STEFANO FERILLI

Monografia su

PROGRAMMAZIONE LOGICA E PROLOG

Università degli Studi di Bari
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Corso di Ingegneria della Conoscenza e Sistemi Esperti

PROGRAMMAZIONE LOGICA

La programmazione logica nacque all'inizio degli anni '70, grazie soprattutto alle ricerche di due studiosi.

- *Kowalski* ne elaborò i fondamenti teorici. In particolare, a lui si deve la scoperta della doppia interpretazione, procedurale e dichiarativa, delle clausole di Horn.
Proprio l'interpretazione procedurale è quella che ha consentito la realizzazione pratica su calcolatore dei linguaggi di programmazione logica come il Prolog.

Esempio di interpretazione della clausola di Horn P :- Q, R.
Interpretazioni dichiarative:
- P È vero se Q e R sono veri.
- Da Q e R segue P.

Interpretazioni procedurali:
- Per risolvere il problema P, risolvi nell'ordine i sottoproblemi Q e R.
- Per soddisfare P, soddisfa nell'ordine Q e R.

- *Colmerauer* fu il primo a progettare e implementare un interprete per un linguaggio logico: il Prolog.

Col passare del tempo l'interesse per questo nuovo tipo di programmazione si è allargato, uscendo dall'ambito accademico ed approdando nel mondo industriale. Nel 1981 i giapponesi, in un rapporto, hanno "eletto" il Prolog come linguaggio su cui fondare la progettazione della V generazione di computer, caratterizzata da un elevato parallelismo.

Attualmente il Prolog è il linguaggio di programmazione logica per eccellenza, e trova impiego nelle più svariate aree applicative dell'Intelligenza Artificiale e dell'Ingegneria della Conoscenza.

Il *Sillogismo* è una forma di ragionamento deduttivo, dovuta ad Aristotele, tramite cui partendo da un giudizio generale si giunge ad un giudizio particolare. Esso è composto da una *premessa maggiore* e da una *premessa minore*, che collegate tramite un *termine medio* consentono di giungere ad una *conclusione*.

Esempio.

Tutti gli uomini sono mortali	premessa maggiore
Socrate è un uomo	premessa minore
---------- *uomo*	termine medio
Socrate è mortale	conclusione

In programmazione logica un problema viene *descritto* con un insieme di formule della logica, dunque in forma *dichiarativa*. L'interpretazione procedurale delle clausole consente di usare alcune tecniche di dimostrazione di teoremi come meccanismi di esecuzione dei programmi.

logica = rappresentazione del problema
deduzione = risoluzione del problema

La programmazione classica, *procedurale*, è più adatta a problemi quotidiani, ben definiti. Essa adotta un paradigma *imperativo*: richiede, cioè, che siano specificate delle rigorose sequenze di passi (*algoritmi*) che, a partire dai dati a disposizione, portino ad ottenere i risultati desiderati.

Corrispondentemente alla famosa equazione di Wirth per la programmazione classica:

Algoritmi + Strutture dati = Programmi

Kowalski propose la seguente equazione:

Algoritmo = Logica + Controllo

Nei programmi classici i dati (cosa) e il controllo (come) si intersecano tra loro formando un agglomerato inscindibile.
Nella programmazione logica, al contrario, essi sono ben distinti. Nell'idea di Kowalski il programmatore, qui, dovrebbe preoccuparsi di specificare solo la componente logica, mentre il controllo dovrebbe essere appannaggio esclusivo del sistema. Ciò significa che egli deve solo definire il problema, formulandone le specifiche, senza dire alla macchina come risolverlo.

Nei linguaggi procedurali i programmi si scrivono specificando la sequenza di operazioni che il calcolatore deve eseguire per risolvere il

problema. Vengono generalmente lasciate implicite le assunzioni su cui l'algoritmo si basa. (*stile prescrittivo*, *how-type*).

Nella programmazione logica i programmi si scrivono descrivendo la conoscenza relativa al problema, cioè specificando gli oggetti che vi intervengono e le relazioni fra di essi. (*stile descrittivo*, *what-type*).

Poiché il controllo è totalmente avulso dalla descrizione del problema, e il programmatore non deve farsene carico perché è demandato totalmente al sistema, eventuali modifiche possono solo influenzare l'efficienza, non la correttezza dei risultati.

Esempio.

Problema: *Prendere l'ascensore*.

