le espressioni sono utilizzate per rappresentare valori numerici o altri tipi di dati. Ad esempio, un'espressione potrebbe rappresentare l'età di un animale domestico.
- *Query:* le query sono domande che vengono poste al sistema PROLOG per ottenere informazioni. Ad esempio, una query potrebbe essere "è il gatto un animale domestico?" e il sistema PROLOG risponderà "sì" ("True").
- Ci sono diversi ambienti o interfacce di sviluppo integrato (IDE) gratuite disponibili per programmare in PROLOG. Tra le più comuni:
- *SWI-Prolog:* è uno dei sistemi PROLOG più diffusi e completi. Offre un'interfaccia grafica intuitiva e una vasta libreria di funzioni che possono essere utilizzate per creare applicazioni.
- *Gnu Prolog:* è un'altra popolare IDE gratuita per PROLOG. Offre un'interfaccia semplice e intuitiva e una vasta libreria di funzioni e moduli.
- *Visual Prolog:* è una IDE visuale per PROLOG che offre un'interfaccia grafica intuitiva e una vasta libreria di funzioni.
- *Yap Prolog*: è una potente IDE per PROLOG che offre una vasta libreria di funzioni e moduli, oltre a un'interfaccia grafica intuitiva.

Queste sono solo alcune delle molte IDE gratuite disponibili per programmare in PROLOG.

È importante scegliere l'IDE che meglio si adatta alle esigenze e al livello di esperienza dell'utente, oltre che all'infrastruttura hardware e software di cui si dispone. In generale, le IDE gratuite per PROLOG offrono un ambiente di sviluppo completo e accessibile che è ideale per imparare a programmare in questo linguaggio.

Come per tutti i linguaggi di programmazione, la maniera migliore per approcciare è attraverso la scrittura di poche e semplici righe di codice.

Però, prima di partire occorrerebbe avere a disposizione un manuale di riferimento e fortunatamente per il PROLOG si trovano molti interessanti manuali di programmazione per principianti gratuiti in rete. Ad esempio, i più noti e diffusi in lingua inglese sono:

- *Learn Prolog Now!:* È un manuale online gratuito che fornisce una introduzione completa al PROLOG. Include esempi e esercizi che possono aiutare a capire i concetti di base.
- *Prolog Programming in Depth:* Questo è un altro manuale gratuito che fornisce una introduzione dettagliata al PROLOG. Incluse lezioni sulla logica, i fatti, le regole, le query e molto altro ancora.
- *A Guide to Artificial Intelligence with Prolog:* È un libro elettronico gratuito che introduce ai concetti di intelligenza artificiale e di programmazione in PROLOG. Include esempi e esercizi per aiutare a comprendere i concetti.
- *The Art of Prolog*: Altro noto libro elettronico gratuito che fornisce una introduzione completa al PROLOG e alla PL. Incluse lezioni sui fatti, le regole, le query, la risoluzione dei problemi e molto altro ancora.
- Ovviamente esistono manuali e riferimenti gratuiti anche in lingua italiana, tra i quali:
- *Come Programmare in Prolog:* Manuale online dal sito TeleTec.it
- *Prolog Language:* È un e.book in italiano gratuito liberamente scaricabile

A seguire alcuni esempi semplici di programmazione in PROLOG (Figg. 2-6) che possono aiutare a capire i concetti di base (la parte di testo che in una riga segue il simbolo % è da considerarsi commento e non ha effetti nell'esecuzione del programma):

```
% Definizione dei fatti (costrutto "genitore di")
genitore(Pamela,Roberto).
genitore(Tommaso,Roberto).
genitore(Tommaso,Elisabetta).
genitore(Roberto,Anna).
genitore(Roberto,Patrizia).
genitore(Patrizia,Giovanni).

% Definizione delle regole
fratello(X,Y) :- genitore(Z,X), genitore(Z,Y), X = Y.
```

FIG. 2 – DEFINIZIONE DI FATTI E REGOLE.

```
% Definizione dei fatti
prodotto(penna,blue,1).
prodotto(matita,red,2).
prodotto(quaderno,green,3).
% Definizione della regola
cerca_prodotto(C,N) :- prodotto(N,C,_).
```

FIG. 3 – RICERCA DI UNA SOLUZIONE.

```
% Definizione dei fatti
costa(pera,100).
costa(banana,200).
costa(mela,150).

% Definizione delle regole
totale_costo(Lista,Totale) :-
findall(Costo,(member(Frutto,Lista),costa(Frutto,Costo)),L
istaCosto), sumlist(ListaCosto,Totale).
```

FIG. 4 – RISOLUZIONE DI PROBLEMI.

