Linguaggi di Programmazione

Marco Natali

Indice

Capitolo 1

Introduzione ai Paradigmi di Programmazione

Nel corso di Linguaggi di Programmazione ci occupiamo di 3 importanti paradigmi di di linguaggi quali i linguaggi logici, linguaggi funzionali e linguaggi predicativi.

Durante il corso, e soprattutto durante la vita futura lavorativa, si devono rispettare le regole standard di formattazione dei programmi come le regole google, regole gnu ed ecc... ossia nei programmi devono essere rispettate le seguenti regole:

- usare Emacs come editor in quanto indenta lui al meglio
- le linee di codice non devono essere più lunghe di 80 colonne
- inserire spazi tra gli operatori e uno spazio dopo , e ; ossia 2 + 3 e (4, 5)
- non inserire uno spazio tra il nome di una funzione/predicato e le parentesi ossia foo(4)
- inserire uno spazio tra un istruzione di controllo e logica e le parentesi ossia if (4 || 5)
- non usare i commenti all'interno di un box perchè difficilmente mantenibile

I linguaggi di programmazione vengono classificati in base a dei paradigmi, ossia un modello di riferimento su come strutturare un programma, al fine di facilitare l'apprendimento di linguaggi similari:

- Imperative Languages: linguaggi basati sull'architettura di Von Neumann, base di tutti i calcolatori moderni, in cui il processore ha il compito di leggere e scrivere le celle di memoria durante l'esecuzione della computazione. Questa tipologia di linguaggi utilizza uno stile prescrittivo, ossia i programmi iterativi specificano una sequenza di istruzioni da eseguire per modificare lo stato del sistema e questo flusso può essere modificato soltanto con le strutture di controllo. Fanno parte di questo paradigma il C/C++, i linguaggi Assembler, Pascal, Python, ...

- Logic Languages: linguaggi in cui un programma è una deduzione logica ossia ogni istruzione è una formula del linguaggio e noi interroghiamo il sistema per sapere se una formula del linguaggio fa parte della conoscenza rappresentata nel programma; questo paradigma viene utilizzato per la dimostrazione della correttezza di un programma, per rappresentare i database ed infine sta avendo un notevole utilizzo ultimamente nel campo dell'AI(Artificial Intelligence). Fa parte di questo paradigma principalmente solo il Prolog e i suoi derivati.

- Functional Languages: linguaggi in cui il concetto di funzione è l'unica cosa importante per cui i programmi definiscono funzioni e l'esecuzione consiste nell'applicazione di uno o più funzioni agli argomenti. Fanno parte di codesto paradigma i linguaggi Lisp, Common Lisp, Scheme, Javascript, ML, Ocaml, Haskell, R, ...

Ogni dei seguenti paradigmi può avere la presenza di linguaggi object oriented, come Java, C++, Comon Lisp, Python, ..., infatti il paradigma ad oggetti è ortogonale alla definizione dei seguenti paradigmi, anche se presume alcune features tipiche dei linguaggi imperativi.

1.1 Paradigma Imperativo

Il paradigma imperativo, come già visto durante la definizione dei diversi paradigmi, si occupa di eseguire una sequenza di istruzioni per effettuare la computazione infatti il paradigma imperativo è stato inventato più per la computazione "numerica" rispetto agli altri paradigmi.

La struttura del programma è composta da due componenti, una per rappresentare le strutture dati e l'altra per gli algoritmi:

- dichiarazione: istruzioni in cui vengono dichiarate le variabili, i tipi e le funzioni necessarie per effettuare la computazione, ossia vengono definite le strutture dati necessarie per eseguire la computazione voluta.

- definizione: istruzioni in cui vengono implementati gli algoritmi, per eseguire correttamente la computazione, attraverso l'utilizzo di istruzione facenti parte del linguaggio.