Linguaggio imperativo

- Attendere che l'ascensore arrivi al piano di chiamata
- Aprire la porta dell'ascensore
- Entrare nell'ascensore
- Chiudere la porta dell'ascensore
- Spingere il tasto corrispondente al piano da raggiungere

N.B.: Ai fini del risultato, conta l'ordine in cui sono elencate le azioni da intraprendere.

Linguaggio dichiarativo

- Se l'ascensore è arrivato e la porta è aperta, allora si può entrare nell'ascensore
- Se si vuole aprire la porta dell'ascensore, bisogna aspettare che l'ascensore arrivi
- Se si è entrati e la porta è aperta, allora la si può chiudere
- Se si è entrati e la porta è chiusa, allora si deve spingere il tasto corrispondente al piano da raggiungere

N.B.: L'ordine in cui le regole sono elencate non cambia il risultato.

Nella programmazione logica si deve abbandonare il modo di pensare orientato al processo. Il programma è una descrizione della soluzione, non del processo, e si costruisce descrivendo in un linguaggio formale l'area applicativa, ossia gli oggetti che in essa esistono, le relazioni fra loro e i fatti che li riguardano.

Gli oggetti possono essere concreti o astratti, esistenti o immaginari.
Le relazioni sono qualità o attributi di un gruppo di oggetti, che legano gli uni agli altri.

I fatti e le regole asseriti costituiscono le assunzioni del programma (ossia la base di conoscenza del sistema).
I programmi si mettono in funzione ponendo al sistema delle domande circa gli oggetti del dominio e le loro relazioni.

Perché la macchina tragga dalla base di conoscenza conclusioni o risposte non esplicitamente scritte, serve una procedura di inferenza (o deduzione), che è indipendente dalle applicazioni. Di qui segue la possibilità di sviluppo incrementale della base di conoscenza.
Le conclusioni rilevanti possono essere memorizzate nella base di conoscenza per uso successivo.

Il linguaggio formale alla base della programmazione logica è rappresentato dalle *clausole di Horn*.

PROLOG

Nato grazie alle ricerche di Colmerauer e Kowalski, il *Prolog* (acronimo per PROgramming in LOGic) è, come si deduce dal nome stesso, un linguaggio di programmazione logica. E' molto potente e flessibile, ed È adatto particolarmente all'Intelligenza Artificiale e alla programmazione non numerica. E' indicato soprattutto per problemi che riguardano oggetti, specie se strutturati, e relazioni fra essi.

Il programmatore pone la sua attenzione sugli oggetti e sulle relazioni che li legano, esprimendo la conoscenza ad essi relativa sotto forma di fatti e di regole, per poi poter porre domande.
L'interprete Prolog possiede un meccanismo di inferenza che tenta di rispondere alle domande ponendole in relazione con i fatti e le regole della base di conoscenza e tentando di eseguire delle deduzioni.

Dunque, le componenti fondamentali del Prolog sono:

- dichiarazioni di *fatti* sugli oggetti e le loro relazioni
- dichiarazioni di *regole* sugli oggetti e le loro relazioni
- *domande* sugli oggetti e le loro relazioni.

Tramite esse il programmatore interagisce col linguaggio: con le prime due scrive i programmi, con l'ultima li esegue.

Programmare in Prolog equivale a descrivere il dominio del problema tramite fatti e regole sotto forma di clausole di Horn.

PROLOG - fatti

I *fatti* Prolog sono clausole di Horn non condizionali.
Essi, cioè, esprimono un'affermazione (compiuta) che non È vincolata alla preventiva verifica di un insieme di condizioni.

Esempio.

p :- q, r. "*p* è vera a condizione che siano verificate *q* ed *r*".

p :- . "*p* è vera a condizione che ... (nessuna)". Scriveremo semplicemente

p.

Detto in maniera più informale, un fatto è una frase riguardante qualcosa o qualcuno, che stabilisce una relazione fra gli argomenti e che può essere vera o falsa.

Il Prolog assume implicitamente che tutti i fatti presenti nella base di conoscenza siano veri.
La verità o falsità di un fatto asserito nella base di conoscenza di un programma Prolog è, quindi, responsabilità esclusiva del programmatore.

Esempio.
La base di conoscenza:

rosso (sangue). rosso (neve).