Questi sono solo alcuni esempi semplici di programmazione in PROLOG. Man mano che si acquisisce esperienza è possibile creare programmi più complessi e sofisticati.

```
% Definizione dei fatti
fratello(mario, giovanni).
fratello(giovanni, mario).

% Definizione delle regole
fratello(X, Y) :- fratello(Y, X).

% Interrogazione
?- fratello(mario, giovanni).
True.

?- fratello(giovanni, mario).
True.

?- fratello(mario, anna).
False.
```

FIG. 5 – IL PROBLEMA DEI FRATELLI.

In questo esempio, sono definiti due fatti che affermano che Mario e Giovanni sono fratelli.

A seguire si è poi definita una regola che afferma che se Y è il fratello di X, allora X è il fratello di Y. Infine, due interrogazioni per verificare se Mario e Giovanni sono fratelli e se Mario è fratello di Anna.

```
% Definizione delle caratteristiche degli animali
caratteristica(gatto, pelo).
caratteristica(gatto, miagola).
caratteristica(cane, pelo).
caratteristica(cane, abbaia).
caratteristica(uccello, pennuti).
caratteristica(uccello, canta).

% Regola per identificare un animale in base alle sue
caratteristiche
identifica_animale(Animale) :-
   caratteristica(Animale, Caratteristica1),
   caratteristica(Animale, Caratteristica2),
   write("Questo animale ha i seguenti tratti: "),
   write(Caratteristica1), write(", "), write(Caratteristica2),
   nl.

% Esempi di utilizzo del sistema esperto
?- identifica_animale(gatto).
Questo animale ha i seguenti tratti: pelo, miagola
true.

?- identifica_animale(cane).
Questo animale ha i seguenti tratti: pelo, abbaia
true.

?- identifica_animale(uccello).
Questo animale ha i seguenti tratti: pennuti, canta
true.
```

FIG. 6 – UN ESEMPIO DI CODICE PER UN SISTEMA ESPERTO CHE IDENTIFICA GLI ANIMALI IN BASE ALLE LORO CARATTERISTICHE.

In questo esempio, il sistema esperto ha una base di conoscenza che descrive le caratteristiche di diversi animali. La regola *identifica_animale* utilizza queste conoscenze per identificare l'animale corrispondente in base alle sue caratteristiche. Nel codice sono presenti anche alcuni esempi di utilizzo del sistema esperto, che mostrano come il sistema sia in grado di identificare un animale in base alle sue caratteristiche.

Conclusioni

La Programmazione Logica (PL) e il PROLOG possono aiutare a maturare importanti soft-skill come il problem-solving, il ragionamento logico e quello creativo e rappresentano un compendio importante alle tradizionali attività di coding basate sulla programmazione imperativa per un completo sviluppo di ciò che è definito oggi pensiero computazionale.

La logica formale permette agli studenti di pensare in modo più strutturato e sistematico, sviluppando la loro capacità di analisi e risoluzione dei problemi. Inoltre, la PL richiede una forte attenzione ai dettagli e la capacità di organizzare e rappresentare

informazioni in modo logico, che a loro volta sono importanti soft-skill utilizzate in molti ambiti lavorativi.

Il PROLOG risulta essere un linguaggio di programmazione molto flessibile che incoraggia la creatività e la capacità di pensare fuori dagli schemi, in quanto fornisce molte opportunità per la modellizzazione di problemi complessi in modo innovativo.

In definitiva, l'introduzione alla PL e al PROLOG in un contesto più generale di EL possono aiutare gli studenti a sviluppare competenze chiave trasversali essenziali ad affrontare le sfide complesse che il futuro porrà loro.

