

# Approfondimenti Matematici del Corso di Intelligenza Artificiale

Federico Bergenti

19 maggio 2023

## 1 Insiemi Numerici

In questo testo vengono indicati gli insiemi con lettere maiuscole dell’alfabeto latino, mentre gli elementi degli insiemi vengono indicati con lettere minuscole, sempre dell’alfabeto latino. Unica eccezione è la notazione per gli insiemi numerici:

- $\mathbb{N}$  è l’insieme dei *numeri naturali*, che include zero, mentre  $\mathbb{N}_+$  è l’insieme dei numeri naturali positivi;
- $\mathbb{Z}$  è l’insieme dei *numeri interi* e  $\mathbb{Z}_+$  è l’insieme dei numeri interi positivi;
- $\mathbb{Q}$  è l’insieme dei *numeri razionali* e  $\mathbb{Q}_+$  è l’insieme dei numeri razionali positivi;
- $\mathbb{R}$  è l’insieme dei *numeri reali* e  $\mathbb{R}_+$  è l’insieme dei numeri reali positivi; e
- $\mathbb{C}$  è l’insieme dei *numeri complessi*.

## 2 Vettori Reali

Dato  $n \in \mathbb{N}_+$ , un elemento  $\mathbf{x}$  dell’insieme  $\mathbb{R}^n$  viene detto *vettore reale* (o di  $\mathbb{R}^n$ )

$$\mathbf{x} = (x_1, x_2, \dots, x_n) = (x_i)_{i=1}^n \quad (1)$$

e i numeri reali  $x_i$ , con  $1 \leq i \leq n$ , sono le *componenti* (del vettore)  $\mathbf{x}$ . Si noti che  $n$  viene detto *dimensione* di  $\mathbb{R}^n$  e, quando si lavora anche con matrici, i vettori reali si comportano come *vettori colonna*. Infine, si noti che  $\mathbf{0} \in \mathbb{R}^n$  e che i numeri reali vengono spesso chiamati *scalari*.

I vettori reali possono essere moltiplicati per uno scalare e sommati, quindi  $\mathbb{R}^n$  è uno *spazio vettoriale*:

- Se  $\alpha \in \mathbb{R}$  e  $\mathbf{x} \in \mathbb{R}^n$ , allora  $\alpha\mathbf{x} \in \mathbb{R}^n$  e  $\alpha\mathbf{x} = (\alpha x_1, \alpha x_2, \dots, \alpha x_n)$ ; e
- Se  $\mathbf{x} \in \mathbb{R}^n$  e  $\mathbf{y} \in \mathbb{R}^n$ , allora  $\mathbf{x} + \mathbf{y} \in \mathbb{R}^n$  e  $\mathbf{x} + \mathbf{y} = (x_1 + y_1, x_2 + y_2, \dots, x_n + y_n)$ .

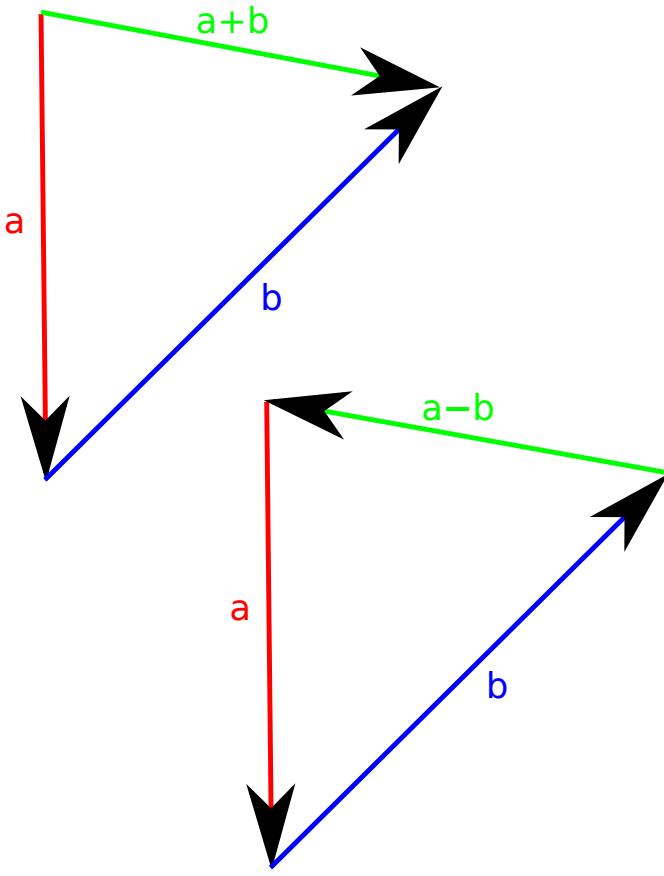


Figura 1: I vettori  $\mathbf{a} \in \mathbb{R}^2$  e  $\mathbf{b} \in \mathbb{R}^2$  e la loro somma  $\mathbf{a} + \mathbf{b}$  e differenza  $\mathbf{a} - \mathbf{b}$

Naturalmente, il prodotto per uno scalare e la somma consentono di esprimere la sottrazione tra vettori  $\mathbf{x} - \mathbf{y} = \mathbf{x} + (-1)\mathbf{y}$ .

Sui vettori reali è possibile definire la *norma euclidea* (o *lunghezza*) e, dato un qualsiasi  $\mathbf{x} \in \mathbb{R}^n$ ,

$$\|\mathbf{x}\| \triangleq \sqrt{\sum_{i=1}^n x_i^2} \quad (2)$$

dove per radice quadrata si intende, come di consueto, la radice quadrata reale positiva e quindi, per ogni  $\mathbf{x} \in \mathbb{R}^n$ , valgono le due seguenti proprietà

$$\|\mathbf{x}\| \geq 0 \quad \text{e} \quad \|\mathbf{x}\| = 0 \iff \mathbf{x} = \mathbf{0} \quad (3)$$

Si noti che un vettore la cui norma euclidea vale 1 viene detto *vettore unitario* (o *versore*). In più, si noti che  $\mathbb{R}^n$  è uno *spazio normato* perché è stata definita opportunamente la norma euclidea.

**Proposizione 1.** *Per ogni coppia di vettori  $\mathbf{x} \in \mathbb{R}^n$  e  $\mathbf{y} \in \mathbb{R}^n$  vale la seguente diseguaglianza*

$$\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\| \quad (4)$$

*che viene normalmente chiamata diseguaglianza triangolare.*

**Esempio 1.** Si consideri il caso bidimensionale, se

$$\mathbf{x} = (x_1, 0) \quad \text{e} \quad \mathbf{y} = (0, y_2) \quad (5)$$

allora

$$\|\mathbf{x}\| + \|\mathbf{y}\| = |x_1| + |y_2| \geq \sqrt{x_1^2 + y_2^2} = \|\mathbf{x} + \mathbf{y}\| \quad (6)$$

che conferma la diseguaglianza triangolare sfruttando il teorema di Pitagora.

La norma euclidea può essere utilizzata per definire la *distanza* tra due vettori reali  $\mathbf{x} \in \mathbb{R}^n$  e  $\mathbf{y} \in \mathbb{R}^n$ , rendendo quindi  $\mathbb{R}^n$  uno *spazio metrico*,

$$d(\mathbf{x}, \mathbf{y}) \triangleq \|\mathbf{x} - \mathbf{y}\| = \|\mathbf{y} - \mathbf{x}\| \quad (7)$$

che, nel caso bidimensionale e tridimensionale, è la consueta distanza tra i *punti* identificati dai vettori una volta che sia stato scelto un sistema di riferimento.

Dati due vettori reali  $\mathbf{x} \in \mathbb{R}^n$  e  $\mathbf{y} \in \mathbb{R}^n$ , si definisce *prodotto scalare* tra i vettori

$$\mathbf{x} \cdot \mathbf{y} \triangleq \sum_{i=1}^n x_i y_i \quad (8)$$

Si noti che, per ogni  $\mathbf{x} \in \mathbb{R}^n$ , valgono le due seguenti proprietà

$$\|\mathbf{x}\|^2 = \mathbf{x} \cdot \mathbf{x} \quad \text{e} \quad \mathbf{0} \cdot \mathbf{x} = \mathbf{x} \cdot \mathbf{0} = 0 \quad (9)$$

Quindi,  $\mathbb{R}^n$  è uno *spazio con prodotto interno*.

**Proposizione 2.** *Dati due vettori  $\mathbf{x} \in \mathbb{R}^n$  e  $\mathbf{y} \in \mathbb{R}^n$ , vale la seguente diseguaglianza di Cauchy-Schwarz*

$$|\mathbf{x} \cdot \mathbf{y}| \leq \|\mathbf{x}\| \|\mathbf{y}\| \quad (10)$$

e l'uguaglianza vale se e soltanto se esiste un  $\lambda \in \mathbb{R}$  tale che  $\mathbf{x} = \lambda \mathbf{y}$ .

La proposizione ha un'interpretazione geometrica interessante nel caso bidimensionale, che si estende facilmente al caso tridimensionale. In particolare, considerati  $\mathbf{x} \in \mathbb{R}^2$  e  $\mathbf{y} \in \mathbb{R}^2$

$$\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + x_2 y_2 \quad (11)$$

se ruotiamo entrambi i vettori di un angolo  $\beta \in \mathbb{R}$  in senso antiorario, allora

$$\mathbf{x}' = R_\beta \mathbf{x} \quad \text{e} \quad \mathbf{y}' = R_\beta \mathbf{y} \quad (12)$$

dove

$$R_\beta = \begin{pmatrix} \cos \beta & -\sin \beta \\ \sin \beta & \cos \beta \end{pmatrix} \quad (13)$$

e quindi, riscrivendo opportunamente  $\mathbf{x}'$  e  $\mathbf{y}'$ , si ottiene

$$\mathbf{x}' \cdot \mathbf{y}' = \mathbf{x} \cdot \mathbf{y} \quad (14)$$

da cui si evince che il prodotto scalare in  $\mathbb{R}^2$  è indipendente dalla rotazione di entrambi i vettori dello stesso angolo. Quindi, è sempre possibile calcolare il prodotto

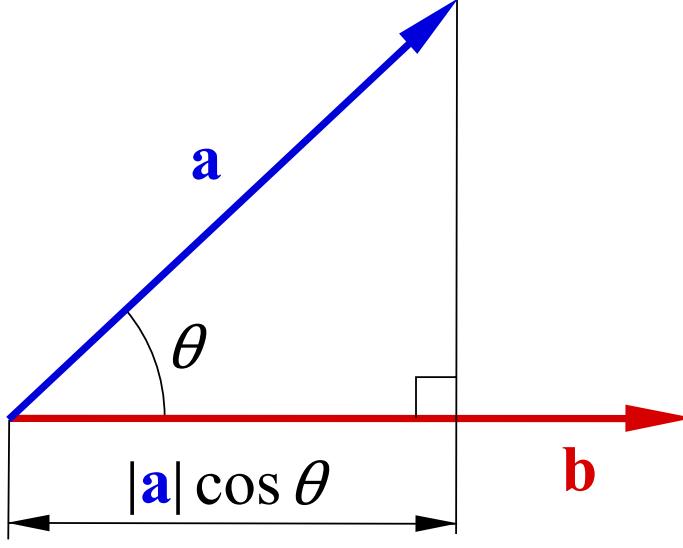


Figura 2: I vettori  $\mathbf{a} \in \mathbb{R}^2$  e  $\mathbf{b} \in \mathbb{R}^2$  e il loro prodotto scalare  $\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos \theta$

scalare tra due vettori allineando uno dei due vettori con l'asse delle ascisse. Ad esempio, se il vettore  $\mathbf{x}$  viene allineato con l'asse delle ascisse, allora

$$\mathbf{x} = (\|\mathbf{x}\|, 0) \quad \text{e} \quad \mathbf{y} = (\|\mathbf{y}\| \cos \theta, \|\mathbf{y}\| \sin \theta) \quad (15)$$

dove  $\theta \in \mathbb{R}$  è l'angolo tra i due vettori. In questa situazione,

$$|\mathbf{x} \cdot \mathbf{y}| = \|\mathbf{x}\| \|\mathbf{y}\| |\cos \theta| \leq \|\mathbf{x}\| \|\mathbf{y}\| \quad (16)$$

che conferma che l'uguaglianza vale se e soltanto se i due vettori giacciono sulla stessa retta perché, in questo caso, vale  $\cos \theta = 1$ .

Dato  $n \in \mathbb{N}_+$ , una matrice  $R = (r_{i,j})_{i,j=1}^n$  si dice *ortogonale* se e soltanto se è invertibile e vale

$$R^\top = R^{-1} \quad (17)$$

Quindi, una matrice ortogonale  $R$  rappresenta una *isometria* (applicazione lineare che preserva le lunghezze dei vettori) perché per ogni  $\mathbf{x} \in \mathbb{R}^n$  vale

$$\|R\mathbf{x}\|^2 = (R\mathbf{x}) \cdot (R\mathbf{x}) = \mathbf{x}^\top R^\top R \mathbf{x} = \|\mathbf{x}\|^2 \quad (18)$$

Si noti che il determinante di una matrice di ortogonale può valere solo  $\pm 1$ , infatti se  $R$  è una matrice di ortogonale

$$1 = \det RR^{-1} = \det RR^\top = (\det R)^2 \quad (19)$$

Una matrice ortogonale  $R$  per cui vale  $\det R = 1$  si dice *matrice di rotazione* e vale, per ogni  $\mathbf{x} \in \mathbb{R}^n$  e  $\mathbf{y} \in \mathbb{R}^n$ ,

$$R\mathbf{x} \cdot R\mathbf{y} = \mathbf{x}^\top R^\top R\mathbf{y} = \mathbf{x} \cdot \mathbf{y} \quad (20)$$

Dato  $n \in \mathbb{N}_+$ , due vettori  $\mathbf{x} \in \mathbb{R}^n$  e  $\mathbf{y} \in \mathbb{R}^n$ , entrambi diversi da  $\mathbf{0}$ , si dicono *ortogonal* se e soltanto se

$$\mathbf{x} \cdot \mathbf{y} = 0 \quad (21)$$

coerentemente con il fatto che, nei casi bidimensionali e tridimensionali, l'angolo tra i due vettori annulla il coseno utilizzato per il calcolo di  $\mathbf{x} \cdot \mathbf{y}$ .