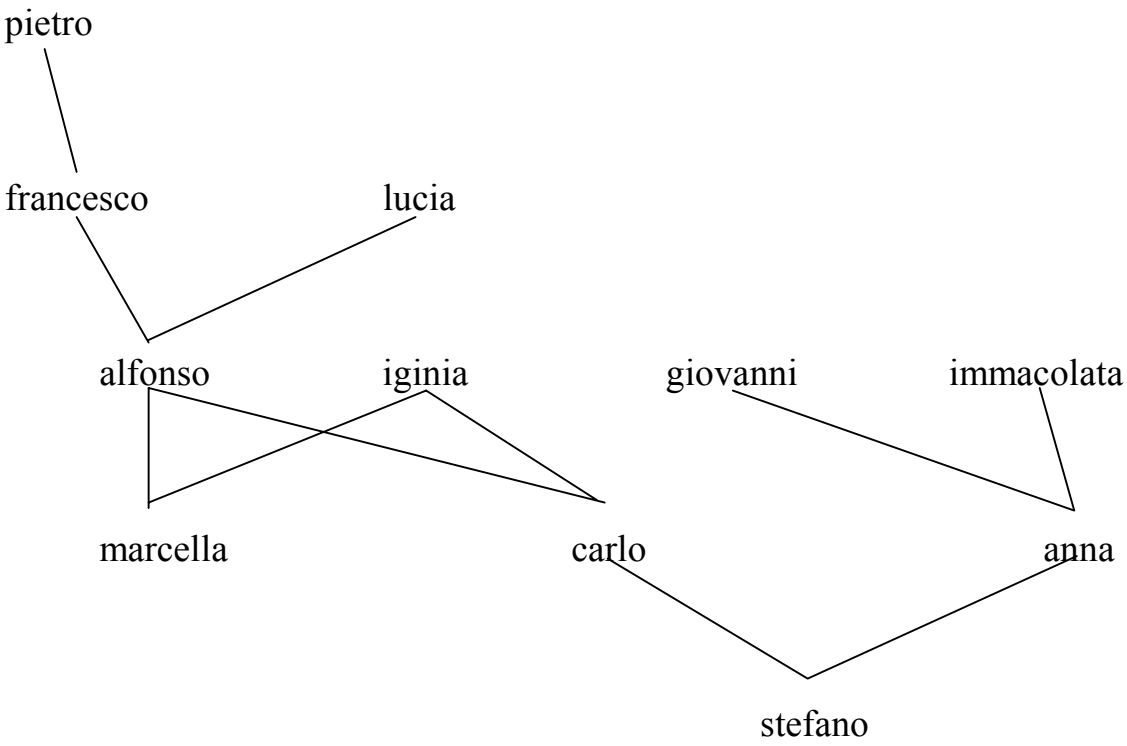
è valida, ed il Prolog riterrà che sia il sangue che la neve godano della proprietà espressa dal predicato *rosso*.

Una relazione è, in genere, definita come l'insieme di tutte le sue istanze.

Esempio.
Consideriamo la relazione *genitore*, composta dalle seguenti istanze (sulla destra è riportata la rappresentazione corrispondente in un database):

genitore (carlo, stefano).
genitore (alfonso,marcella).
genitore (anna, stefano).
genitore (francesco, alfonso).
genitore (lucia, alfonso).
genitore (iginia, carlo).
genitore (alfonso, carlo).
genitore (iginia, marcella).
genitore (pietro, francesco).
genitore (giovanni, anna).
genitore (immacolata, anna).

GENITORE	
carlo	stefano
alfonso	carlo
anna	stefano
francesco	alfonso
lucia	alfonso
iginia	carlo
alfonso	marcella
iginia	marcella
pietro	francesco
giovanni	anna
immacolata	anna



PROLOG - sintassi

Essendo un linguaggio formale, il Prolog richiede che nella specifica dei programmi vengano rispettate delle norme lessicali e sintattiche.
Iniziamo a vederne alcune:

- I nomi di predicati e oggetti devono necessariamente iniziare con una lettera minuscola.

Esempio.

Genitore	Carlo	***NO!***
genitore	carlo	***SI***

- Un'asserzione è composta dal nome del predicato che la esprime, seguito immediatamente dagli argomenti, fra parentesi tonde e separati da virgole.

Esempio.

genitore (carlo, stefano). "Carlo è genitore di Stefano".

N.B.: su alcune verisoni del Prolog è possibile usare predicati in notazione infissa.

- Ogni asserzione termina con un punto.
Questo è il simbolo che indica al Prolog la fine di un elemento di conoscenza.

- Il nome di un predicato o di un argomento deve formare un blocco unico, per cui se dovesse contenere degli spazi (ad es., se è composto da più parole) questi andrebbero eliminati o rimpiazzati con un trattino "_".

Esempio.

part of figlio di è sposato con

non sono validi; devono diventare, ad es.,

part_of figlio_di sposato_con

oppure

partof figlio sposatocon.