1.2 Paradigma Logico

La necessità di gestire le applicazione ad un maggiore livello di astrazione e di scrivere programmi il più concisi possibile ha spinto alla creazione di nuovi paradigmi, come quello logico che analizzeremo ora.

Il paradigma logico non è più basato sull'architettura di Von Neumann ma su concetti matematici, come la logica matematica, ed utilizza uno stile di programmazione descrittivo, ossia viene definito cosa fa parte e cosa no del problema da rappresentare.

Inoltre nella programmazione logica non è presente alcuna separazione netta tra gli algoritmi e le strutture dati e, come già visto nella separazione tra i vari paradigmi, un programma consiste nel descrivere un problema come una serie di

sentenze del linguaggio ed interrogare il sistema, il quale effettua una deduzione sulla base della conoscenza rappresentata.

Un esempio di programma logico è il seguente:

1.3 Paradigma Funzionale

Il paradigma funzionale, come visto nel paradigma logico, viene usato per una programmazione simbolica, è basata su concetti matematici, adotta solitamente uno stile descrittivo di programmazione ed infine non vi è una completa separazione tra strutture dati ed algoritmi.

Il concetto fondamentale di questo paradigma è la funzione, che è una relazione tra due insiemi che associa un elemento del dominio uno e un solo elemento del codominio.

Dopo che è stata definita una funzione, possiamo applicarla su un elemento del dominio per ottenere un valutazione della funzione per cui un programma funzionale si riduce alla valutazione di una funzione per ottenere un valore, in cui in un linguaggio funzionale puro esso è determinato soltanto dalla funzione e non da i valori in memoria.
Questa assenza di effetti dati dalla memoria consiste nel definire una variabile come una costante matematica ossia il valore non è mutabile a differenza delle variabili nei linguaggi imperativi che sono una astrazione di una locazione in memoria.

Un programma funzionale, come già visto nella definizione dei vari paradigmi, consiste nella definizione di un insieme di funzioni, eventualmente ricorsive, e l'esecuzione di un programma consiste nell'applicazione di una funzione agli argomenti.

Esempio di programma funzionale:

1.4 Richiami sugli ambienti RunTime e architettura degli elaboratori

Per eseguire un programma in un qualsiasi linguaggio il sistema (ovvero il sistema operativo) deve mettere a disposizione un ambiente *run time*, che può essere anche una macchina virtuale, il quale fornisce almeno due funzionalità:

- mantenimento dello stato della computazione (program counter, limiti di memoria etc)
- gestione della memoria disponibile (fisica e virtuale)

La gestione della memoria avviene usando due aree concettualmente ben distinte con funzioni diverse:

- lo *Stack* serve per la gestione delle chiamate (soprattutto ricorsive) a procedure, metodi, funzioni etc...(record di attivazione).
- lo *Heap* serve per la gestione di strutture dati dinamiche (liste, alberi etc...)

I linguaggi logici e funzionali (ma anche Java) utilizzano pesantemente lo Heap dato che forniscono come strutture dati built-in liste e spesso vettori di dimensione variabile.

La gestione dei record di attivazione è stata affrontata nei corsi di Programmazione ed Architettura, a cui si può consultare i libri e gli appunti per rinfrescare la memoria.

La gestione della memoria, in particolare quella dinamica, può avvenire in maniera automatica, attraverso il *Garbage Collector* che è usato da Python, Java, Lisp, Prolog ed altri, oppure in maniera manuale, come in C/C++ attraverso i comandi malloc(new) e free(delete).

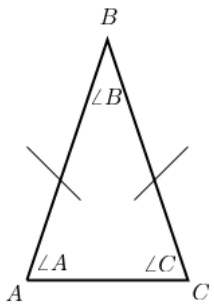
Capitolo 2

Ripasso Logica Proposizionale e Predicativa

Effettuiamo ora un ripasso della logica proposizionale e predicativa, affrontata nel corso Fondamenti dell'Informatica, al fine di rivedere e perfezionare i concetti di logica necessari per la comprensione e la scrittura di programmi logici.