Un insieme finito di vettori non nulli  $O \subset \mathbb{R}^n$  si dice formato da vettori *mutuamente ortogonali* se e soltanto se, qualsiasi siano  $\mathbf{x} \in O$  e  $\mathbf{y} \in O$ , vale  $\mathbf{x} \cdot \mathbf{y} = 0$ . Si noti che gli insiemi di vettori mutuamente ortogonali sono particolarmente interessanti perché vale la seguente proposizione.

**Proposizione 3.** *Dato  $n \in \mathbb{N}_+$ , se  $O \subset \mathbb{R}^n$  è un insieme di vettori mutuamente ortogonali, allora è anche un insieme di vettori linearmente indipendenti.*

Spesso si studiano dei sottoinsiemi di  $\mathbb{R}^n$  che hanno un interesse dal punto di vista dell'intuizione geometrica. Questi sottoinsiemi generalizzano gli insiemi con gli stessi nomi comunemente descritti nei casi bidimensionali e tridimensionali.

**Rette.** Dati due vettori  $\mathbf{x} \in \mathbb{R}^n$  e  $\mathbf{y} \in \mathbb{R}^n$ , la retta  $\overline{(\mathbf{x}, \mathbf{y})}$  passante per i punti identificati da  $\mathbf{x}$  e  $\mathbf{y}$  è

$$\overline{(\mathbf{x}, \mathbf{y})} = \{\mathbf{v} \in \mathbb{R}^n : \mathbf{v} = \mathbf{x} + \lambda(\mathbf{y} - \mathbf{x}) \wedge \lambda \in \mathbb{R}\} \quad (22)$$

**Segmenti.** Dati due vettori  $\mathbf{x} \in \mathbb{R}^n$  e  $\mathbf{y} \in \mathbb{R}^n$ , il segmento  $[\mathbf{x}, \mathbf{y}]$  che congiunge i punti identificati da  $\mathbf{x}$  e  $\mathbf{y}$  è

$$[\mathbf{x}, \mathbf{y}] = \{\mathbf{v} \in \mathbb{R}^n : \mathbf{v} = \mathbf{x} + \lambda(\mathbf{y} - \mathbf{x}) \wedge \lambda \in [0, 1]\} \quad (23)$$

**Iperpiani.** Dato il vettore  $\mathbf{x} \in \mathbb{R}^n$  e il versore  $\mathbf{n} \in \mathbb{R}^n$ , l'*iperpiano* che contiene  $\mathbf{x}$  e ha  $\mathbf{n}$  come (versore) *normale* è

$$\mathcal{H}_{\mathbf{n}}(\mathbf{x}) = \{\mathbf{v} \in \mathbb{R}^n : (\mathbf{v} - \mathbf{x}) \cdot \mathbf{n} = 0\} \quad (24)$$

**Scatole.** Dati due vettori  $\mathbf{x} \in \mathbb{R}^n$  e  $\mathbf{y} \in \mathbb{R}^n$ , la scatola (aperta)  $(\mathbf{x}, \mathbf{y})$  è

$$(\mathbf{x}, \mathbf{y}) = \{\mathbf{v} \in \mathbb{R}^n : \underline{v}_1 < v_1 < \bar{v}_1 \wedge \underline{v}_2 < v_2 < \bar{v}_2 \wedge \dots \wedge \underline{v}_n < v_n < \bar{v}_n\} \quad (25)$$

dove, per ogni  $1 \leq i \leq n$ ,

$$\underline{v}_i = \min\{x_i, y_i\} \quad \text{e} \quad \bar{v}_i = \max\{x_i, y_i\} \quad (26)$$

mentre la scatola chiusa  $[\mathbf{x}, \mathbf{y}]$  è

$$[\mathbf{x}, \mathbf{y}] = \{\mathbf{v} \in \mathbb{R}^n : \underline{v}_1 \leq v_1 \leq \bar{v}_1 \wedge \underline{v}_2 \leq v_2 \leq \bar{v}_2 \wedge \dots \wedge \underline{v}_n \leq v_n \leq \bar{v}_n\} \quad (27)$$

Si noti che una scatola aperta può degenerare nell'insieme vuoto se viene usato lo stesso vettore per identificare i due estremi della scatola. Al contrario, in questa situazione, una scatola chiusa degenera in un insieme singoletto.

**Palle.** Dato un vettore  $\mathbf{x} \in \mathbb{R}^n$  e uno scalare  $r \in \mathbb{R}_+$ , la palla (aperta)  $\mathcal{B}_r(\mathbf{x})$ , centrata nel punto identificato da  $\mathbf{x}$  e di raggio  $r$ , è

$$\mathcal{B}_r(\mathbf{x}) = \{\mathbf{v} \in \mathbb{R}^n : \|\mathbf{v} - \mathbf{x}\| < r\} \quad (28)$$

mentre la palla chiusa  $\mathcal{B}_r[\mathbf{x}]$  è

$$\mathcal{B}_r[\mathbf{x}] = \{\mathbf{v} \in \mathbb{R}^n : \|\mathbf{v} - \mathbf{x}\| \leq r\} \quad (29)$$

Si noti che una palla non può degenerare nell'insieme vuoto perché  $r > 0$ .

**Insiemi limitati, aperti, chiusi, compatti e convessi.** Le definizioni precedenti consentono di generalizzare a  $\mathbb{R}^n$  alcune nozioni comunemente usate nei casi monodimensionali, bidimensionali e tridimensionali:

- $L \subseteq \mathbb{R}^n$  si dice *limitato* se esistono  $r \in \mathbb{R}_+$  e  $\mathbf{x} \in \mathbb{R}^n$  tali che  $L \subseteq \mathcal{B}_r(\mathbf{x})$ ;
- $A \subseteq \mathbb{R}^n$  si dice *aperto* se, per ogni  $\mathbf{x} \in A$ , esiste  $r \in \mathbb{R}_+$  tale che  $\mathcal{B}_r(\mathbf{x}) \subseteq A$ ;
- $C \subseteq \mathbb{R}^n$  si dice *chiuso* se esiste un  $A \subseteq \mathbb{R}^n$  aperto tale che  $C = \mathbb{R}^n \setminus A$ ;
- $K \subseteq \mathbb{R}^n$  si dice *compatto* se è chiuso e limitato; e
- $V \subseteq \mathbb{R}^n$  si dice *convesso* se per ogni coppia  $\mathbf{x} \in V$  e  $\mathbf{y} \in V$  vale  $\overline{[\mathbf{x}, \mathbf{y}]} \subseteq V$ .

**Esempio 2.** Si consideri il sottoinsieme  $S$  di  $\mathbb{R}^2$  definito come segue:

$$S = \{(x, y) \in \mathbb{R}^2 : 2x^2 + 2y^2 + 2xy - 1 \leq 0\} \quad (30)$$

Visto che la *forma quadrica* non è in forma *canonica* (parabola, ellisse, iperbole, coppia di rette), cerchiamo una matrice di rotazione  $M$  tale che

$$2x^2 + 2y^2 + 2xy = \mathbf{x}^\top Q \mathbf{x} \quad \mathbf{x} = (x, y) \quad Q = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \quad (31)$$

risulti *diagonale* quando  $\mathbf{x} = M\mathbf{x}'$ . Gli *autovalori* di  $Q$  si ottengono risolvendo

$$\det(Q - \lambda I_2) = \det \begin{pmatrix} 2 - \lambda & 1 \\ 1 & 2 - \lambda \end{pmatrix} = 0 \quad (32)$$

e valgono  $\lambda_1 = 1$  e  $\lambda_2 = 3$ . Due autovettori corrispondenti ai due autovalori sono

$$\mathbf{v}_1 = (1, -1) \quad \mathbf{v}_2 = (1, 1) \quad (33)$$

da cui si può ottenere la matrice diagonale  $D$  cercata

$$D = M^\top Q M \quad M = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \quad D = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \quad (34)$$

Quindi,  $S$  è l'insieme dei vettori racchiuso da una ellisse, inclinata di  $-\frac{\pi}{4}$  radianti, con semiassi di lunghezza 1 e  $\frac{1}{\sqrt{3}}$ .

**Esempio 3.** Si consideri il sottoinsieme  $S$  di  $\mathbb{R}^2$  definito come segue:

$$S = \{(x, y) \in \mathbb{R}^2 : 2x^2 + 2xy - 14x + 2y^2 - 10y + 25 \leq 0\} \quad (35)$$

Per eliminare i termini in  $x$  e  $y$  dal polinomio che identifica  $S$  è possibile procedere al *completamento dei quadrati* e ottenere:

$$S = \{(x, y) \in \mathbb{R}^2 : 2(x - 3)^2 + 2(y - 1)^2 + 2(x - 3)(y - 1) - 1 \leq 0\} \quad (36)$$

Quindi, sfruttando il risultato dell'esempio precedente, è possibile dire che  $S$  è l'insieme dei vettori racchiuso da una ellisse centrata in  $(3, 1)$ , inclinata di  $-\frac{\pi}{4}$  radianti, con semiassi di lunghezza 1 e  $\frac{1}{\sqrt{3}}$ .

### 3 Funzioni Reali di Più Variabili Reali

Dati  $n \in \mathbb{N}_+$  e  $m \in \mathbb{N}_+$ , si considerino un *dominio*  $D \subseteq \mathbb{R}^n$  e un *codomnio*  $C \subseteq \mathbb{R}^m$ . La funzione

$$\mathbf{f} : D \rightarrow C \quad (37)$$

è una *funzione vettoriale* a  $m$  componenti reali di  $n$  variabili reali. Le funzioni di questo tipo possono essere studiate come  $m$  funzioni reali di  $n$  variabili reali

$$\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})) \quad (38)$$

e quindi, normalmente, lo studio delle funzioni vettoriali viene ridotto allo studio delle funzioni reali di più variabili reali.

Si consideri una funzione reale  $f$  di più variabili reali definita almeno in un aperto  $D \subseteq \mathbb{R}^n$ . Si dice che

$$\lim_{\mathbf{x} \rightarrow \mathbf{w}} f(\mathbf{x}) = l \quad (39)$$

se e soltanto se

$$\forall \epsilon \in \mathbb{R}_+ \exists \delta \in \mathbb{R}_+ : \forall \mathbf{x} \in D \quad \|\mathbf{x} - \mathbf{w}\| < \delta \implies |f(\mathbf{x}) - l| < \epsilon \quad (40)$$

La definizione precedente permette di chiarire quando una funzione reale possa essere definita *continua* in  $\mathbf{w} \in D$

$$\lim_{\mathbf{x} \rightarrow \mathbf{w}} f(\mathbf{x}) = f(\mathbf{w}) \quad (41)$$

**Proposizione 4.** *Sia  $f$  una funzione reale di  $n \in \mathbb{N}_+$  variabili reali definita almeno in un aperto  $S \subseteq \mathbb{R}^n$ , se la funzione  $f$  è continua in tutto  $S$ , allora per ogni compatto  $D \subset S$  la funzione ammette almeno un minimo globale in  $D$  e almeno un massimo globale in  $D$ .*

Sia  $L \subseteq \mathbb{R}^n$  un insieme limitato e sia  $f$  una funzione reale definita almeno nel dominio  $D = \mathbb{R}^n \setminus L$ . Si dice che

$$\lim_{\mathbf{x} \rightarrow \infty} f(\mathbf{x}) = l \quad (42)$$

se e soltanto se

$$\forall \epsilon \in \mathbb{R}_+ \exists m \in \mathbb{R}_+ : \forall \mathbf{x} \in D \quad \|\mathbf{x}\| > m \implies |f(\mathbf{x}) - l| < \epsilon \quad (43)$$

La definizione precedente permette di studiare il comportamento asintotico delle funzioni reali di più variabili reali.

Si consideri una funzione reale  $f$  definita almeno in un aperto  $D \subseteq \mathbb{R}^n$  e si considerino  $\mathbf{x} \in D$  e un versore  $\mathbf{v} \in \mathbb{R}^n$ . Se esiste finito

$$\partial_{\mathbf{v}} f(\mathbf{x}) \triangleq \lim_{h \rightarrow 0} \frac{f(\mathbf{x} + h\mathbf{v}) - f(\mathbf{x})}{h} \quad (44)$$

allora viene chiamato *derivata direzionale* della funzione  $f$  nella direzione  $\mathbf{v}$ .