PROLOG - interpretazione

Il prolog tratta ogni relazione come una pura entità sintattica.
Ciò vuol dire che il *significato* di una relazione è dato dal programmatore (intendendo con ciò ordine degli argomenti, modo in cui vengono messi in relazione, nome), ed è a suo carico usarla sempre in modo coerente con tale significato.

Una volta definita l'interpretazione, essa va sempre rispettata e non può essere cambiata, pena la perdita di significato delle risposte.

Esempio.

genitore (x, y). "*x* è genitore di *y*"

È l'interpretazione usata nella relazione definita precedentemente.
Tutte le coppie dovranno avere il genitore in prima posizione ed il figlio in seconda, altrimenti il Prolog confonderebbe i loro ruoli.
Se avessimo scritto

genitore (stefano, carlo).

intendendo con ciò che "Il genitore di Stefano è Carlo", il sistema avrebbe comunque "capito" il contrario, in contrasto con la corretta genealogia, espressa dal grafo.

Una *relazione* è un insieme di asserzioni con uguale predicato e numero di argomenti.
La *base di conoscenza* È costituita dall'insieme delle relazioni definite.

PROLOG - interrogazioni

Il primo passo, nella programmazione Prolog, consiste nell'inserimento della base di conscenza tramite un apposito *ambiente di editing*.
Successivamente sarà possibile interrogarla (il che equivale ad eseguire il programma) in quello che viene chiamato *ambiente di query*.

In ambiente di query, il sistema presenta il seguente *prompt*:

? -

con il quale avverte che è pronto a rispondere a delle eventuali domande riguardanti la base di conoscenza in suo possesso.

Tali domande, in Prolog, vanno sotto il nome di *goal* (ossia obiettivi da dimostrare) o *query* (ossia interrogazioni da soddisfare), e corrispondono a clausole di Horn senza la conclusione (negazioni).

Esempio.

? - genitore (carlo, stefano).
? - genitore (carlo, stefano).

Posta una domanda, il Prolog risponde **Yes** o **No** per confermarne o meno la validità all'interno della base di conoscenza.

Esempio.

? - genitore (carlo, stefano).
Yes.
? - genitore (giovanni,carlo).
No.

E' possibile aggiornare la base di conoscenza aggiungendo, togliendo o modificando relazioni, senza per questo dover mutare il programma che le elabora.

E' anche possibile porre al Prolog delle interrogazioni multiple, separandole tramite delle virgole.

Esempio.

? - genitore (carlo, stefano), genitore (alfonso, carlo).
Yes.
? - genitore (carlo, stefano), genitore (giovanni,carlo).
No.

In definitiva, in Prolog la virgola può assumere due significati distinti:

- *simbolo di punteggiatura*, per separare argomenti di una relazione;
- *congiunzione* ("e", equivalente all'operatore logico AND), per separare fatti.

In un'interrogazione la virgola separa goal differenti che devono essere soddisfatti uno alla volta, in sequenza, per poter rispondere alla domanda. Va notato che, in tale sequenza, ai fini del risultato non conta l'ordine in cui i vari goal vengono specificati.

Esempio.

? - genitore (carlo, stefano), genitore (alfonso, carlo).

È equivalente a

? - genitore (alfonso, carlo), genitore (carlo, stefano).

In entrambi i casi il Prolog risponderà:

Yes.

? - genitore (carlo, stefano), genitore (giovanni,carlo).

È equivalente a

? - genitore (giovanni,carlo), genitore (carlo, stefano).

In entrambi i casi il Prolog risponderà:

No.

PROLOG - variabili

Un fatto, oltre che esprimere un'affermazione riguardante dei particolari individui, può rappresentare una considerazione generale, riguardante tutti gli elementi di un certo insieme.

Esempio.

"Tutti sono cattivi".
"Se qualcuno abita a Roma, allora vive in Italia".

In Prolog, il concetto che viene espresso dai pronomi indefiniti "tutti" (nei fatti) e "qualcuno" (nelle regole) si esprime tramite le *variabili*.

Una variabile è una sequenza qualsiasi di caratteri che iniziano con una lettera maiuscola o un carattere di sottolineatura "_".

Esempio.

```
cattivo (X).
vive (X, italia) :- abita (X, roma).
```

In particolare, una variabile che inizi con un carattere di sottolineatura è considerata sempre nuova, anche se occorre più volte in uno stesso elemento di conoscenza.

Esempio.

da_lavare(X) :- sporco (X), nero (X).