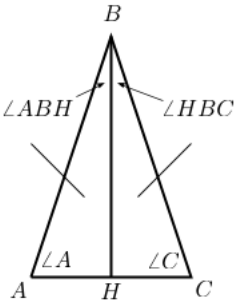
Partiamo con l'esempio di una semplice dimostrazione geometrica effettuata con le regole della logica:

Teorema 2.1. *Dato un triangolo isoscele (con $\overline{AB} = \overline{BC}$) si ha che $\angle A$, ovvero l'angolo in A, e $\angle C$, ovvero l'angolo in C, sono uguali.*



Dimostrazione. Si comincia la dimostrazione con l'elenco delle conoscenze pregresse:

1. se due triangoli sono uguali essi hanno lati e angoli uguali
2. se due triangoli hanno due lati e l'angolo sotteso uguale allora i due triangoli sono uguali
3. se viene definita la bisettrice di $\angle B$, $\overline{BH}$, si ha che $\angle ABH = \angle HBC$



Procediamo ora coi passi della dimostrazione:

1. $\overline{AB} = \overline{BC}$ per ipotesi
2. $\angle ABH = \angle HBC$ per la terza conoscenza pregressa
3. $\triangle HBC = \triangle ABH$ per la seconda conoscenza pregressa in quanto due lati sono uguali per ipotesi e dal passo precedente l'angolo sotteso ai due lati uguali è uguale in ambedue i triangoli.
4. $\angle A = \angle C$ per la prima conoscenza pregressa dato che $\triangle HBC = \triangle ABH$

Siamo così giunti alla fine della dimostrazione ma adesso vogliamo rappresentarla attraverso gli strumenti della logica al fine di rendere totalmente formale la dimostrazione per cui si effettua i seguenti passaggi:

- si è trasformata la seconda conoscenza pregressa in:
  se $\overline{AB} = \overline{BC}$ **e** $\overline{BH} = \overline{BH}$ **e** $\angle ABH = \angle HBC$ **allora** $\triangle ABH = \triangle HBC$
- si è trasformata la prima conoscenza pregressa in:
  se $\triangle ABH = \triangle HBC$ **allora** $\overline{AB} = \overline{BC}$ **e** $\overline{BH} = \overline{BH}$ **e** $\overline{AH} = \overline{HC}$ **e** $\angle ABH = \angle HBC$ **e** $\angle AHB = \angle CBH$ **e** $\angle A = \angle C$

Possiamo ora procedere col processo di formalizzazione, ossia il processo che ci permette di affermare $\overline{AB} = \overline{BC} \vdash \angle A = \angle C$ con $\vdash$ simbolo di derivazione logica, che significa consegue, allora, ecc...

Assumendo $P = \{\overline{AB} = \overline{BC}, \angle ABH = \angle HBC, \overline{BH} = \overline{BH}\}$ ed avendo le seguenti conoscenze pregresse:

1. $\overline{AB} = \overline{BC} \wedge \overline{BH} = \overline{BH} \wedge \angle ABH = \angle HBC \rightarrow \triangle ABH = \triangle HBC$
2. $\triangle ABH = \triangle HBC \rightarrow \overline{AB} = \overline{BC} \wedge \overline{BH} = \overline{BH} \wedge \overline{AH} = \overline{HC} \wedge \angle ABH = \angle HBC \wedge \angle AHB = \angle CBH \wedge \angle A = \angle C$.

per ottenere $\overline{AB} = \overline{BC} \vdash \angle A = \angle C$ bisogna effettuare la seguente catena:

1. **P1:** $\overline{AB} = \overline{BC}$ preso da **P**
2. **P2:** $\angle ABH = \angle HBC$ preso da **P**
3. **P3:** $\overline{BH} = \overline{BH}$ preso da **P**
4. **P4:** $\overline{AB} = \overline{BC} \wedge \overline{BH} = \overline{BH} \wedge \angle ABH = \angle HBC$ preso da **P1, P2, P3** e dalla regola **introduzione della congiunzione**.