Si consideri una funzione reale  $f$  definita almeno in un aperto  $D \subseteq \mathbb{R}^n$  e si considerino  $\mathbf{x} \in D$  e un versore del tipo

$$\mathbf{v} = (0, 0, \dots, 1, 0, 0, \dots, 0) \quad (45)$$

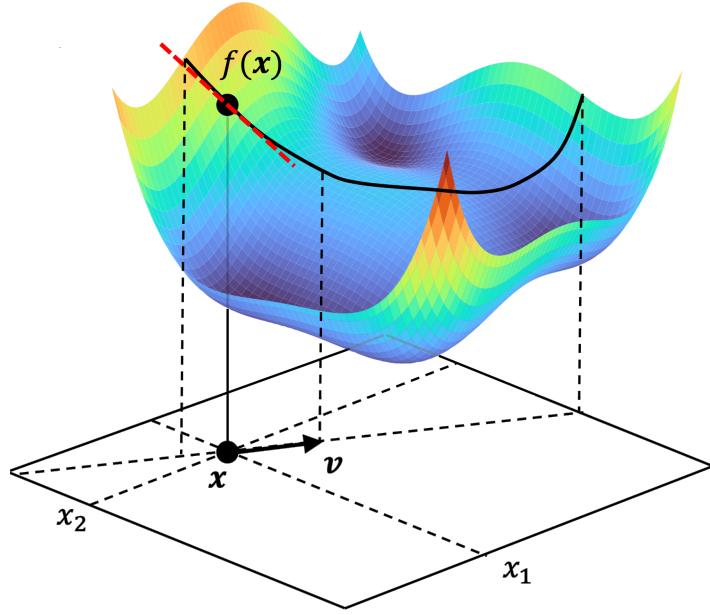


Figura 3: Derivata di  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  in  $\mathbf{x} \in \mathbb{R}^2$  nella direzione  $\mathbf{v} \in \mathbb{R}^2$

con il valore 1 in posizione  $1 \leq i \leq n$ . Se esiste, la derivata nella direzione  $\mathbf{v}$  viene chiamata *derivata parziale* in  $x_i$  e viene indicata con

$$\frac{\partial f}{\partial x_i}(\mathbf{x}) \quad \text{oppure} \quad \partial_i f(\mathbf{x}) \quad (46)$$

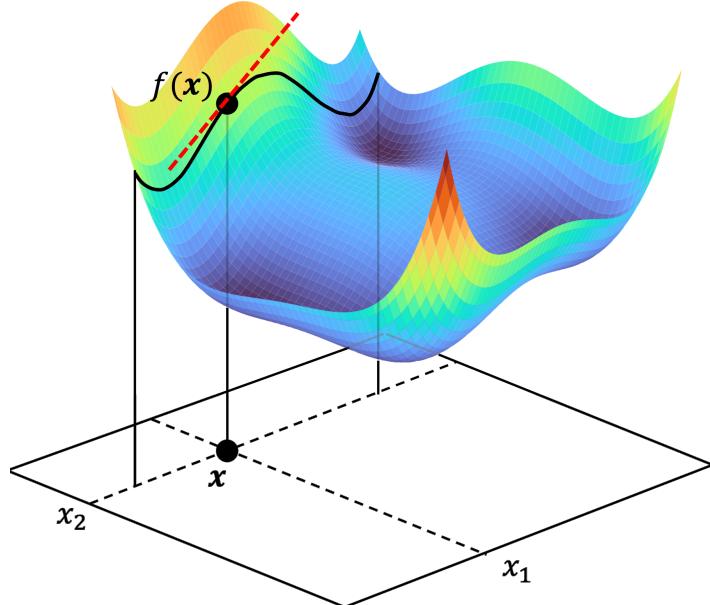


Figura 4: Derivata parziale di  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  in  $\mathbf{x} \in \mathbb{R}^2$  in  $x_1$

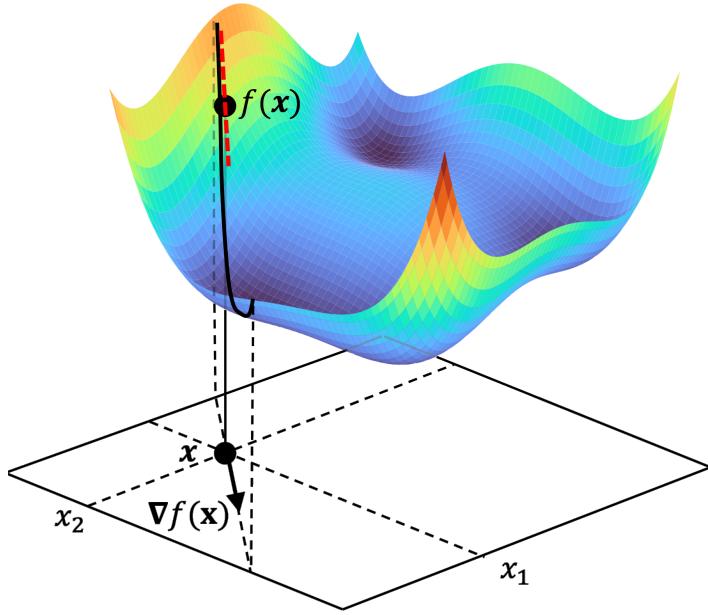


Figura 5: Gradiente di  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  in  $\mathbf{x} \in \mathbb{R}^2$

Si noti che il calcolo delle derivate parziali può essere ridotto al calcolo delle *derivate ordinarie*. Infatti, siccome il limite del *rapporto incrementale* coinvolge una sola variabile, durante il calcolo di una derivata parziale è possibile trattare le altre variabili come se fossero delle costanti.

Si noti che, se esiste, il vettore formato dalle  $n \in \mathbb{N}_+$  derivate parziali di una funzione reale di  $n$  variabili reali viene detto *gradiente* della funzione e si indica con

$$\nabla f(\mathbf{x}) \triangleq \left( \frac{\partial f}{\partial x_1}(\mathbf{x}), \frac{\partial f}{\partial x_2}(\mathbf{x}), \dots, \frac{\partial f}{\partial x_n}(\mathbf{x}) \right) \quad (47)$$

**Esempio 4.** Si consideri

$$f(x, y, z) = 3x^2 + 5xy - 7z^3 \quad (48)$$

allora

$$\nabla f(x, y, z) = (6x + 5y, 5x, -21z^2) \quad (49)$$

Se le derivate parziali di una funzione possono essere ulteriormente derivate, allora si parla di *derivate (parziali) di ordine superiore (al primo)* quando le derivate vengono calcolate rispetto ad una sola variabile, oppure di *derivate (parziali) miste* quando le derivate vengono calcolate rispetto a più variabili. In generale, l'*ordine* di una derivata parziale è pari al numero di derivate calcolate, indipendentemente dalle variabili utilizzate per il calcolo.

Sia  $D \subseteq \mathbb{R}^n$  un aperto e sia  $k \in \mathbb{N}_+$ , una funzione  $f : S \rightarrow \mathbb{R}$ , con  $D \subseteq S$ , si dice di classe  $C^k(D)$  se è derivabile almeno  $k$  volte su  $D$  e se le sue derivate sono continue su  $D$ . In più, una funzione si dice di classe  $C^0(D)$  se è continua su  $D$  e si dice di classe  $C^\infty(D)$  se è derivabile un numero arbitrario di volte su  $D$  e le sue derivate sono tutte continue su  $D$ .

**Proposizione 5.** Se una funzione reale di  $n \in \mathbb{N}_+$  variabili reali è di classe  $C^k(D)$ , con  $k \in \mathbb{N}$  e  $D \subseteq \mathbb{R}^n$  aperto, allora le sue derivate fino all'ordine  $k$  incluso sono indipendenti dall'ordine in cui vengono calcolate. Se una funzione è di classe  $C^\infty(D)$ , con  $D \subseteq \mathbb{R}^n$  aperto, allora le sue derivate, di qualsiasi ordine, sono indipendenti dall'ordine in cui vengono calcolate.

La seguente proposizione mette in relazione la possibilità di calcolare le derivate parziali con la proprietà di continuità delle funzioni.

**Proposizione 6.** Se una funzione reale di  $n \in \mathbb{N}_+$  variabili reali è di classe  $C^k(D)$ , con  $k \in \mathbb{N}$  e  $D \subseteq \mathbb{R}^n$  aperto, allora la funzione è di classe  $C^{k-1}(D)$ . Se una funzione è di classe  $C^\infty(D)$ , con  $D \subseteq \mathbb{R}^n$  aperto, allora per ogni  $k \in \mathbb{N}_+$  è di classe  $C^k(D)$ .

**Esempio 5.** Si consideri

$$f(x, y, z) = 3x^2 + 5xy - 7z^3 \quad (50)$$

allora

$$\frac{\partial^2 f}{\partial x \partial y}(x, y, z) = \frac{\partial^2 f}{\partial y \partial x}(x, y, z) = 5 \quad (51)$$

che conferma come l'ordine in cui vengono calcolate le derivate non sia rilevante visto che, in questo caso, tutte le derivate di qualsiasi ordine esistono e sono continue.

**Proposizione 7.** Sia  $f$  una funzione reale di  $n \in \mathbb{N}_+$  variabili reali definita almeno in un aperto  $D \subseteq \mathbb{R}^n$ , se esistono tutte le derivate parziali di  $f$  in  $\mathbf{x} \in D$ , allora

$$\partial_{\mathbf{v}} f(\mathbf{x}) = \mathbf{v} \cdot \nabla f(\mathbf{x}) \quad (52)$$

per ogni versore  $\mathbf{v} \in \mathbb{R}^n$ .

Si noti che la diseguaglianza di Cauchy-Schwarz applicata ad una funzione  $f$  per cui valgono le ipotesi della proposizione precedente permette di dire che

$$-||\nabla f(\mathbf{x})|| \leq \partial_{\mathbf{v}} f(\mathbf{x}) \leq ||\nabla f(\mathbf{x})|| \quad (53)$$

qualsiasi sia il versore  $\mathbf{v} \in \mathbb{R}^n$  e, in più, permette di dire che le uguaglianze valgono se e soltanto se esiste un  $\lambda \in \mathbb{R}$  tale che

$$\mathbf{v} = \lambda \nabla f(\mathbf{x}) \quad (54)$$

Quindi, si evince che, fissato un vettore  $\mathbf{x} \in \mathbb{R}^n$  nel dominio della funzione,  $\nabla f(\mathbf{x})$  indica la direzione di massima velocità di crescita della funzione nel punto identificato da  $\mathbf{x}$  e  $-\nabla f(\mathbf{x})$  indica la direzione di massima velocità di decrescita della funzione nel punto identificato da  $\mathbf{x}$ . Se vale  $\nabla f(\mathbf{x}) = \mathbf{0}$ , allora la funzione si dice *stazionaria* nel punto identificato da  $\mathbf{x}$ .

**Proposizione 8.** Sia  $f$  una funzione reale di  $n \in \mathbb{N}_+$  variabili reali definita almeno in un aperto  $D \subseteq \mathbb{R}^n$ , se il gradiente di  $f$  è definito su tutto  $D$ , allora  $f$  può ammettere massimi locali e minimi locali in  $D$  solo nei punti in cui è stazionaria.

**Esempio 6.** Si consideri la funzione  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  definita da

$$f(x, y) = -2x^2 - 3y^2 + 4xy + 10 \quad (55)$$

Il gradiente della funzione vale

$$\nabla f(x, y) = (4y - 4x, 4x - 6y) \quad (56)$$

e quindi la funzione ammette un unico punto stazionario  $(0, 0)$  in cui vale  $f(0, 0) = 10$ . Siccome è possibile scrivere la funzione come

$$f(x, y) = -2(x - y)^2 - y^2 + 10 \quad (57)$$

si evince che l'unico punto stazionario è un massimo globale.

**Esempio 7.** Si consideri la funzione  $f : D \rightarrow \mathbb{R}$  definita da

$$f(x, y) = 2x^2 + 5y^2 + 4 \quad (58)$$

sul dominio  $D = \mathcal{B}_2[1, 0]$ . Il gradiente della funzione vale

$$\nabla f(x, y) = (4x, 10y) \quad (59)$$

e quindi la funzione ammette un unico punto stazionario  $(0, 0)$  in cui vale  $f(0, 0) = 4$ . Questo punto stazionario non può essere un massimo globale sul dominio  $D$  perché, ad esempio,  $f(1, 0) = 6$ . Quindi, i massimi globali possono solo trovarsi sul *bordo* (o *frontiera*) del dominio, che è l'insieme

$$\partial D = \{(x, y) : (x - 1)^2 + y^2 = 4\} \quad (60)$$

Sfruttando il fatto che in  $\partial D$  vale  $y^2 = 4 - (x - 1)^2$  è possibile cercare i massimi globali di  $f$  cercando i massimi globali di  $g : \mathbb{R} \rightarrow \mathbb{R}$

$$g(x) = 2x^2 + 5[4 - (x - 1)^2] + 4 = -3x^2 + 10x + 19 \quad (61)$$

Siccome  $g$  è una parabola con concavità verso il basso, è semplice calcolarne il valore massimo. Infatti, siccome  $g$  ammette massimo globale in  $x = \frac{5}{3}$ , il massimo globale di  $f$  si trova nei due punti  $\left(\frac{5}{3}, \pm \frac{4\sqrt{2}}{3}\right)$ .

Le precedenti considerazioni sulle relazioni tra minimi, massimi e punti stazionari permettono di descrivere un algoritmo per la ricerca di un minimo locale di una funzione reale di  $n \in \mathbb{N}_+$  variabili reali definita su tutto  $\mathbb{R}^n$  per cui sia definito il gradiente su tutto  $\mathbb{R}^n$ . L'algoritmo, detto di *discesa del gradiente*, è descritto nell'Algoritmo 1.

L'algoritmo di discesa del gradiente richiede tre argomenti: la funzione  $f$  di cui si cerca un minimo locale, una prima approssimazione  $\mathbf{x} \in \mathbb{R}^n$  di uno dei minimi locali di  $f$  e il *passo di discesa*  $\alpha \in \mathbb{R}_+$ . L'algoritmo parte dall'approssimazione  $\mathbf{x}$  fornita e cerca un minimo locale in  $\mathbb{R}^n$  spostandosi, ad ogni iterazione, nella direzione identificata dal gradiente della funzione cambiato di segno. Siccome la direzione identificata dal gradiente cambiato di segno corrisponde alla direzione di massima velocità di decrescita della funzione, la ricerca dei minimi locali procede sempre nella direzione più promettente. L'algoritmo termina quando  $\mathbf{x}$  identifica un punto stazionario della funzione, che corrisponde quindi ad un minimo locale per la Proposizione 8. Si noti che il passo di discesa  $\alpha$  corrisponde alla velocità utilizzata dall'algoritmo per percorrere la discesa nella direzione identificata dal gradiente cambiato di segno.