"Se qualcosa è sporco e nero, allora quel qualcosa va lavato".

da_lavare(X) :- sporco (X), nero (_).

"Se qualcosa è sporco ed esiste qualcosa di nero, allora la cosa sporca va lavata".

Nel primo caso la cosa sporca e quella nera coincidono; nel secondo non necessariamente.

interessante (mondo) :- bello (_), divertente (_).

"Il mondo è interessante se c'è qualcosa di bello e qualcosa di divertente".

La cosa bella e la cosa divertente potrebbero anche essere diverse.

Le variabili possono anche essere usate nelle interrogazioni, per far sì che il Prolog non risponda solo **Yes/No**, ma ci dica anche, se la risposta è affermativa, per quali oggetti la domanda è soddisfatta.

Esempio.

? - genitore (X, francesco).
X = pietro.
? - genitore (anna, Figlio).
Figlio = stefano.
? - genitore (GENITORE, giovanni).
No.

Se ci sono più risposte possibili, il Prolog risponde con la prima, e chiede se se ne vogliono conoscere altre. Quando non ce ne saranno più, risponderà **No**.
Esiste un particolare predicato, *setof*, che consente di conoscere direttamente tutti i valori che soddisfano l'interrogazione.

Esempio.

? - genitore (X, marcella).
X = alfonso. More? y.
X = iginia. More? y.
No.

? - setof (X, genitore (X, marcella), Z).
Z = alfonso, iginia.

In una interrogazione la variabile è usata come argomento, e rappresenta oggetti non noti a priori, che saranno determinati in seguito dal sistema (in tal caso si dicono *non istanziate*). Il sistema, analizzando la base di conoscenza, cerca, se esistono, soluzioni; se le trova, le associa alla variabile (che diventa *istanziata*), l'interrogazione è soddisfatta e il risultato trovato È il valore della variabile istanziata.

PROLOG - regole

Le regole, in Prolog, indicano situazioni di carattere generale servendosi di oggetti e relazioni tra essi. Esse evitano ridondanza e spreco di risorse nella base di conoscenza.

Esempio.
Abbiamo la relazione *genitore* definita precedentemente; se vogliamo esprimere la relazione *figlio* sulla stessa genealogia, invece che ridefinire tutte le coppie (magari invertendole), È sufficiente esprimere il fatto che "Se Y è genitore di X, allora X è figlio di Y" tramite la seguente regola:

```
figlio (X, Y) :- genitore (Y, X).
```

Una regola è costituita da una parte condizione (*corpo*, a destra) e una parte conclusione (*testa*, a sinistra), separate dal simbolo **:-** ("Se", che rappresenta l'implicazione).

Un fatto è una cosa sempre, incondizionatamente, vera; una regola specifica una cosa vera a patto che la condizione sia soddisfatta.

Esempio.

```
intelligente (mario).
```

"Mario È intelligente".

```
promosso (X) :- studioso (X).
```

"Condizione perchÉ qualcuno sia promosso È che sia studioso".

La testa descrive il fatto che si vuole definire, il corpo la congiunzione di obiettivi che devono essere soddisfatti, in sequenza, perchÉ la testa risulti vera. Se si scambiano, il significato muta totalmente.

Esempio.

```
studioso (X) :- promosso (X).
```

"Se qualcuno È stato promosso, allora È studioso".

PROLOG - rappresentazione grafica

E' possibile rappresentare la definizione di relazioni in termini di altre relazioni tramite grafi, in cui:

- *nodi* = oggetti (ossia argomenti delle relazioni)

- *archi* = relazioni binarie (cioè a due posti)

- *orientamento degli archi*: dal primo al secondo argomento

- *relazioni unarie*: si marcano semplicemente gli oggetti corrispondenti con il nome della relazione.

- *relazioni da definire*: archi tratteggiati

Esempio.
Supponiamo di voler rappesentare graficamente la base di conoscenza composta dalla relazione unaria

maschio (X).

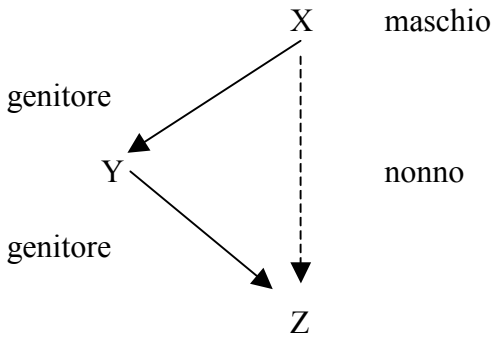
dalla relazione binaria

genitore (X, Y).

e dalla regola che definisce la relazione binaria

nonno (X, Y) :- genitore (X, Z), genitore (Z, Y), maschio (X).