5. **P5:** $\triangle ABH = \triangle HBC$ da **P4**, dalla **regola 2** e dalla regola di inferenza detta **modus ponens**.
6. **P6:** $\overline{AB} = \overline{BC} \wedge \overline{BH} = \overline{BH} \wedge \overline{AH} = \overline{HC} \wedge \angle ABH = \angle HBC \wedge \angle AHB = \angle CBH \wedge \angle A = \angle C$ da **P5**, dalla **regola 1** e dalla regola **modus ponens**.
7. **P7:** $\angle A = \angle C$ da **P6** e dalla regola d'inferenza eliminazione della congiunzione

Abbiamo così dimostrato tutto anche per mezzo dei costrutti della logica $\square$

Definizione 1. *Una dimostrazione del tipo F è conseguenza di S,* ***dim****, si indica con:*

$$S \vdash F$$

ed è una sequenza:

$$dim =< P_1, P_2, ..., P_n >$$

con:

- $P_n = F$
- $P_i \in S$ *o con* P_i *ottenibile dalle* $P_1, ..., P_{i-1}$ *applicando una regola di inferenza*

Un insieme di regole di inferenza costituisce la base di un calcolo logico, il quale ha lo scopo di manipolare le formule in modo unicamente sintattico al fine di stabilire una connessione tra un insieme di formule di partenza, dette assiemi e un insieme di conclusioni.

2.0.1 Logica Proporzionale

La logica proposizionale si occupa delle conclusioni che si possono trarre da un insieme di proposizioni, che definiscono sintatticamente la logica proposizionale, ma purtroppo è un linguaggio limitato che non si può generalizzare le proposizioni.

La sintassi di un linguaggio è composta da una serie di formule ben formate(FBF) definite induttivamente nel seguente modo:

1. Le costanti e le variabili proposizionali $\in FBF$(chiamate atomi o letterali).
2. Se A e $B \in FBF$ allora $(A \wedge B)$,$(A \vee B)$,$(\neg A)$,$(A \rightarrow B)$, $(A \iff B)$,TA e FA sono delle formule ben formate.
3. nient'altro è una formula

Esempio:
$(P \wedge Q) \in Fbf$ è una formula ben formata
$(PQ \wedge R) \notin Fbf$ in quanto non si rispetta la sintassi del linguaggio definita.

La semantica di una logica consente di dare un significato e un interpretazione alle formule del Linguaggio.

Definizione 2. *Sia data una formula proposizionale P e sia $P_1, \ldots, P_n$, l'insieme degli atomi che compaiono nella formula A. Si definisce come* interpretazione *una funzione $v : \{P_1, \ldots, P_n\} \mapsto \{T, F\}$ che attribuisce un valore di verità a ciascun atomo della formula A.*

I connettivi della Logica Proposizionale hanno i seguenti valori di verità:

A	B	$A \wedge B$	$A \vee B$	$\neg A$	$A \Rightarrow B$	$A \iff B$
F	F	F	F	T	T	T
F	T	F	T	T	T	F
T	F	F	T	F	F	F
T	T	T	T	F	T	T

La tavola di verità costituisce la semantica di un insieme di proposizioni mentre un calcolo logico dice come generare nuove formule logiche, ovvero espressioni sintattiche, a partire dagli assiomi e questo processo di generazione si chiama dimostrazione.