---

**Algoritmo 1** Algoritmo di discesa del gradiente

---

```

function GRADIENT DESCENT( $f$ ,  $\mathbf{x}$ ,  $\alpha$ )
     $\mathbf{g} \leftarrow \nabla f(\mathbf{x})$                                  $\triangleright$  inizializza  $\mathbf{g}$  a  $\nabla f(\mathbf{x})$ 
    while  $\mathbf{g} \neq \mathbf{0}$  do           $\triangleright$  entra se  $\mathbf{x}$  non identifica un punto stazionario di  $f$ 
         $\mathbf{v} \leftarrow -\frac{\mathbf{g}}{\|\mathbf{g}\|}$            $\triangleright$   $\mathbf{v}$  indica la direzione di  $-\nabla f(\mathbf{x})$ 
         $\mathbf{x} \leftarrow \mathbf{x} + \alpha \mathbf{v}$            $\triangleright$  muovi  $\mathbf{x}$  nella direzione di  $-\nabla f(\mathbf{x})$ 
         $\mathbf{g} \leftarrow \nabla f(\mathbf{x})$            $\triangleright$  aggiorna  $\mathbf{g}$  con il nuovo valore di  $\nabla f(\mathbf{x})$ 
    end while
    return  $\mathbf{x}$            $\triangleright$   $\mathbf{x}$  identifica un punto stazionario di  $f$ 
end function

```

---

## 4 Integrali Definiti di Funzioni Reali in Più Variabili Reali

Sia  $D \subset \mathbb{R}^n$  un insieme compatto, se esiste finito, l'*integrale definito* di una funzione  $f : D \rightarrow \mathbb{R}$  si indica con

$$\int_D f(\mathbf{x}) d^n \mathbf{x} \quad (62)$$

**Proposizione 9.** *Dati  $\mathbf{v} \in \mathbb{R}^n$  e  $\mathbf{w} \in \mathbb{R}^n$ , sia  $D = [\mathbf{v}, \mathbf{w}]$  una scatola chiusa e sia  $f : S \rightarrow \mathbb{R}$  continua su tutto  $S$ , con  $S$  aperto e  $D \subset S$ . Allora, esiste l'integrale definito di  $f$  su  $D$  e vale*

$$\int_D f(\mathbf{x}) d^n \mathbf{x} = \int_{\underline{x}_1}^{\bar{x}_1} \int_{\underline{x}_2}^{\bar{x}_2} \cdots \int_{\underline{x}_n}^{\bar{x}_n} f(\mathbf{x}) dx_n dx_{(n-1)} \cdots dx_1 \quad (63)$$

where, per ogni  $1 \leq i \leq n$ ,  $\underline{x}_i = \min\{v_i, w_i\}$ ,  $\bar{x}_i = \max\{v_i, w_i\}$  e gli integrali definiti possono essere calcolati in qualsiasi ordine.

Quindi, la proposizione precedente permette di ridurre il calcolo di un integrale definito su una scatola chiusa al calcolo di  $n$  integrali definiti.

**Esempio 8.** Si considerino

$$\mathbf{v} = (2, 4) \quad \text{e} \quad \mathbf{w} = (3, 5) \quad (64)$$

con  $D = [\mathbf{v}, \mathbf{w}] \subseteq \mathbb{R}^2$ , e sia  $f(x, y) = x^3 + y^2$ . Si può calcolare l'integrale definito di  $f$  su  $D$  integrando prima nella variabile  $y$  e poi nella variabile  $x$

$$\int_D (x^3 + y^2) dx dy = \int_2^3 \int_4^5 (x^3 + y^2) dy dx = \int_2^3 \left( x^3 + \frac{61}{3} \right) dx = \frac{439}{12} \quad (65)$$

Si noti che lo stesso risultato sarebbe stato ottenuto integrando prima nella variabile  $x$  e poi nella variabile  $y$ , dopo un opportuno scambio degli estremi di integrazione.

**Proposizione 10.** *Sia  $D \subset \mathbb{R}^n$  un insieme compatto e sia  $f : S \rightarrow \mathbb{R}$ , con  $S$  aperto e  $D \subset S$ , una funzione continua. Sia  $g : S \rightarrow \mathbb{R}$  una funzione tale che  $g$  sia uguale a  $f$  su tutto  $D$  tranne che per un insieme al più numerabile di punti in cui  $g$  è discontinua. Allora entrambi i seguenti integrali esistono e sono uguali*

$$\int_D g(\mathbf{x}) d^n \mathbf{x} = \int_D f(\mathbf{x}) d^n \mathbf{x} \quad (66)$$

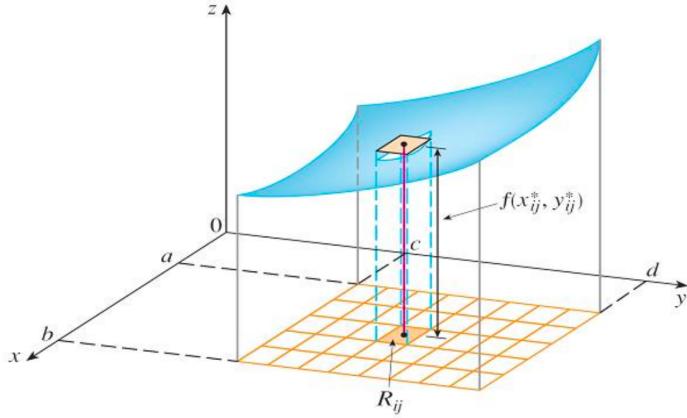


Figura 6: Il significato geometrico di un integrale definito in due variabili

Dato  $n \in \mathbb{N}_+$ , sia  $L \subseteq \mathbb{R}^n$  un insieme limitato e sia  $D = \mathbb{R}^n \setminus L$ , se  $f : D \rightarrow \mathbb{R}$  ed esiste finito

$$\int_D f(\mathbf{x}) d^n \mathbf{x} \triangleq \lim_{\mathbf{v} \rightarrow \infty} \int_{[-\mathbf{v}, \mathbf{v}] \cap D} f(\mathbf{x}) d^n \mathbf{x} \quad (67)$$

allora il limite si dice *integrale improprio* di  $f$  su  $\mathbb{R}^n$ .

## 5 Convoluzione tra Funzioni

Dato  $n \in \mathbb{N}_+$ , siano  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  e  $g : \mathbb{R}^n \rightarrow \mathbb{R}$ , se esiste finito si definisce *convoluzione* (o *prodotto di convoluzione*) tra  $f$  e  $g$

$$(f \star g)(\mathbf{x}) \triangleq \int_{\mathbb{R}^n} f(\mathbf{v})g(\mathbf{x} - \mathbf{v}) d^n \mathbf{v} = \int_{\mathbb{R}^n} f(\mathbf{x} - \mathbf{v})g(\mathbf{v}) d^n \mathbf{v} \quad (68)$$

In Figura 7 sono mostrati i passi necessari al calcolo della convoluzione tra due generiche funzioni  $x : \mathbb{R} \rightarrow \mathbb{R}$  e  $h : \mathbb{R} \rightarrow \mathbb{R}$  usando  $\tau$  come variabile di integrazione e  $t$  come variabile associata al risultato della convoluzione. In particolare, il calcolo della convoluzione tra  $x$  e  $h$  prevede di calcolare la funzione simmetrica di  $h$  rispetto all'asse delle ordinate

$$\bar{h}(\tau) = h(-\tau) \quad (69)$$

e di traslare la funzione ottenuta  $\bar{h}$  di  $t$  lungo l'asse delle ascisse per calcolare l'integrale del prodotto

$$x(\tau)\bar{h}(\tau - t) = x(\tau)h(t - \tau) \quad (70)$$

Si noti che la convoluzione ha le seguenti proprietà, nell'ipotesi che tutti gli integrali esistano finiti e che  $a \in \mathbb{R}$ ,

$$f \star g = g \star f \quad (71)$$

$$(af) \star g = f \star (ag) = a(f \star g) \quad (72)$$

$$f \star (g \star h) = (f \star g) \star h = f \star g \star h \quad (73)$$

$$f \star (g + h) = f \star g + f \star h \quad (74)$$

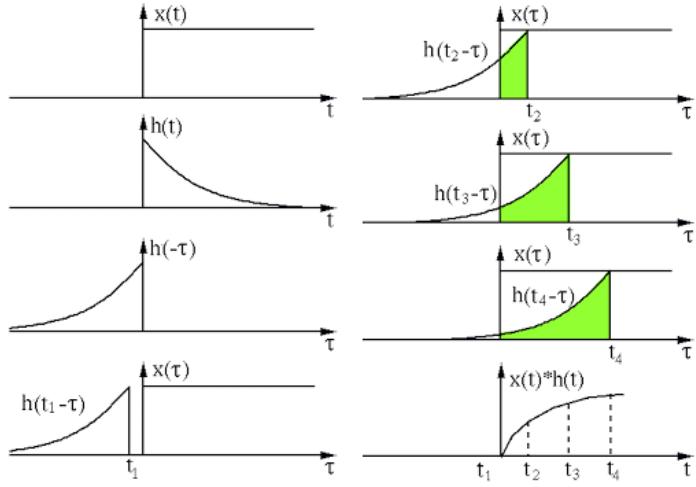


Figura 7: Passi necessari al calcolo della convoluzione tra  $x : \mathbb{R} \rightarrow \mathbb{R}$  e  $h : \mathbb{R} \rightarrow \mathbb{R}$

In più, si noti che la convoluzione ha la seguente proprietà

$$\frac{\partial(f * g)}{\partial x} = \frac{\partial f}{\partial x} * g = f * \frac{\partial g}{\partial x} \quad (75)$$

Infine, detto  $T_{\mathbf{r}}$  l'operatore di traslazione di  $\mathbf{r} \in \mathbb{R}^n$ , che opera su funzioni reali di più variabili reali nel seguente modo

$$(T_{\mathbf{r}}f)(\mathbf{x}) = f(\mathbf{x} - \mathbf{r}) \quad (76)$$

vale la seguente proprietà della convoluzione

$$T_{\mathbf{r}}(f * g) = f * T_{\mathbf{r}}g = T_{\mathbf{r}}f * g \quad (77)$$

La definizione di convoluzione mostra come sia possibile usare la convoluzione per descrivere l'effetto dell'applicazione di una trasformazione descritta dalla funzione  $g$  sulla funzione  $f$ . Però, le proprietà della convoluzione ricordate in precedenza limitano le possibili trasformazioni che è possibile descrivere. In particolare, la convoluzione tra una funzione  $f$  e una funzione  $g$  può descrivere trasformazioni di  $f$  tramite  $g$  nell'ipotesi che la trasformazione sia lineare e che la trasformazione trasli di  $\mathbf{r} \in \mathbb{R}^n$  il risultato della sua applicazione se la funzione  $f$  viene traslata di  $\mathbf{r}$ . Quando la convoluzione con una funzione  $g$  viene usata per descrivere trasformazioni con queste caratteristiche, la trasformazione prende il nome di *filtro* e la funzione  $g$  viene detta *risposta impulsiva (del filtro)*.

L'operazione di convoluzione è, di solito, particolarmente rilevante se almeno una delle funzioni è una *funzione a supporto compatto*. Dato  $n \in \mathbb{N}_+$ , una funzione a supporto compatto è una funzione definita su tutto  $\mathbb{R}^n$  che assume valore 0 al di fuori di un insieme compatto detto, appunto, *supporto (della funzione)*.

Un esempio di funzione a supporto compatto è la seguente funzione *rettangolo unitario* (o *porta unitaria*)

$$r(x) = \begin{cases} 1 & \text{se } |x| < \frac{1}{2} \\ \frac{1}{2} & \text{se } |x| = \frac{1}{2} \\ 0 & \text{altrimenti} \end{cases} \quad (78)$$

Si noti che la funzione rettangolo unitario può essere espressa mediante la *funzione gradino unitario* (o *funzione di Heaviside*)

$$h(x) = \begin{cases} 1 & \text{se } x > 0 \\ \frac{1}{2} & \text{se } x = 0 \\ 0 & \text{altrimenti} \end{cases} \quad (79)$$

infatti

$$r(x) = h\left(x + \frac{1}{2}\right) - h\left(\frac{1}{2} - x\right) \quad (80)$$

**Esempio 9.** Sia  $f : \mathbb{R} \rightarrow \mathbb{R}$  una funzione continua e sia  $r : \mathbb{R} \rightarrow \mathbb{R}$  la funzione rettangolo unitario, allora esiste finito

$$\bar{f}(x) = (f \star r)(x) = \int_{-\infty}^{\infty} f(v)r(x-v) dv = \int_{x-\frac{1}{2}}^{x+\frac{1}{2}} f(v) dv \quad (81)$$

Si noti che  $f \star r$  calcola, per ogni  $x$ , la media di  $f$  nell'intervallo chiuso  $[x - \frac{1}{2}, x + \frac{1}{2}]$ . Quindi, se si vuole calcolare la media di  $f$  in un intervallo ampio  $w \in \mathbb{R}_+$ , allora è sufficiente calcolare

$$\bar{f}_w(x) = \frac{1}{w} \int_{-\infty}^{\infty} f(v)r\left(\frac{x-v}{w}\right) dv = \frac{1}{w} \int_{x-\frac{w}{2}}^{x+\frac{w}{2}} f(v) dv \quad (82)$$

che prende il nome di *media mobile* di  $f$  centrata in  $x$  con ampiezza  $w$ .