Avremo:



PROLOG

Riassumendo:

- i programmi Prolog possono essere estesi semplicemente aggiungendo nuove clausole;

- le clausole Prolog possono essere di tre tipi: fatti, regole, domande;
- i fatti dichiarano cose sempre, incondizionatamente vere;
- le regole dichiarano cose vere in dipendenza da una data condizione;
- l'utente può chiedere al programma quali cose sono vere, tramite domande;

- le clausole Prolog sono formate da una testa e un corpo; quest'ultimo consiste di una lista di goal separati da virgole (da intendersi come congiunzioni logiche);
- i fatti hanno solo la testa, le domande solo il corpo, le regole sia la testa che il corpo;

- durante l'elaborazione una variabile può essere istanziata, cioÈ sostituita da un altro oggetto;

- le variabili assumono quantificate universalmente ("*Per tutte*"), tranne quelle che si trovano nel corpo delle regole (che si assumono quantificate esistenzialmente, "*Esiste* (almeno) *un*").

PROLOG

Le regole si dicono *ricorsive* se definiscono una relazione in termini di se stessa.

Esempio.
La relazione *antenato* è ricorsiva:

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

"X È antenato di Y se X È genitore di Y oppure di un antenato (Z) di Y".

La ricorsione consente di ottenere programmi meno lunghi e più generali. Inoltre, le definizioni ricorsive sono indispensabili alla soluzione di problemi complessi o di natura intrinsecamente ricorsiva.

Esempio.
Se volessimo esprimere la relazione *antenato* senza far uso della ricorsione, dovremmo dire che:

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

antenato (X, Y) :- genitore (X, Z), genitore (Z, W), genitore (W, Y).
...

Si nota immediatamente come questo programma sia molto più prolisso e ridondante del precedente e sia comunque meno generale, non potendo continuare all'infinito ad aggiungere generazioni.

Il Prolog usa molto facilmente definizioni ricorsive, che sono uno dei suoi principi fondamentali.

PROLOG

In versioni alternative di una relazione si possono usare nomi di variabili diversi, purché lo si faccia in maniera coerente (cioè corrispondano allo stesso posto).

Esempio.

antenato (A, B) :- genitore (A,B).
antenato (X,Y) :- genitore (X,Z), antenato (Z, Y).

In entrambe le regole dovremo intendere che il primo argomento è l'antenato, ed il secondo il discendente. Questa condizione è dettata dalla seconda regola, in quanto la prima sarebbe comunque coerente sotto entrambe le interpretazioni.

Viceversa, la stessa variabile, usata in diversi elementi di conoscenza, non implica alcun legame fra i due oggetti rappresentati.

Esempio.

fratello (Y, X) :- genitore (Z, Y), genitore (Z, X).

"Y È fratello di X se sono entrambi figli dello stesso genitore Z".

fratello (X,Y) :- fratello (Y,X).

"Se Y è fratello di X allora è vero anche il viceversa".

N.B.: In base alla definizione data, chiunque è fratello di se stesso. Per avere la descrizione esatta del concetto, bisogna precisare che i due figli devono essere diversi:

fratelli (Y,Z) :- genitore (X, Y), genitore (X, Z), diversi (Y, Z).

Tecnicamente, si dice che *il campo d'azione di una variabile è locale alla regola a cui appartiene*.

PROLOG

Come Prolog risponde alle domande.

Domanda = sequenza di uno o più goals da soddisfare

Soddisfare un goal = dimostrare che il goal è vero (ossia segue logicamente dalla base di conoscenza del programma).

Se la domanda contiene variabili, Prolog cerca di trovare gli oggetti particolari che, sostituiti alle variabili, consentono di soddisfare il goal.
Se il Prolog non riesce a soddisfare tutti i goals della domanda, risponde **No**.

Esempio.
Nella base di conoscenza composta dalla relazione *genitore* , poniamo al Prolog le seguenti domande:

? - genitore (immacolata, anna), genitore (lucia, alfonso), antenato (giovanni, stefano).
Yes.

? - genitore (lucia, anna), genitore (stefano, mario), antenato (giovanni, carlo).
No.

? - genitore (giovanni, anna), genitore (marcella, alfonso), antenato (giovanni, stefano).
No.