Note

(1) La proposizione logica è uno specifico enunciato o dichiarazione che può evidenziarsi oggettivamente come "vero" o "falso". Ad esempio, è una proposizione logica "Il mio nome è Maurizio" mentre non lo è "Maurizio è bello", in quanto la prima ha una connotazione oggettiva, mentre la seconda ha una connotazione soggettiva.

(2) Parlare di "ragionamento logico" in effetti può apparire come una ridondanza di concetti in quanto, in sé, il ragionamento è già normalmente inteso come un processo cognitivo che, attraverso passaggi logici, porta da una serie di premesse o congetture ad una situazione finale o conclusiva.

(3) Il linguaggio della Logica del Primo Ordine (in inglese First Order Logic – FOL) è un linguaggio formale per la rappresentazione degli enunciati. Il linguaggio della Logica del Primo Ordine ci permette di rappresentare la realtà mettendo in relazione o definendo elementi attraverso predicati; ad es. "Mario è forte", "Aldo chiama Lucia", dove gli elementi sono "Mario", "Aldo" e "Lucia" e i predicati "è forte" e "chiama".

Bibliografia

Battani, G., Meloni, H. (1973). *Interpreteur du langage de programmation. Rapport de DEA, Groupe d'Intelligence Artificielle.* Université de Marseille.

Frédéric Benhamou, F., Bouvier, P., Colmerauer, A., Henri Garreta, H., et Al. (1996). *Le manuel de Prolog IV.* PrologIA.

Calvani A., Peru A., Pellegrini M., Di Martino V. (2023). *Potenziare logica e problem solving*. Carocci Editore.

Clocksin, W. F., Mellish, C. S. (1987). *Programming in Prolog: Using the ISO Standard.* Springer-Verlag.

Colmerauer, A., de Chastellier, G. (1968). *W-Grammar.* Conference ACM.

Colmerauer, A., Kanoui, H., Roussel, P., Pasero R. (1973). Un système de communication homme-machine en Français. *Rapport preliminaire, Groupe de Res. en Intell. Artif.*.

Colmerauer, A. (1990). An Introduction to Prolog III. *Communications of the ACM,* 33(7).

Colmerauer, A. (1970). Les systèmes-q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur. *TAL. Traitement automatique des langues,* 33(1-2), 105-148.

Colmerauer, A., Roussel, P. (1996). The birth of Prolog, pdf, (La naissance de Prolog), (draft of a paper)- *History of Programming Languages,* ed.Thomas J. Bergin e Richard G. Gibson, ACM Press/ Addison-Wesley, 1996.

Colmerauer, A. (2011). From natural language processing to Prolog. *Academy of Mathematics and Systems Science*.

Console, L., Lamma, E., Mello, P. (1997). *Programmazione Logica e PROLOG*. UTET.

Frangioni, A., Lippi, M. (2021). *Introduzione alla programmazione logica: Applicazioni didattiche e industriali.* Springer.

Giannesini, F, Kanoui, H., Pasero, R., van Caneghem, M. (1986). *Prolog.* Addison-Wesley.

Kong S., Abelson H. (Editors) (2019). *Computational Thinking Education*. Springer..

Kowalski, R. A. (1974). Predicate logica as programming language, in Proceedings of IFIP (International Federation for Information Processing) Congress 74, North-Holland Ed.

Kowalski, R. A. (1979). *Logic for Problem Solving*. Elsevier Science Ltd.

Kowalski, R. A. (2011). *Logic Programming.* Imperial College London – Dpt. of Computing..

Pereira, F.C.N., Shieber, S.M. (1987). *Prolog and Natural-Language Analysis.* Microtome Publishing.

Robinson, J. A. (1965). A Machine Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1), 23-41.

Roussel P. (1975). *PROLOG Manuel de référence et d'utilisation.* Groupe de recherche en Intelligence Artificielle.

Sterling, L., Arienti, M.,Shapiro, E. (1994). *L'arte del Prolog: tecniche di programmazione avanzata*. Hoepli.