Si noti che se  $g : \mathbb{R} \rightarrow \mathbb{R}$  è una funzione pari a supporto compatto che abbia come supporto l'intervallo chiuso  $S = [-s, s]$ , con  $s \in \mathbb{R}_+$ , e  $f : \mathbb{R} \rightarrow \mathbb{R}$  è una funzione continua, allora  $f \star g$  rappresenta, per ogni  $x \in \mathbb{R}$ , una sorta di somma dei valori differenziali  $f(x) dx$  nell'intervallo chiuso  $[x-s, x+s]$  pesata con i corrispondenti valori di  $g$ . Infatti, in queste condizioni,  $f \star g$  esiste e vale

$$(f \star g)(x) = \int_{x-s}^{x+s} f(v)g(v-x) dv \quad (83)$$

In generale, quindi, fissato  $n \in \mathbb{N}_+$ , la convoluzione permette di calcolare alcune proprietà locali di una funzione  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  in un intorno di un vettore fissato come riferimento mobile mediante una funzione  $g : \mathbb{R}^n \rightarrow \mathbb{R}$  a supporto compatto che prende il nome di *nucleo* (o *kernel*) di convoluzione. Si noti, però, che se la funzione  $g$  non è pari, allora bisogna tenere presente che l'operazione di convoluzione lavora sulla funzione  $g$  dopo aver cambiato di segno la variabile.

**Esempio 10.** Detta  $h : \mathbb{R} \rightarrow \mathbb{R}$  la funzione di Heaviside, se  $f : \mathbb{R} \rightarrow \mathbb{R}$  è la seguente funzione

$$f(x) = 3e^{-2x}h(x) \quad (84)$$

allora

$$(f \star h)(x) = \int_{-\infty}^{+\infty} 3e^{-2\alpha}h(\alpha)h(x-\alpha)d\alpha = h(x) \int_0^x 3e^{-2\alpha}d\alpha \quad (85)$$

e quindi

$$(f \star h)(x) = -\frac{3}{2}e^{-2x} \Big|_0^x h(x) = \frac{3}{2}h(x)(1 - e^{-2x}) \quad (86)$$

**Esempio 11.** Detta  $h : \mathbb{R} \rightarrow \mathbb{R}$  la funzione di Heaviside, siano  $f : \mathbb{R} \rightarrow \mathbb{R}$  e  $g : \mathbb{R} \rightarrow \mathbb{R}$  le seguenti funzioni

$$f(x) = e^{-x}h(x) \quad g(x) = h(x)\sin x \quad (87)$$

allora

$$(f \star h)(x) = \int_{-\infty}^{+\infty} e^{-\alpha}h(\alpha)h(x-\alpha)\sin(x-\alpha)d\alpha = h(x) \int_0^x e^{-\alpha}\sin(x-\alpha)d\alpha \quad (88)$$

Quindi, il calcolo di  $f \star g$  si riduce al calcolo di

$$\int e^{-\alpha}\sin(x-\alpha)d\alpha \quad (89)$$

che può essere ottenuto mediante *integrazione per parti*, infatti

$$\int e^{-\alpha}\sin(x-\alpha)d\alpha = -e^{-\alpha}\sin(x-\alpha) - \int e^{-\alpha}\cos(x-\alpha)d\alpha + C_1 \quad (90)$$

ma, in modo simile, vale che

$$\int e^{-\alpha}\cos(x-\alpha)d\alpha = -e^{-\alpha}\cos(x-\alpha) + \int e^{-\alpha}\sin(x-\alpha)d\alpha + C_2 \quad (91)$$

e quindi

$$\int e^{-\alpha}\sin(x-\alpha)d\alpha = \frac{1}{2}e^{-\alpha}[\cos(x-\alpha) - \sin(x-\alpha)] + C \quad (92)$$

da cui si ottiene che

$$(f \star g)(x) = \frac{1}{2}e^{-\alpha}[\cos(x-\alpha) - \sin(x-\alpha)] \Big|_0^x h(x) \quad (93)$$

che equivale a dire che

$$(f \star g)(x) = \frac{1}{2}e^{-\alpha}[\cos(x-\alpha) - \sin(x-\alpha)] \Big|_0^x h(x) = \frac{1}{2}h(x)(e^{-x} + \sin x - \cos x) \quad (94)$$

La definizione di convoluzione può essere estesa a successioni di valori reali definite con indici in  $\mathbb{Z}$ . Date due successioni  $(f_i)_{i \in \mathbb{Z}}$  e  $(g_i)_{i \in \mathbb{Z}}$ , è possibile definire, se esiste, la loro convoluzione  $(f \star g)_{i \in \mathbb{Z}}$  come

$$(f \star g)_i = \sum_{k=-\infty}^{+\infty} f_k g_{i-k} = \sum_{k=-\infty}^{+\infty} f_{i-k} g_k \quad (95)$$

Esattamente come nel caso della convoluzione tra funzioni, se una delle due successioni, ad esempio  $g$ , è identicamente nulla per indici esterni ad un intervallo finito, allora la convoluzione tra successioni permette di calcolare delle proprietà locali di  $f$  mediante il nucleo  $g$ . In più, conviene notare che se entrambe le successioni sono identicamente nulle per indici fuori da due intervalli finiti, allora la convoluzione si

riduce al prodotto tra polinomi. Infine, si noti che in alcune situazioni il prodotto  $f_k g_{i-k}$  viene sostituito dal prodotto  $f_k g_{k-i}$ , di fatto cambiando la definizione di convoluzione tra successioni.

In modo simile a quanto fatto per la convoluzione tra funzioni, è possibile definire la *convoluzione discreta* andando a generalizzare quanto fatto per le successioni. Un caso particolarmente interessante è quello della convoluzione tra matrici. Date due matrici  $A^{(n \times m)}$  e  $B^{(s \times t)}$  è possibile definire la convoluzione  $A \star B$

$$(A \star B)_{r,c} = \sum_{i=1}^n \sum_{j=1}^m A_{i,j} B_{r-i,c-j} = \sum_{i=1}^s \sum_{j=1}^t A_{r-i,c-j} B_{i,j} \quad (96)$$

dove si assume che il valore di una matrice per indici non validi sia nullo. Si noti che in alcune situazioni il prodotto  $A_{i,j} B_{r-i,c-j}$  viene sostituito dal prodotto  $A_{i,j} B_{i-r,j-c}$ , di fatto cambiando la definizione di convoluzione tra matrici.

## 6 Distribuzioni

Come già accennato, la convoluzione è uno strumento particolarmente utile per descrivere particolari trasformazioni dette filtri e le relative risposte impulsive. Conviene però subito notare che un filtro molto semplice, detto *filtro identità*, non può essere descritto mediante la convoluzione.

Per semplicità di notazione consideriamo solo funzioni di una variabile reale. Il filtro identità è il filtro che, applicato a una funzione, produce esattamente la stessa funzione. Quindi, se esiste una funzione  $\delta : \mathbb{R} \rightarrow \mathbb{R}$  da usare come risposta impulsiva del filtro identità, allora

$$f(x) = (f \star \delta)(x) = \int_{-\infty}^{+\infty} f(v) \delta(x - v) \, dv \quad (97)$$

per qualsiasi funzione  $f$  per cui l'integrale esiste finito. Siccome, però, il risultato della convoluzione deve essere lo stesso per tutte le funzioni che abbiano lo stesso valore in  $x \in \mathbb{R}$ , allora si sta implicitamente dicendo che  $\delta(x) = 0$  per ogni  $x \neq 0$ . Infatti, siccome è possibile scegliere una qualsiasi funzione per cui la convoluzione esista, è possibile scegliere una funzione a supporto compatto che abbia l'intervallo chiuso  $[0, 0]$  come supporto. Però, se  $h : \mathbb{R} \rightarrow \mathbb{R}$  è una funzione identicamente nulla tranne che per un singolo valore della sua variabile, allora vale

$$\int_{-\infty}^{+\infty} h(v) \, dv = 0 \quad (98)$$

perché due funzioni continue hanno lo stesso integrale su un compatto se differiscono per un insieme al più numerabile di punti di discontinuità all'interno del compatto considerato. Quindi, la funzione  $\delta$  cercata non può esistere perché il risultato della convoluzione sarebbe comunque sempre nullo.

Per permettere di usare la convoluzione per descrivere il filtro identità, e tanti altri filtri interessanti, si immagina che la funzione  $\delta$  possa esistere e la si tratta

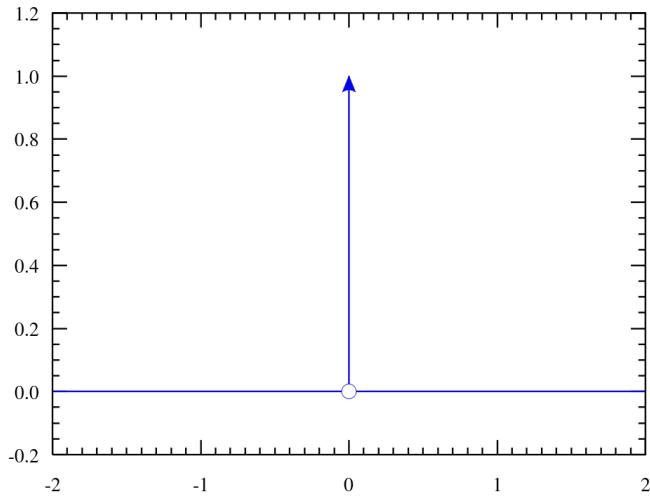


Figura 8: Rappresentazione della  $\delta$  di Dirac nel piano cartesiano

come una funzione speciale con le seguenti proprietà. Se  $x_0 \in \mathbb{R}$ ,  $a \in \mathbb{R} \setminus \{0\}$  e  $f : \mathbb{R} \rightarrow \mathbb{R}$  è una funzione di classe  $C^\infty(\mathbb{R})$ , allora valgono le seguenti proprietà di  $\delta$

$$\delta(x - x_0) = 0 \quad \text{se } x \neq x_0 \quad (99)$$

$$\delta(ax) = \frac{1}{|a|}\delta(x) \quad (100)$$

$$f(x)\delta(x - x_0) = f(x_0)\delta(x - x_0) \quad (101)$$

$$\int_{-\infty}^{+\infty} f(x)\delta(x - x_0) dx = f(x_0) \quad (102)$$

La funzione  $\delta$  caratterizzata in questo modo prende il nome di  $\delta$  di Dirac ed è un esempio di una classe di oggetti matematici che, in alcune situazioni, si comportano come funzioni e che quindi vengono chiamati *funzioni generalizzate* o *distribuzioni*.

**Esempio 12.** Valgono le seguenti relazioni, che sono casi particolari delle proprietà caratteristiche della  $\delta$  di Dirac

$$\delta(x) = 0 \quad \text{se } x \neq 0 \quad (103)$$

$$\delta(-x) = \delta(x) \quad (104)$$

Queste relazioni giustificano la rappresentazione grafica della  $\delta$  di Dirac nel piano cartesiano mostrata in Figura 6.

Oltre alla  $\delta$  di Dirac, una qualsiasi funzione  $f : \mathbb{R} \rightarrow \mathbb{R}$  continua, eventualmente, tranne che per un insieme al più numerabile di punti di discontinuità, può essere considerata una distribuzione. Data una distribuzione, è possibile moltiplicarla per uno scalare per ottenere una distribuzione, e, date due distribuzioni, è possibile sommarle per ottenere una distribuzione. Quindi, è possibile operare sulle distribuzioni

con le consuete regole di calcolo per somme, sottrazioni e moltiplicazioni, facendo però attenzione all'uso delle moltiplicazioni se la distribuzione è una  $\delta$  di Dirac.

Quando si parla di distribuzioni, il calcolo delle derivate viene esteso anche a funzioni che normalmente non sarebbero derivabili. Ad esempio, se  $h$  è la funzione di Heaviside, allora

$$h'(x) = \delta(x) \quad (105)$$

Si noti che, estendendo alle distribuzioni il concetto di derivata, rimangono comunque valide tutte le regole per il calcolo delle derivate delle funzioni.

**Esempio 13.** Detta  $h : \mathbb{R} \rightarrow \mathbb{R}$  la funzione di Heaviside, se

$$f(x) = 5 e^{-2x} h(x) \quad (106)$$

allora

$$f'(x) = 5 \delta(x) - 10 e^{-2x} h(x) \quad (107)$$

confermando come sia possibile calcolare la derivata nel senso delle distribuzioni anche di funzioni che non sono derivabili in senso ordinario. Quindi, si noti che vale

$$\int [5 \delta(x) - 10 e^{-2x} h(x)] dx = 5 e^{-2x} h(x) + C \quad (108)$$

confermando come sia possibile estendere le regole di calcolo delle primitive anche alle distribuzioni.

In modo simile a quanto fatto per le derivate, tenuto conto delle proprietà caratteristiche della  $\delta$  di Dirac, è possibile utilizzare le comuni regole di calcolo per gli integrali definiti, eventualmente verificando che gli integrali impropri esistano finiti.

**Esempio 14.** La  $\delta$  di Dirac, introdotta per descrivere il filtro identità, è anche utile a descrivere il *filtro di traslazione* (o *di ritardo*), che ha l'effetto di traslare la funzione a cui è applicato di una quantità nota. Se  $x_0 \in \mathbb{R}$  è la traslazione che viene introdotta dal filtro, allora la risposta impulsiva del filtro vale  $g(x) = \delta(x - x_0)$ , infatti, per una qualsiasi  $f : \mathbb{R} \rightarrow \mathbb{R}$  per cui l'integrale esiste, vale

$$(f \star g)(x) = \int_{-\infty}^{+\infty} f(v) \delta(x - x_0 - v) dv = \int_{-\infty}^{+\infty} f(v) \delta(v - (x - x_0)) dv = f(x - x_0)$$

e, quindi, l'effetto di  $\delta(x - x_0)$  è proprio quello di traslare  $f$  di  $x_0$ .

Si noti che, dato un filtro descritto da mediante una risposta impulsiva  $g : \mathbb{R} \rightarrow \mathbb{R}$ , se il filtro viene applicato alla  $\delta$  di Dirac si ottiene

$$(\delta \star g)(x) = \int_{-\infty}^{+\infty} \delta(v - x) g(v) dv = g(x) \quad (109)$$

che giustifica il nome di risposta impulsiva del filtro considerato per la funzione  $g$ .