Interpretazione matematica di un programma Prolog.
Prolog accetta fatti e regole come un insieme di assiomi, e la domanda come un teorema da provare, e deve dimostrare che il teorema può essere logicamente derivato dagli assiomi.

PROLOG - strategia di controllo

Se l'obiettivo da dimostrare si compone di più goals, Prolog cerca di soddisfarli partendo da quello più a sinistra e procedendo, man mano che riesce a dimostrarli, verso quelli a destra; al primo goal che non riesce a dimostrare blocca l'esecuzione e risponde **No**, altrimenti (se riesce a dimostrarli tutti) risponde **Yes** e, se la domanda conteneva variabili, comunica i valori di tali variabili per cui la risposta è affermativa.

Sfruttando le regole, ogni goal viene sostituito da nuovi goals finché non diventano fatti semplici.

Se ci sono percorsi alternativi (cioè più clausole che definiscono la stessa relazione), Prolog sceglie per primo quello che compare prima nel programma.

Se si arriva ad un punto morto (ossia nessuna clausola ha la testa che corrisponde con il goal da dimostrare), il goal fallisce e Prolog torna al "bivio" immediatamente precedente (*backtracking*) per cercare un percorso alternativo.
Questo avviene anche quando la domanda è verificata da più oggetti e ne si vuole conoscere più di uno.

Esempio.

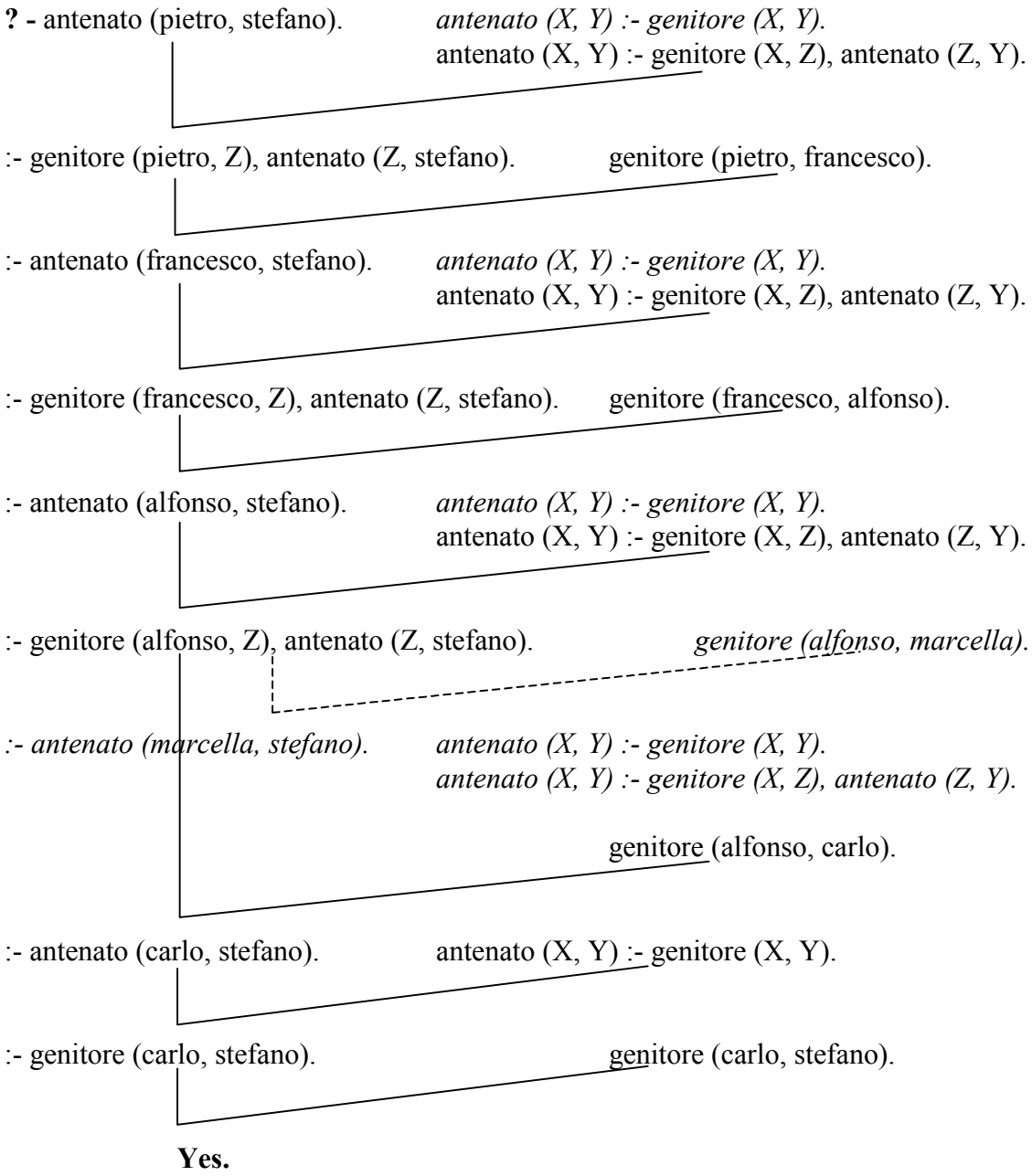
Sfruttiamo la base di conoscenza composta dalla relazione *genitore* e dalle regole per la relazione *antenato* definite in precedenza.