Per ottenere nuove formule dagli assiomi si usa il calcolo proposizionale, che si base su regole di inferenza, ossia regole attraverso i quali si può derivare una nuova formula ben formata.
Le regole di inferenza analizzate sono le seguenti:

Esempio 1 (Modus Ponens)**.**

$$\frac{a \to b,\ a}{b}$$

Esempio 2 (Modus Tollens)**.**

$$\frac{a \to b, \neg b}{\neg a}$$

Esempio 3 (Eliminazione e Introduzione di $\wedge$)**.**

$$\frac{P_1 \wedge P_2 \wedge ... \wedge P_n}{P_i}\ [\textit{Eliminazione di } \wedge]$$

Esempio 4.

$$\frac{P_1, P_2, ..., P_n}{P_1 \wedge P_2 \wedge ... \wedge P_n}\ [\textit{Introduzione di } \wedge]$$

Esempio 5 (Introduzione di $\vee$)**.**

$$\frac{a}{a \vee b}$$

Esempio 6. *Ecco altre regole utili:*

- ***Terzo Escluso:***
$$\frac{a \vee \neg a}{vero}$$
- ***Eliminazione di*** $\neg$***:***
$$\frac{\neg\neg a}{a}$$
- ***Eliminazione di*** $\wedge$***:***
$$\frac{a \wedge vero}{a}$$

- ***Contraddizione:***

$$\frac{a \vee \neg a}{b}$$

ovvero da una contraddizione posso trarre qualsiasi conseguenza

Queste regole di inferenza fanno parte del calcolo naturale, detto anche di Gentzen, similare al calcolo tramite Tableaux visto nel corso di Fondamenti dell'informatica.
Questo tipo di calcolo consiste nel formalizzare i modi di derivare conclusioni a partire dalle premesse, ovvero di derivare direttamente un FBF mediante una sequenza di passi ben codificati. La regola del modus ponens insieme al principio del terzo escluso, posso essere usati anche procedendo per assurdo alla dimostrazione di una data formula e ciò viene detto *principio di risoluzione*, affrontata poi quando analizziamo il linguaggio Prolog.

Una formula nella logica proposizionale può essere di tre diversi tipi:

valida o tautologica : la formula è soddisfatta da qualsiasi valutazione della Formula

Soddisfacibile non Tautologica :la formula è soddisfatta da qualche valutazione della formula ma non da tutte.

falsibicabile :la formula non è soddisfatta da qualche valutazione della formula.

Contraddizione :la formula non viene mai soddisfatta

Teorema 2.2. *A è una formula valida se e solo se $\neg A$ è insoddisfacibile. A è soddisfacibile se e solo se $\neg A$ è falsibicabile*

Si definisce *modello*, indicato con $M \models A$, tutte le valutazioni booleane che rendono vera la formula A. Si definisce *contromodello*, indicato con , tutte le valutazioni booleane che rendono falsa la formula A.

La logica proposizionale è decidibile, ossia posso sempre verificare il significato di una formula, infatti esiste una procedura effettiva che stabilisce la validità o no di una formula, o se questa ad esempio è una tautologia. In particolare il verificare se una proposizione è tautologica o meno è l'operazione di decibilità principale che si svolge nel calcolo proposizonale.

Definizione 3. *Se $M \models A$ per tutti gli M, allora A è una tautologia e si indica $\models A$*

Definizione 4. *Se $M \models A$ per qualche M, allora A è soddisfacibile*

Definizione 5. *Se $M \models A$ non è soddisfatta da nessun M, allora A è insoddisfacibile*

Una dimostrazione di una formula di una logica può venire tramite:

- **Metodo diretto**: Data un'ipotesi, attraverso una serie di passi si riesce a dimostrare la correttezza della Tesi
- **Metodo per assurdo**(non sempre accettato in tutte le logiche): Si nega la tesi ed attraverso una serie di passi si riesce a dimostrare la negazione delle ipotesi.

Teorema 2.3. *Un apparato deduttivo R è completo se, per ogni formula $A \in Fbf$, $\vdash A$ implica $\models A$*

Teorema 2.4. *Un apparato deduttivo R è corretto se, per ogni formula $A \in Fbf$, $\models A$ implica $\vdash A$*

2.0.2 Logica del primo ordine

La logica del primo ordine, chiamata anche logica predicativa, permette di quantificare i vari fatti ed introduce il concetto di funzione e predicato per poter esprimere delle proprietà su una serie di individui.