Infine, si noti che le distribuzioni e, quindi anche la  $\delta$  di Dirac, possono essere estese a più variabili reali mantenendo le stesse proprietà viste nel caso di una singola variabile reale. In particolare, dato  $n \in \mathbb{N}_+$ , è possibile trattare la  $\delta$  di Dirac in  $\mathbb{R}^n$  immaginando che valga

$$\delta(\mathbf{x}) = \delta(x_1) \delta(x_2) \cdots \delta(x_n) \quad (110)$$

per qualsiasi  $\mathbf{x} \in \mathbb{R}^n$ .

**Esempio 15.** Sia  $g : \mathbb{R} \rightarrow \mathbb{R}$  una funzione sufficientemente regolare e sia  $f : \mathbb{R} \rightarrow \mathbb{R}$  una funzione derivabile su  $\mathbb{R}$  tale che, per  $a \in \mathbb{R}$ , valga

$$f'(x) + af(x) = g(x) \quad (111)$$

Si dice che la funzione  $f$  risolve l'*equazione differenziale ordinaria lineare non omogenea* (111). Per trovare  $f$ , supponiamo di disporre di una funzione  $t : \mathbb{R} \rightarrow \mathbb{R}$  tale che

$$t'(x) + at(x) = \delta(x) \quad (112)$$

Si nota facilmente che, se esiste,  $f(x) = (g \star t)(x)$  risolve l'equazione (111), infatti

$$(g \star t)'(x) + a(g \star t)(x) = (g \star t')(x) + (g \star at)(x) = [g \star (t' + at)](x) = g(x) \quad (113)$$

Quindi, il problema iniziale è ridotto a trovare una funzione  $t$  che soddisfi (112). Si può verificare facilmente che la seguente funzione soddisfa (112)

$$t(x) = h(x)e^{-ax} \quad (114)$$

dove  $h$  è la funzione di Heaviside. Infatti

$$t'(x) = e^{-ax}\delta(x) + h(x)(-a)e^{-ax} = \delta(x) - at(x) \quad (115)$$

Quindi, utilizzando questa tecnica è possibile, ad esempio, trovare una soluzione della seguente equazione differenziale ordinaria lineare non omogenea

$$f'(x) + 2f(x) = e^{3x} \quad (116)$$

Infatti, la seguente funzione soddisfa (116)

$$f(x) = \int_0^{+\infty} e^{-2\alpha} e^{3(x-\alpha)} d\alpha = e^{3x} \int_0^{+\infty} e^{-5\alpha} d\alpha = \frac{1}{5}e^{3x} \quad (117)$$

Se ora si è interessati a risolvere il seguente *problema di Cauchy*, con  $a \in \mathbb{R}$ ,  $x_0 \in \mathbb{R}$  e  $y_0 \in \mathbb{R}$ ,

$$y'(x) + ay(x) = g(x) \quad y(x_0) = y_0 \quad (118)$$

sarà sufficiente trovare una funzione  $f$  che risolva l'equazione differenziale ordinaria lineare non omogenea e poi trovare una funzione  $r : \mathbb{R} \rightarrow \mathbb{R}$  che risolva il seguente problema di Cauchy omogeneo

$$r'(x) + ar(x) = 0 \quad r(x_0) = y_0 - f(x_0) \quad (119)$$

per ottenere  $y(x) = r(x) + f(x)$ . Si noti che (119) è risolto da

$$r(x) = [y_0 - f(x_0)]e^{-a(x-x_0)} \quad (120)$$

Quindi, ad esempio, la soluzione del seguente problema di Cauchy

$$y'(x) + 2y(x) = e^{3x} \quad y(0) = 1 \quad (121)$$

è la funzione

$$y(x) = \frac{1}{5}e^{3x} + \frac{4}{5}e^{-2x} \quad (122)$$

com'è facile verificare direttamente.

# Approfondimenti sulle Reti Neurali del Corso di Intelligenza Artificiale

Federico Bergenti

19 maggio 2023

## 1 Reti Neurali e Approssimazione di Funzioni

Una rete neurale *feed forward* (o *in avanti*) è un grafo orientato aciclico pesato organizzato a livelli e come una *foresta*, quindi come un insieme di alberi tra loro disgiunti, ognuno dei quali è formato da una sequenza di *strati*. I nodi del grafo sono neuroni di McCulloch-Pitts e i pesi sugli archi sono i pesi entranti nei neuroni. Il vettore di uscita della rete è associato alle radici degli alberi e il vettore di ingresso è associato alle foglie degli alberi. Si noti che uno dei valori di ingresso per ogni strato della rete viene fissato ad un valore costante detto *valore di bias*, che normalmente vale  $-1$ . Se non viene esplicitamente indicato il contrario, si assume che una rete neurale in avanti abbia il massimo numero di connessioni che le consentono comunque di mantenere la struttura di foresta. Infine, si noti che, se una rete neurale non può essere descritta come una foresta, allora prende il nome di rete neurale *ricorrente*.

Normalmente si considerano reti neurali in avanti con una sola uscita in modo da semplificare la notazione senza perdere di generalità. Infatti, visto che gli alberi della foresta che descrive una rete neurale in avanti sono tra loro disgiunti è sempre possibile considerare i singoli alberi separatamente. Se non esplicitamente indicato il contrario, nel resto del testo si assumerà che le reti neurali in avanti abbiano una sola uscita e quindi che siano descritte da un solo albero. Ad esempio, in Figura 1 viene mostrata una rete neurale in avanti con una sola radice, corrispondente al neurone di uscita, uno strato di cinque figli della radice, corrispondente all'unico *strato nascosto*, e uno strato di quattro foglie, corrispondente ai neuroni di ingresso.

Un *Single-Layer Perceptron (SLP)* è una rete neurale in avanti con  $n \in \mathbb{N}_+$  ingressi reali, nessuno strato nascosto, un'uscita reale e una funzione di attivazione  $g : \mathbb{R} \rightarrow \mathbb{R}$ . Come discusso nel seguito, un SLP può essere utilizzato come un approssimatore di una funzione  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  di cui si conoscano i valori per alcuni vettori. Alcuni dei vettori per cui la funzione è nota vengono raccolti in un insieme detto *training set* e i rimanenti vengono raccolti in un insieme detto *test set*.

Per semplificare la notazione, senza comunque perdere di generalità, si ipotizza che tutti i vettori di  $\mathbb{R}^n$  utilizzati come ingressi del SLP, compresi quelli raccolti nel training set e nel test set, siano del tipo

$$\mathbf{x} = (x_1, x_2, \dots, x_{n-1}, x_n = -1) \tag{1}$$

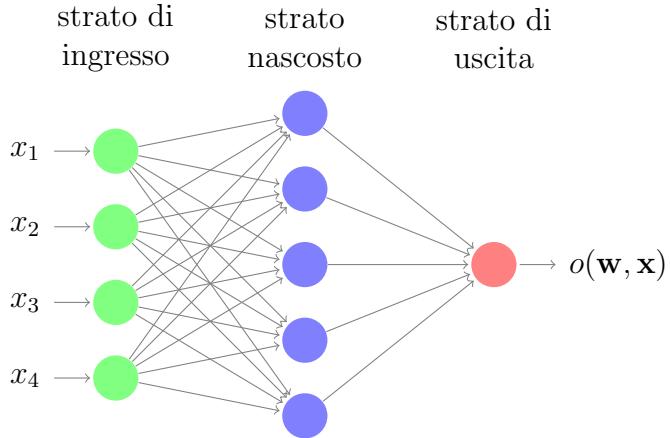


Figura 1: Esempio di una rete neurale in avanti

Naturalmente, se si vuole effettivamente disporre di  $n$  valori di ingresso per il SLP, sarà sufficiente considerare un SLP con  $n + 1$  neuroni di ingresso.

Dato un vettore di ingresso  $\mathbf{x} \in \mathbb{R}^n$ , il valore calcolato dal SLP per un certo *vettore dei pesi*  $\mathbf{w} \in \mathbb{R}^n$  vale

$$o(\mathbf{w}, \mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x}) = g\left(\sum_{i=1}^{n-1} w_i x_i - w_n\right) \quad (2)$$

perché l'ultima componente del vettore  $\mathbf{x}$ , quindi  $x_n$ , consente di trattare l'ultima componente di  $\mathbf{w}$ , quindi  $w_n$ , come un *peso di bias*.

Detto  $\mathbf{x} \in \mathbb{R}^n$  uno dei vettori nel training set, l'errore compiuto dal SLP nell'approssimazione della funzione  $f$  per lo specifico  $\mathbf{x}$  vale

$$\epsilon(\mathbf{w}, \mathbf{x}) = f(\mathbf{x}) - o(\mathbf{w}, \mathbf{x}) = f(\mathbf{x}) - g(\mathbf{w} \cdot \mathbf{x}) \quad (3)$$

Si noti che l'approssimazione della funzione  $f$  per il vettore  $\mathbf{x}$  è tanto migliore quanto  $|\epsilon(\mathbf{w}, \mathbf{x})|$  è piccolo. Quindi, la scelta del vettore dei pesi sarà tanto migliore quanto sarà in grado di rendere  $|\epsilon(\mathbf{w}, \mathbf{x})|$  piccolo. In sintesi, fissato un vettore  $\mathbf{x}$ , il problema di individuare un vettore dei pesi  $\mathbf{w}$  in grado di approssimare bene  $f(\mathbf{x})$  può essere espresso mediante la ricerca di un  $\mathbf{w}$  che minimizzi  $|\epsilon(\mathbf{w}, \mathbf{x})|$ .

Con l'obiettivo di utilizzare le comuni tecniche di ricerca dei minimi di funzioni reali in più variabili reali, normalmente si assume che  $g \in C^\infty(\mathbb{R})$  e si cerca il minimo dell'*errore quadratico*

$$e(\mathbf{w}, \mathbf{x}) = \frac{1}{2}\epsilon^2(\mathbf{w}, \mathbf{x}) \quad (4)$$

anziché di  $|\epsilon(\mathbf{w}, \mathbf{x})|$ . Naturalmente, un vettore dei pesi  $\mathbf{w}$  in grado di rendere minimo  $e(\mathbf{w}, \mathbf{x})$  è anche in grado di rendere minimo  $|\epsilon(\mathbf{w}, \mathbf{x})|$ .

In sintesi, fissato un vettore  $\mathbf{x} \in \mathbb{R}^n$ , l'errore quadratico è una funzione di  $\mathbf{w}$  di cui si può cercare il minimo mediante l'algoritmo di discesa del gradiente. Quindi, impostando il problema di approssimare  $f$  in questi termini, è necessario esprimere il gradiente di  $e(\mathbf{w})$  in modo da utilizzarlo per aggiornare i pesi del SLP mediante la regola

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla e(\mathbf{w}) \quad (5)$$

dove  $\alpha \in \mathbb{R}_+$  prende il nome di *coefficiente di apprendimento*. Si noti che l'aggiornamento del vettore dei pesi viene fatto usando  $\nabla e(\mathbf{w})$  e non il versore ottenibile da  $\nabla e(\mathbf{w})$ , rendendo quindi il processo di discesa sensibile anche alla norma di  $\nabla e(\mathbf{w})$ .

Per potere applicare l'algoritmo di discesa del gradiente è necessario esprimere il gradiente di  $e(\mathbf{w})$  in funzione di  $\mathbf{w}$  mediante il calcolo delle  $n$  derivate parziali di  $e(\mathbf{w})$ . Quindi, fissato  $1 \leq i \leq n$ , quello che interessa è

$$\frac{\partial e}{\partial w_i}(\mathbf{w}) = \frac{1}{2} \cdot 2 \cdot \epsilon(\mathbf{w}) \frac{\partial \epsilon}{\partial w_i}(\mathbf{w}) = \epsilon(\mathbf{w}) \frac{\partial \epsilon}{\partial w_i}(\mathbf{w}) \quad (6)$$

ma

$$\frac{\partial \epsilon}{\partial w_i}(\mathbf{w}) = -\frac{\partial o}{\partial w_i}(\mathbf{w}) \quad (7)$$

perché  $f(\mathbf{x})$  non dipende dal vettore dei pesi. Quindi, ricordando che l'uscita del SLP dipende unicamente da  $\mathbf{w}$  perché  $\mathbf{x}$  è fissato, si ottiene

$$\frac{\partial o}{\partial w_i}(\mathbf{w}) = \frac{\partial g}{\partial w_i}(\mathbf{w} \cdot \mathbf{x}) = g'(\mathbf{w} \cdot \mathbf{x}) \frac{\partial(\mathbf{w} \cdot \mathbf{x})}{\partial w_i} = g'(\mathbf{w} \cdot \mathbf{x}) x_i \quad (8)$$

perché

$$\frac{\partial(\mathbf{w} \cdot \mathbf{x})}{\partial w_i} = \sum_{j=1}^n x_j \frac{\partial w_j}{\partial w_i} \quad (9)$$

ma è semplice stabilire che

$$\frac{\partial w_j}{\partial w_i} = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{altrimenti} \end{cases} \quad (10)$$

Quindi, in sintesi, fissato  $\mathbf{x} \in \mathbb{R}^n$  nel training set è possibile aggiornare il vettore dei pesi del SLP in modo da spostare i pesi verso un minimo locale dell'errore compiuto nell'approssimazione di  $f(\mathbf{x})$  usando la regola

$$w_i \leftarrow w_i + \alpha \epsilon(\mathbf{w}) g'(\mathbf{w} \cdot \mathbf{x}) x_i = w_i + \alpha (f(\mathbf{x}) - g(\mathbf{w} \cdot \mathbf{x})) g'(\mathbf{w} \cdot \mathbf{x}) x_i \quad (11)$$

per  $1 \leq i \leq n$  e assumendo che l'ultima componente di  $\mathbf{x}$  valga  $-1$ , o una qualsiasi altra costante, purché usata in tutto il SLP.