In tale ambito, poniamo al Prolog la seguente domanda:

? - antenato (pietro, stefano).

e vediamo come fa a rispondere.

La risposta che ci aspettiamo in base alla genalogia è **Yes**.



PROLOG - liste

L'unica struttura dati presente in Prolog, molto potente e flessibile, è la *lista*, ossia una sequenza di informazioni sulle quali non è posto alcun tipo di vincolo.

Essa è rappresentata ponendo i suoi elementi fra parentesi quadre, e separandoli tramite virgole.

Si suole usare la notazione [X|Y] per indicare una lista in cui il primo elemento è X (detto *head*) e la lista dei restanti elementi è Y (detta *tail*).

Ogni argomento di una relazione Prolog può essere una lista.

In particolare, una lista può a sua volta entrare a far parte di un'altra lista, come elemento.

Essendo la lista una sequenza, può anche contenere ripetizioni.

Esempio.
[stefano, 1972, [a, e, i, o, u], "ciao", [X,[bianca(neve), rosso (sangue)],X], "1,5"]
È una lista valida.
Se ci riferiamo ad essa con [H|T], avremo:

H = stefano;
T = [1972, [a, e, i, o, u], "ciao", [X,[bianca(neve), rosso (sangue)],X], "1,5"]

Inoltre, sono clausole valide:

```
colori_bandiera (italia, [bianco, rosso, verde]).
genitori ([X, Y], Z) :- padre (X, Z), madre (Y, Z).
```

PROLOG - negazione

In Prolog esiste la possibilità di avere dei goals negati, sia nelle domande che nel corpo delle clausole. Il predicato che si occupa di gestire tale situazione è *not*, che richiede come argomento il goal che non deve essere verificato.

Esempio.
Potremmo esprimere la regola che definisce la relazione *fratelli*, vista in precedenza, come segue:

```
fratelli (X, Y) :- genitore (Z, X), genitore (Z, Y), not(uguali (X, Y)).
```

Inoltre, sarebbero perfettamente corrette domande del tipo:

```
? - genitore (carlo,stefano), not(antenato (giovanni, alfonso)).
```

Yes.
? - genitore (anna,stefano), not(antenato (giovanni, anna)).
No.

Quando incontra un goal negato il Prolog cerca di dimostrare il goal stesso nella base di conoscenza: se ci riesce, la negazione fallisce; se non ci riesce, la negazione È ritenuta vera.
Ciò significa che viene considerata falsa qualunque cosa non sia esplicitamente detta o comunque ricavabile dalla base di conoscenza, anche se questa dovesse essere vera in sé (*Ipotesi del mondo chiuso*).

Esempio.
Per rispondere alla domanda
? – not(antenato (pietro, stefano)).
il Prolog usa la stessa dimostrazione vista in precedenza; poiché essa aveva successo, la risposta sarà **No**.
Inoltre, in quella base di conoscenza, il Prolog risponderà **Yes** anche alla domanda
? - not(fratelli (carlo, marcella)).
in quanto, non essendo definita la relazione *fratelli*, esso non riesce a dimostrare il goal.

Se il goal negato contiene variabili, la negazione verrà assunta falsa se esiste almeno un valore di quelle variabili che renda vero il goal negato.

Esempio.
Nella solita base di conoscenza, avremo:
? - not (genitore (stefano, X)).
Yes.
? - not (genitore (X, stefano)).
No.
Quest'ultima risposta può sembrare strana se si intende la domanda nell'accezione comune di:
"Esiste qualcuno che non è genitore di Stefano? (e, se si, chi è?)"
ma diventa perfettamente logica se si tiene conto dell'interpretazione Prolog, che è:
"E' vero che non esiste alcun genitore di Stefano?".