Un linguaggio predicativo L è composto dai seguenti insiemi di simboli:

1. insieme di variabili individuali(infiniti) $x, y, z, \ldots$
2. connettivi logici $\wedge \vee \neg \rightarrow \Longleftrightarrow$
3. quantificatori $\forall \exists$
4. simboli (,)
5. Costanti proposizionali T, F
6. simbolo di uguaglianza $=$, eventualmente assente

Questa è la parte del linguaggio tipica di ogni linguaggio del primo ordine poi ogni linguaggio definisce la propria segnatura ossia definisce in maniera autonomo:

1. insiemi di simboli di costante $a, b, c, \ldots$
2. simboli di funzione con arieta $f, g, h, \ldots$
3. simboli di predicato $P, Q, Z, \ldots$ con arietà

Esempio:Linguaggio della teoria degli insiemi
Costante:$\emptyset$
Predicati:$\in (x, y)$, $= (x, y)$
Esempio:Linguaggio della teoria dei Numeri
Costante:0
Predicati:$< (x, y)$,$= (x, y)$
Funzioni:$succ(x)$,$+(x, y)$,$*(x, y)$

Per definire le formule ben formate della logica predicativa bisogna prima definire l'insieme di termini e le formule atomiche.

Definizione 6. *L'insieme $TERM$ dei termini è definito induttivamente come segue*

1. *Ogni variabile e costante sono dei Termini*
2. *Se $t_1 \ldots t_n$ sono dei termini e f è un simbolo di funzione di arietà n allora $f(t_1, \ldots, t_n)$ è un termine*

Definizione 7. *L'insieme $ATOM$ delle formule atomiche è definito come:*

1. *T e F sono degli atomi*

2. *Se t_1 e t_2 sono dei termini, allora $t_1 = t_2$ è un atomo*

3. *Se $t_1, \ldots, t_n$ sono dei termini e P è un predicato a n argomenti, allora $P(t_1, \ldots, t_n)$ è un atomo.*

Definizione 8. *L'insieme delle formule ben formate(FBF) di L è definito induttivamente come*

1. *Ogni atomo è una formula*

2. *Se $A, B \in FBF$, allora $\neg A$, $A \wedge B$,$A \vee B$,$A \to B$ e $A \iff B$ appartengono alle formule ben formate*

3. *Se $A \in FBF$ e x è una variabile, allora $\forall x A$ e $\exists x A$ appartengono alle formule ben formate*

4. *Nient'altro è una formula*

Definizione 9. *L'insieme $var(t)$ delle variabili di un termine t è definito come segue:*

- $var(t) = \{t\}$, *se t è una variabile*
- $var(t) = \emptyset$ *se t è una costante*
- $var(f(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n} var(t_i)$
- $var(R(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n} var(t_i)$

Si definisce come *aperto* un termine che non contiene variabili altrimenti si definisce il termine come *chiuso*.
Le variabili nei termini e nelle formule atomiche possono essere libere in quanto gli unici operatori che "legano" le variabili sono i quantificatori.

Il campo di azione dei quantificatori si riferisce soltanto alla parte in cui si applica il quantificatore per cui una variabile si dice *libera* se non ricade nel campo di azione di un quantificatore altrimenti la variabile si dice *vincolata*.