Si noti che la formula di aggiornamento dei pesi di un SLP è spesso espressa nell'ipotesi che la funzione di attivazione  $g$  sia la funzione logistica

$$g(x) = \frac{1}{1 + e^{-x}} \quad (12)$$

perché vale la seguente proprietà della derivata della funzione logistica

$$g'(x) = g(x)(1 - g(x)) \quad (13)$$

e quindi

$$g'(\mathbf{w} \cdot \mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x})(1 - g(\mathbf{w} \cdot \mathbf{x})) = o(\mathbf{w}, \mathbf{x})(1 - o(\mathbf{w}, \mathbf{x})) \quad (14)$$

che permette di aggiornare i pesi senza dovere necessariamente valutare la derivata della funzione di attivazione.

## 2 Strati Nascosti e Approssimazione di Funzioni

Un *Multi-Layer Perceptron (MLP)* è una rete neurale in avanti con  $n \in \mathbb{N}_+$  ingressi reali, un numero  $k \in \mathbb{N}_+$  di strati nascosti, un'uscita reale e una funzione di attivazione  $g : \mathbb{R} \rightarrow \mathbb{R}$ . Per semplificare la notazione, ottenendo comunque risultati interessanti, nel resto di questo testo si ipotizza che i MLP abbiano un solo strato nascosto e che quindi  $k$  valga 1.

Si consideri un MLP con  $n \in \mathbb{N}_+$  ingressi reali, uno strato nascosto composto da  $m \in \mathbb{N}_+$  neuroni, un'uscita reale e una funzione di attivazione  $g : \mathbb{R} \rightarrow \mathbb{R}$ . L'obiettivo è usare il MLP come un approssimatore di una funzione  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  di cui si conoscono i valori per alcuni vettori. Come nel caso del SLP, alcuni dei vettori in cui la funzione è nota vengono raccolti in un training set, mentre i rimanenti vengono raccolti in un test set.

Per semplificare la notazione, senza comunque perdere di generalità, si ipotizza che tutti i vettori  $\mathbf{x} \in \mathbb{R}^n$  utilizzati come ingressi del MLP, compresi quelli raccolti nel training set e nel test set, siano del tipo

$$\mathbf{x} = (x_1, x_2, \dots, x_{n-1}, x_n = -1) \quad (15)$$

In più, si ipotizza che tutti i vettori  $\mathbf{y} \in \mathbb{R}^m$  che vengono posti in ingresso al neurone di uscita siano del tipo

$$\mathbf{y} = (y_1, y_2, \dots, y_{m-1}, y_m = -1) \quad (16)$$

Naturalmente, se si vuole effettivamente disporre di  $n$  valori di ingresso per il MLP, sarà sufficiente considerare un MLP con  $n+1$  neuroni di ingresso. Allo stesso modo, se si vuole disporre effettivamente di  $m$  valori in ingresso al neurone di uscita, sarà sufficiente considerare uno strato nascosto composto da  $m+1$  neuroni.

Sotto queste ipotesi, se  $\mathbf{w}_j \in \mathbb{R}^n$  è il vettore dei pesi associati agli archi che collegano lo strato di ingresso con il  $j$ -esimo neurone dello strato nascosto, e quindi  $1 \leq j \leq m-1$ , sarà

$$y_j = g(\mathbf{w}_j \cdot \mathbf{x}) \quad (17)$$

In più, se  $\mathbf{v} \in \mathbb{R}^m$  è il vettore dei pesi associati agli archi che collegano lo strato nascosto con il neurone di uscita, l'uscita della rete sarà

$$o(\mathbf{w}, \mathbf{x}) = g(\mathbf{v} \cdot \mathbf{y}) \quad (18)$$

dove è stato introdotto il *vettore dei pesi del MLP*

$$\mathbf{w} = (\mathbf{v}, \mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_{m-1}) \quad (19)$$

Quindi, scrivendo per esteso i prodotti scalari,

$$o(\mathbf{w}, \mathbf{x}) = g \left( \sum_{j=1}^m v_j y_j \right) = g \left( \sum_{j=1}^{m-1} v_j g \left( \sum_{i=1}^{n-1} w_{j,i} x_i - w_{j,n} \right) - v_m \right) \quad (20)$$

dove  $w_{j,i}$ , con  $1 \leq j \leq m-1$  e  $1 \leq i \leq n$ , è la  $i$ -esima componente del vettore  $\mathbf{w}_j$ , che non è altro che il vettore che collega lo strato di ingresso con il  $j$ -esimo neurone dello strato nascosto.

Fissato un vettore di ingresso  $\mathbf{x} \in \mathbb{R}^n$  del training set, l'errore commesso dal MLP nell'approssimare la funzione  $f$  per lo specifico  $\mathbf{x}$  vale

$$\epsilon(\mathbf{w}, \mathbf{x}) = f(\mathbf{x}) - o(\mathbf{w}, \mathbf{x}) \quad (21)$$

Introducendo l'errore quadratico e ipotizzando che  $g \in C^\infty(\mathbb{R})$ , esattamente com'è stato fatto per il SLP,

$$e(\mathbf{w}, \mathbf{x}) = \frac{1}{2} \epsilon^2(\mathbf{w}, \mathbf{x}) \quad (22)$$

si può cercare un vettore dei pesi del MLP utilizzando l'algoritmo di discesa del gradiente nel tentativo di minimizzare l'errore quadratico.

Nell'ipotesi che sia stato fissato un vettore di ingresso  $\mathbf{x} \in \mathbb{R}^n$  del training set, allora l'uscita del MLP, l'errore e l'errore quadratico dipendono solo dal vettore dei pesi del MLP. Si consideri il peso  $v_j$ , con  $1 \leq j \leq m$ , associato all'arco che collega il  $j$ -esimo neurone dello strato nascosto con il neurone di uscita

$$\frac{\partial e}{\partial v_j}(\mathbf{w}) = \frac{1}{2} \cdot 2 \cdot \epsilon(\mathbf{w}) \frac{\partial \epsilon}{\partial v_j}(\mathbf{w}) \quad (23)$$

ma

$$\frac{\partial \epsilon}{\partial v_j}(\mathbf{w}) = -\frac{\partial o}{\partial v_j}(\mathbf{w}) \quad (24)$$

perché  $f(\mathbf{x})$  non dipende dal vettore dei pesi del MLP. Ricordando ora la definizione di  $o(\mathbf{w})$ , si ottiene

$$\frac{\partial o}{\partial v_j}(\mathbf{w}) = g'(\mathbf{v} \cdot \mathbf{y}) \frac{\partial (\mathbf{v} \cdot \mathbf{y})}{\partial v_j} = g'(\mathbf{v} \cdot \mathbf{y}) y_j \quad (25)$$

e quindi, coerentemente con il fatto che, rispetto allo strato nascosto, l'uscita del MLP si comporta come un SLP, è possibile esprimere la seguente regola per l'aggiornamento dei pesi che collegano lo strato nascosto con l'uscita

$$v_j \leftarrow v_j + \alpha (f(\mathbf{x}) - g(\mathbf{v} \cdot \mathbf{y})) g'(\mathbf{v} \cdot \mathbf{y}) y_j \quad (26)$$

per  $1 \leq j \leq m$  e assumendo che l'ultima componente di  $\mathbf{y}$  valga  $-1$ , o una qualsiasi altra costante, purché usata in tutto il MLP.

Questa regola consente di modificare i pesi che collegano lo strato nascosto con il neurone di uscita in modo da cercare un minimo locale dell'errore commesso dal MLP quando riceve in ingresso il vettore  $\mathbf{x}$ . Si noti che prima viene applicato il MLP all'ingresso  $\mathbf{x}$  per calcolare il vettore  $\mathbf{y}$  e poi si applica in direzione opposta la regola di aggiornamento dei pesi.

Sempre nell'ipotesi che sia stato fissato un vettore di ingresso  $\mathbf{x} \in \mathbb{R}^n$ , consideriamo ora  $\mathbf{w}_j \in \mathbb{R}^n$ , che è il vettore dei pesi associato agli archi che collegano lo strato di ingresso con il  $j$ -esimo neurone dello strato nascosto, e quindi  $1 \leq j \leq m - 1$ . La derivata dell'errore quadratico rispetto alla  $i$ -esima componente di  $\mathbf{w}_j$  vale

$$\frac{\partial e}{\partial w_{j,i}}(\mathbf{w}) = \frac{1}{2} \cdot 2 \cdot \epsilon(\mathbf{w}) \frac{\partial \epsilon}{\partial w_{j,i}}(\mathbf{w}) \quad (27)$$

ma, come già discusso,

$$\frac{\partial \epsilon}{\partial w_{j,i}}(\mathbf{w}) = -\frac{\partial o}{\partial w_{j,i}}(\mathbf{w}) \quad (28)$$

perché  $f(\mathbf{x})$  non dipende dal vettore dei pesi del MLP. Ora, ricordando la definizione di  $o(\mathbf{w})$ , si ottiene

$$\frac{\partial o}{\partial w_{j,i}}(\mathbf{w}) = g'(\mathbf{v} \cdot \mathbf{y}) \frac{\partial(\mathbf{v} \cdot \mathbf{y})}{\partial w_{j,i}} \quad (29)$$

ma ora, contrariamente a quanto accaduto precedentemente, è necessario considerare la dipendenza del vettore  $\mathbf{y}$  dal vettore dei pesi del MLP

$$\frac{\partial(\mathbf{v} \cdot \mathbf{y})}{\partial w_{j,i}} = \mathbf{v} \cdot \frac{\partial \mathbf{y}}{\partial w_{j,i}} = \sum_{k=1}^m v_k \frac{\partial y_k}{\partial w_{j,i}} = \sum_{k=1}^{m-1} v_k \frac{\partial g(\mathbf{w}_k \cdot \mathbf{x})}{\partial w_{j,i}} \quad (30)$$

da cui

$$\frac{\partial(\mathbf{v} \cdot \mathbf{y})}{\partial w_{j,i}} = \sum_{k=1}^{m-1} v_k g'(\mathbf{w}_k \cdot \mathbf{x}) \frac{\partial(\mathbf{w}_k \cdot \mathbf{x})}{\partial w_{j,i}} \quad (31)$$

ma

$$\frac{\partial(\mathbf{w}_k \cdot \mathbf{x})}{\partial w_{j,i}} = \sum_{r=1}^n x_r \frac{\partial w_{k,r}}{\partial w_{j,i}} = \begin{cases} x_i & \text{se } r = i, k = j \\ 0 & \text{altrimenti} \end{cases} \quad (32)$$

e quindi

$$\frac{\partial(\mathbf{v} \cdot \mathbf{y})}{\partial w_{j,i}} = v_j g'(\mathbf{w}_j \cdot \mathbf{x}) x_i \quad (33)$$

Quindi, in sintesi, la regola di aggiornamento dei pesi  $w_{j,i}$  associati agli archi che collegano i neuroni di ingresso con il  $j$ -esimo neurone dello strato nascosto è

$$w_{j,i} \leftarrow w_{j,i} + \alpha (f(\mathbf{x}) - g(\mathbf{v} \cdot \mathbf{y})) g'(\mathbf{v} \cdot \mathbf{y}) v_j g'(\mathbf{w}_j \cdot \mathbf{x}) x_i \quad (34)$$

con  $1 \leq j \leq m-1$  e  $1 \leq i \leq n$ .

Come si evince dalla regola di aggiornamento dei pesi tra l'ingresso e lo strato nascosto, l'errore compiuto dal MLP viene distribuito tra i pesi in modo simile a come viene distribuito nei pesi associati agli archi che collegano lo strato nascosto con l'uscita del MLP. Infatti

$$\mathbf{v} \leftarrow \mathbf{v} + \alpha \epsilon(\mathbf{w}) g'(\mathbf{v} \cdot \mathbf{y}) \mathbf{y} \quad (35)$$

$$\mathbf{w}_j \leftarrow \mathbf{w}_j + \alpha \epsilon(\mathbf{w}) g'(\mathbf{v} \cdot \mathbf{y}) v_j g'(\mathbf{w}_j \cdot \mathbf{x}) \mathbf{x} \quad (36)$$

e quindi l'errore viene ripartito con coefficiente pari a 1 per l'aggiornamento del vettore dei pesi  $\mathbf{v}$  e con coefficiente pari a  $g'(\mathbf{v} \cdot \mathbf{y}) v_j$  per l'aggiornamento del vettore dei pesi  $\mathbf{w}_j$ , con  $1 \leq j \leq m-1$ .

In più, si noti come sia prima necessario calcolare i valori delle uscite dei neuroni nei vari strati prima di aggiornare i pesi. Quindi, prima si applica il MLP al vettore di ingresso  $\mathbf{x}$  per ottenere il vettore  $\mathbf{y}$  e l'uscita, mediante un processo detto *forward propagation*. Quindi, si valuta l'errore confrontando l'uscita con  $f(\mathbf{x})$  e si ripartisce l'errore commesso sugli strati, uno strato alla volta, mediante un processo detto *backward propagation*. Per questa sua caratteristica, l'algoritmo di addestramento

che prevede di minimizzare l'errore quadratico compiuto dal MLP sui vettori del training set prende il nome di *algoritmo di back propagation*.

Infine, si noti che la formula di aggiornamento dei pesi di un MLP è spesso espressa nell'ipotesi che la funzione di attivazione  $g$  sia una funzione logistica in modo da potere sfruttare la proprietà della derivata di questa funzione già discussa nel caso del SLP.

### 3 Teorema di Approssimazione Universale

Il seguente *teorema di approssimazione universale* consente di usare un MLP con uno strato nascosto come approssimatore di funzioni. Si noti che il MLP utilizzato impiega due funzioni di attivazione diverse per i due strati. In particolare, utilizza una generica funzione di attivazione fissata a priori nello strato nascosto e la funzione identità nello strato di uscita.