Si aggiunge una nuova regola d'inferenza per la logica dei predicati, l'eliminazione del quantificatore universale $\forall$:

$$\frac{\forall x, T(..., x, ...), c \in C}{T(..., c, ...)}$$

Abbiamo altre regole di inferenza per il quantificatore esistenziale:

- Introduzione del quantificatore esistenziale $\exists$:

$$\frac{T(..., c, ...), c \in C}{\exists x, T(..., x, ...)}$$

- si hanno le seguente identità:

$$\exists x, \neg T(..., x, ...) \equiv \neg \forall x, T)..., x, ...)$$

$$\forall x, \neg T(..., x, ...) \equiv \neg \exists x, T(..., x, ...)$$

Capitolo 3

Prolog:Programmazione Logica

Dopo aver effettuato un ripasso della logica, incominciamo a considerare il Prolog e la programmazione logica: le basi del PROLOG (PROgramming in LOGic) sono state poste da Robert Kowalski e Marten Van Emdem, mentre la sua progettazione e implementazione, avvenuta nel 1972 a Marsiglia attraverso Alain Colmerauer e Philippe Roussel.

Si tratta di un linguaggio di programmazione logica basato sulle Clausole di Horn, ovvero un insieme di procedure attivate da una asserzione iniziale d'obiettivo e la procedura utilizzata da prolog per la computazione è il principio di risoluzione.

Un programma prolog è un formato da un insieme di istruzioni, rappresentanti un sottoinsieme di frasi ben formate della logica del primo ordine e il sistema prolog ha il fine di determinare se una data assunzione è verificata o meno nel programma e sotto quali eventuali vincoli ciò risulta verificato.

I componenti basilari di un programma prolog, rappresentanti tutti una clausola di Horn, sono:

Fatti : indica una relazione esistente tra due oggetti, che può venire chiama predicato.

Query : chiede al sistema se una relazione esiste tra gli oggetti e quindi inizia una deduzione per stabilire se la query è una conseguenza diretta del programma, dopo l'applicazione del modus ponens universale per un numero finito di volte.

Regole : definisce una nuova relazione esistente tra gli oggetti, ossia permette di derivare una nuova conclusione da una serie di fatti e regole. la testa della regola A, viene detta conseguenza mentre il corpo $B_1, B_2, \ldots, B_n$ sono l'antecedente e il simbolo $:-$ indica il simbolo logico di implicazione. Una relazione può essere definita ricorsiva, e in questo caso necessita di due regole, una per il caso base e una per il caso passo.

I fatti e le regole sono quantificate universalmente mentre una query si intende sempre quantificata esistenzialmente, per cui una query risponde true se esiste un instanza α che verifica la query altrimenti risponde false.

Le query e le regole le abbiamo definite nella forma generale, ossia possiamo definire regole e query su congiunzioni di termini, infatti il simbolo "," rappresenta l'operatore logico and, perciò in caso più termini hanno lo stesso simbolo di variabile l'instanza α deve essere la stessa per stabilire se una query è una conseguenza diretta del programma.

Le regole stabiliscono una relazione di implicazione, indicato con il simbolo :-, infatti definiamo che risulta A se risulta verificato $B_1, B_2, \ldots, B_n$, $ovviamentesempredalpuntodivistasintattico$ $QuestarelazionestabiliscecheXilnonnodiY serisultacheesisteunZtalecheXpadrediZeZgenitorediY$.

Per riuscire a stabilire se un goals è una conseguenza diretta del programma il sistema prolog utilizza il principio di risoluzione, utilizzato per effettuare la dimostrazione del programma , e il principio di unificazione, necessario come si evince dal nome per unificare le variabili presenti in una formula del programma.

3.0.1 Principio di Risoluzione

Il principio di risoluzione è una regola di inferenza generalizzata semplice e facilmente implementabile in un calcolatore, assieme all' algoritmo di unificazione, che opera su formule ben formate nella forma normale congiuntiva, in cui i letterali si chiamano clausole, e viene utilizzata per la dimostrazione di formule ben formate attraverso la refutazione per assurdo. La regola di inferenza ha la seguente forma:

$$\frac{p \vee r,\ s \vee r}{p \vee s} \qquad \frac{\neg r,\ r}{\bot}$$

dove:

- $p \vee s$ è la *clausola risolvente*
- $\bot$ è la *clausola vuota*, che corrisponde all'aver creato una contraddizione con delle FBF,ossia posso dedurre qualsiasi cosa, compresa anche la clausola vuota(posso infatti dedurre qualsiasi cosa, anche la clausola vuota)

vediamo un'altra regola di inferenza:

Esempio 7 ((unit) resolution)**.**

$$\frac{\neg p,\ q_1 \vee q_2 \vee ... \vee q_k \vee p}{q_1 \vee q_2 \vee ... \vee q_k}$$

o anche:

$$\frac{p,\ q_1 \vee q_2 \vee ... \vee q_k \vee \neg p}{q_1 \vee q_2 \vee ... \vee q_k}$$

è una regola di risoluzione molto generale, chiamata anche procedura di Davis-Putnam, e se una delle due clausole da risolvere è un letterale si parla di unit resolution.
Come esempio si può avere:

- *non piove, piove e c'è il sole*
- *c'è il sole*

Prolog per stabilire se un goal e/o regola risulta verificata utilizza la refutazione per assurdo, ossia data una proposizione P suppone che sia falsa ed

applicando le regole di inferenza se risulta che $\neg P$ sarebbe assurdo afferma che P è verificata.

Le formule del linguaggio Prolog sono *clausole di Horn*, definite nelle seguenti regole. Ogni FBF può essere in *forma normale a clausola*:

- *formula normale congiunta:* congiunzione di disgiunzioni o di negazione di predicati (sia positivi che negativi):

$$\wedge_i({}_iL_{ij})$$

Esempio 8. *ecco degli esempi:*

- $(p(x) \vee q(x,y) \vee \neg t(z)) \wedge (p(w) \vee \neg s((u) \vee \neg r(v))$
- $(\neg t(z)) \vee (p(w) \vee \neg s(u)) \wedge (p(x) \vee s(x) \vee q(y))$

che sono riscrivibili come:

$$t(z) \rightarrow p(x) \vee q(x,y)$$

e

$$s(u) \wedge r(v) \rightarrow p(w)$$

- *forma normale disgiunta*: disgiunzione di congiunzioni o di negazione di predicati (sia positivi che negativi)

$$\vee_i(\wedge_i L_{ij})$$

Le clausole con un solo letterale positivo solo le *clausole di Horn*, con lo presenza o meno di letterali negativi, e un programma in prolog è una collezione di *clausole di Horn*.
Ovviamente non tutte le formule ben formate possono rappresentate tramite le clausole di Horn per cui il Prolog è un sottoinsieme della logica del Primo Ordine.
Attraverso questa restrizione nel 1974 Kowalski produsse un interpretazione procedurale delle dimostrazione e ciò diventò la base della semantica di ogni sistema prolog.

Ogni espressione prolog viene chiamata *Termine*, che può essere della seguente tipologia:

atomi : elemento base del sistema prolog e può essere rappresentato da un numero, qualsiasi sequenza racchiusa da (' '), una sequenza di caratteri alfanumerici, con la lettera minuscola iniziale, e nel swprolog anche una stringa indica un atomo.

variabili : sequenza di caratteri alfanumerici, iniziata con la lettera maiuscola oppure con _ e vengono inizializzate quando il sistema prova a dimostrare una query.
Una variabile composta soltanto dal simbolo _ viene detta *anonymous*.

termini composti : termine composto da un funtore, simbolo rappresentante il nome di una funzione o predicato, e da una sequenza di termini all'interno di parentesi e separati da una virgola.

Un programma caricato nel sistema prolog rappresenta una base di conoscenza e questa base può venire rappresentata in vari modi:

- tutte le informazioni vengono rappresentate come argomenti in un'unica relazione e ciò può essere comodo solo per piccole relazioni dato che è difficile da mantenere e coprendere una relazione così definita.
- le informazioni vengono rappresentate con più relazione annidate tra di loro e ciò aumenta la leggibilità e la mantenibilità del programma prolog.
- tutte le relazioni possono essere rappresentate tramite lo schema XML

3.1 Principio di Unificazione