**Teorema 1** (Di Approssimazione Universale). *Dato  $n \in \mathbb{N}_+$ , sia  $D \subset \mathbb{R}^n$  compatto e sia  $f : S \rightarrow \mathbb{R}$  una funzione reale di più variabili reali continua nell'aperto  $S \subseteq \mathbb{R}^n$  con  $D \subset S$ . Sia  $g : \mathbb{R} \rightarrow \mathbb{R}$  una funzione continua, monotona strettamente crescente e limitata. Per ogni  $\epsilon \in \mathbb{R}_+$  esistono*

- $m \in \mathbb{N}_+$
- $v_j, \theta_j$ , con  $1 \leq j \leq m$
- $w_{j,i} \in \mathbb{R}$ , con  $1 \leq i \leq n$  e  $1 \leq j \leq m$

tali che la funzione

$$h(\mathbf{x}) = \sum_{j=1}^m v_j g \left( \sum_{i=1}^n w_{j,i} x_i + \theta_j \right) \quad (37)$$

soddisfi

$$\max_{\mathbf{x} \in D} |f(\mathbf{x}) - h(\mathbf{x})| < \epsilon \quad (38)$$

Quindi, fissata una certa tolleranza  $\epsilon$  nell'approssimazione di una funzione  $f$  continua su un compatto, è possibile trovare un MLP che possa essere utilizzato come approssimatore della funzione  $f$  con la tolleranza richiesta. Si noti che non è fissata la funzione di attivazione utilizzata per i neuroni dello strato di uscita. Viceversa, si assume che il MLP utilizzi una funzione di attivazione identità per il calcolo dell'uscita. Quest'ultima circostanza è del tutto ragionevole perché la scelta di una funzione di attivazione limitata fissata a priori avrebbe anche limitato a priori l'uscita del MLP e quindi avrebbe reso il MLP meno efficace nell'approssimare  $f$ , visto che non è stata posta una limitazione a priori ai valori di  $f$ .

Si noti che esiste una variante del teorema di approssimazione universale che esplicita come sia possibile utilizzare un MLP con due strati nascosti per approssimare con tolleranza piccola a piacere anche funzioni con un numero finito di discontinuità nel dominio considerato. Questo teorema non viene però discusso perché richiede una notazione più complessa di quella utilizzata in questo testo.

Nonostante sia possibile approssimare una qualsiasi funzione, anche non continua, con la tolleranza richiesta, vincolare la rete ad avere uno o due strati nascosti può aumentare troppo il numero di neuroni per ogni strato, aumentando quindi anche il numero di pesi da addestrare. Quindi, in alcune situazioni, si preferisce ridurre il numero di pesi da addestrare aumentando però il numero di strati nascosti. In questi casi si parla di *reti profonde* e la scelta di quali pesi fissare a valori noti viene fatta sulla base delle specifiche applicazioni.

In senso generale, le reti profonde non sono adatte a tutti gli scopi, ma spesso sono utili quando la funzione da approssimare ha un comportamento che dipende da alcune *feature locali* del vettore di ingresso. Gli strati nascosti vicini all'ingresso sono addestrati a riconoscere queste feature mentre gli strati nascosti vicini all'uscita sono addestrati a raggruppare queste feature per produrre un output globale ottenuto dall'identificazione delle feature locali.

Quando una rete profonda viene usata per approssimare una funzione che si basa su caratteristiche locali che possono essere presenti in varie parti del vettore di input, allora è spesso utile fare ricorso alle *reti convoluzionali* (*o convolutive*). Figura 2 mostra un esempio di rete convoluzionale.

I pesi di una rete convoluzionale sono addestrati in modo che possano comportarsi come un nucleo di convoluzione applicato al vettore di input. Siccome il nucleo di convoluzione viene normalmente scelto con un numero di componenti decisamente inferiore al numero di componenti del vettore di input, il numero di pesi che è necessario addestrare viene ridotto sensibilmente.

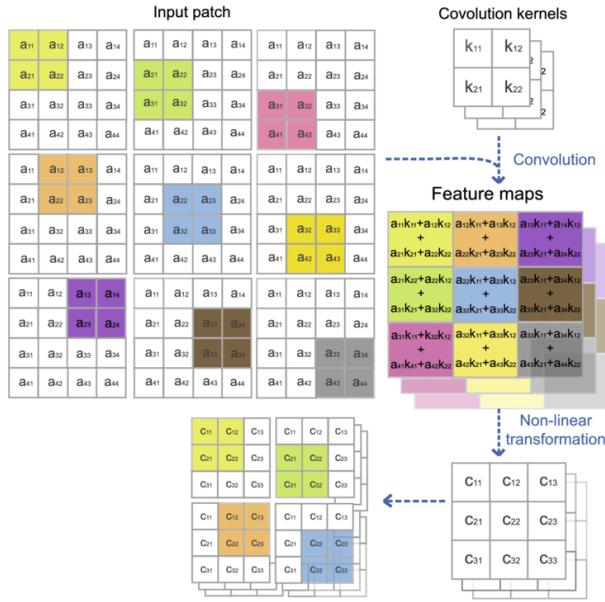


Figura 2: Un esempio di rete convoluzionale

## 4 Reti Neurali Ricorrenti e di Hopfield

Una rete neurale ricorrente è un grafo orientato pesato i cui nodi sono neuroni di McCulloch-Pitts e, per ogni neurone, i pesi sugli archi sono i pesi in ingresso al neurone. In generale, per una rete ricorrente non è più possibile identificare quale sia l'ingresso e quale sia l'uscita, anche se è sempre possibile identificare quale sia l'ingresso e quale sia l'uscita di un singolo neurone perché il grafo è orientato. Anche in questo caso, per ogni neurone della rete, viene fissato uno degli ingressi ad un valore costante che viene detto valore di bias e normalmente vale  $-1$ . Se non viene esplicitamente indicato il contrario, una rete neurale ricorrente si intende descritta da un grafo completamente connesso privo di autoanelli.

Una rete (*neurale*) di Hopfield è una rete neurale ricorrente in cui la funzione di attivazione dei neuroni è la funzione *signum*

$$\text{sgn}(x) = \begin{cases} +1 & \text{se } x > 0 \\ 0 & \text{se } x = 0 \\ -1 & \text{se } x < 0 \end{cases} \quad (39)$$

Visto che non è stato detto altrimenti, una rete di Hopfield è descritta da un grafo completamente connesso privo di autoanelli.

L'ingresso della rete viene fornito fissando i valori di uscita dei neuroni a valori iniziali nell'insieme  $\{-1, +1\}$ . L'uscita della rete viene ottenuta leggendo i valori delle uscite dei neuroni dopo un tempo di elaborazione sufficientemente lungo, nell'ipotesi che le uscite si siano stabilizzate a valori asintotici, se esistono. Si noti che, di norma, il valore 0 non viene usato come ingresso e si ipotizza che venga prodotto come uscita solo durante le fasi transitorie in cui l'uscita non è ancora considerata stabile. Quindi, una rete di Hopfield legge un ingresso binario e produce un'uscita binaria, sempre nell'insieme  $\{-1, +1\}$ .

Una rete di Hopfield è quindi un *sistema dinamico non lineare* il cui comportamento può essere studiato in modo analitico sotto opportune ipotesi. Normalmente, però, una rete di Hopfield viene *simulata a tempo discreto* aggiornando l'uscita di un neurone scelto casualmente per ogni istante di simulazione. Quindi, data una rete di Hopfield, si fissano inizialmente i valori delle uscite dei neuroni e poi, ciclicamente, si sceglie un neurone casualmente e si aggiorna la sua uscita sulla base dei relativi ingressi, fino al raggiungimento di un numero sufficiente di iterazioni o fino alla stabilizzazione dei valori di uscita dei neuroni.

Data una rete di Hopfield con  $n \in \mathbb{N}_+$  neuroni, fissato un qualsiasi istante di simulazione, il vettore  $\mathbf{s} \in \{-1, 0, +1\}^n$  formato delle uscite dei neuroni prima dell'aggiornamento delle uscite stesse viene detto *stato della rete*. Lo stato della rete evolve dinamicamente partendo da uno stato iniziale corrisponde al vettore di ingresso della rete e giungendo ad uno stato finale corrisponde al vettore di uscita della rete. Questo tipo di simulazione del comportamento di una rete di Hopfield viene detto *simulazione asincrona* e, contrariamente alla *simulazione sincrona* che non verrà discussa qui, descrive bene la rete nei termini di un sistema dinamico.

Si consideri una rete di Hopfield formata da  $n \in \mathbb{N}_+$  neuroni, ognuno dei quali è associato ad un vettore dei pesi e ad un valore di bias che vengono entrambi usati per produrre un valore di uscita. I neuroni della rete vengono normalmente numerati

da 1 a  $n$  e, quindi, l' $i$ -esimo neurone, con  $1 \leq i \leq n$ , viene descritto da un vettore dei pesi  $\mathbf{w}_i \in \mathbb{R}^n$  e da un valore di bias  $b_i \in \mathbb{R}$ , dove si assume che  $w_{i,i} = 0$  perché nella rete non sono presenti autoanelli. Se  $\mathbf{s} \in \mathbb{R}^n$  è lo stato attuale della rete, allora l'uscita dell' $i$ -esimo neurone vale

$$o_i = \text{sgn}(\mathbf{w}_i \cdot \mathbf{s} - b_i) = \text{sgn}\left(\sum_{j=1}^n w_{i,j} s_j - b_i\right) \quad (40)$$

Si noti che, per  $1 \leq i \leq n$ ,  $o_i \in \{-1, 0, +1\}$ . In più, si noti che l'uscita dell' $i$ -esimo neurone si dice *stabile*, o *stazionaria*, se  $o_i$  non varia anche dopo l'aggiornamento causato da una variazione dello stato della rete.

Si consideri una rete di Hopfield formata da  $n \in \mathbb{N}_+$  neuroni, ognuno dei quali è associato ad un vettore dei pesi  $\mathbf{w}_i \in \mathbb{R}^n$ , con  $1 \leq i \leq n$ , è possibile definire la matrice dei pesi della rete  $W = (w_{i,j})_{n,n}$ . Si noti che, come già discusso, la matrice dei pesi ha solo zeri sulla diagonale principale perché la rete non presenta autoanelli. Fissata una rete di Hopfield mediante la relativa matrice dei pesi della rete  $W$  e il relativo vettore di bias  $\mathbf{b}$ , è possibile definire l'*energia della rete* associata ad uno stato  $\mathbf{s} \in \mathbb{R}^n$  della rete come

$$E(\mathbf{s}) = -\frac{1}{2} \mathbf{s}^\top W \mathbf{s} + \mathbf{b} \cdot \mathbf{s} = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{i,j} s_i s_j + \sum_{i=1}^n b_i s_i \quad (41)$$

Si noti che, come di consueto, i vettori di  $\mathbb{R}^n$  vengono trattati come vettori colonna quando vengono considerati insieme a delle matrici. In più, si noti che la parola *energia* è stata scelta per la quantità  $E(\mathbf{s})$  perché vale il seguente teorema.

**Teorema 2** (Di Convergenza delle Reti Neurali di Hopfield). *Si consideri una rete neurale di Hopfield formata da  $n \in \mathbb{N}_+$  neuroni e descritta da una matrice dei pesi  $W$  e da un vettore di bias  $\mathbf{b}$ . Se la matrice dei pesi è simmetrica, allora, indipendentemente dallo stato iniziale, tutti i neuroni della rete tenderanno ad uno stato stazionario che corrisponde ad un minimo locale dell'energia della rete.*

*Dimostrazione.* Si consideri un passo della simulazione asincrona della rete e sia  $1 \leq a \leq n$  l'indice del neurone il cui stato viene aggiornato nel passo considerato. Se  $\mathbf{s} \in \mathbb{R}^n$  è lo stato della rete prima del passo di simulazione e  $\mathbf{s}' \in \mathbb{R}^n$  è lo stato della rete dopo il passo di simulazione, allora la variazione dell'energia da  $E(\mathbf{s})$  a  $E(\mathbf{s}')$  vale  $\Delta E = E(\mathbf{s}') - E(\mathbf{s})$

$$\begin{aligned} \Delta E = & -\frac{1}{2} \left( \sum_{i=1}^n w_{i,a} s'_i s'_a + \sum_{j=1}^n w_{a,j} s'_a s'_j - \sum_{i=1}^n w_{i,a} s_i s_a - \sum_{j=1}^n w_{a,j} s_a s_j \right) \\ & + b_a (s'_a - s_a) \end{aligned} \quad (42)$$

dove si è sfruttato il fatto che  $w_{a,a} = 0$  per ipotesi. Ma, siccome, per ipotesi, la matrice dei pesi della rete è simmetrica, vale

$$\Delta E = - \left( \sum_{i=1}^n w_{i,a} s'_i s'_a - \sum_{i=1}^n w_{i,a} s_i s_a \right) + b_a (s'_a - s_a) \quad (43)$$

e quindi

$$\begin{aligned}\Delta E &= - \left( s'_a \sum_{i=1}^n w_{i,a} s'_i - s_a \sum_{i=1}^n w_{i,a} s_i \right) + b_a (s'_a - s_a) \\ &= -(s'_a - s_a) \left( \sum_{i=1}^n w_{i,a} s_i - b_a \right)\end{aligned}\tag{44}$$

perché, nel passo di aggiornamento considerato, solo  $s_a$  può cambiare e quindi, per ogni  $1 \leq i \leq n$ , con  $i \neq a$ , vale  $s'_i = s_i$ . In più, si noti che

$$\left( \sum_{i=1}^n w_{i,a} s_i - b_a \right)\tag{45}$$

è il valore dell'uscita del neurone  $a$  prima dell'applicazione della funzione signum e quindi il suo segno è uguale al segno di  $s'_a$ . Quindi,

- Se l'uscita del neurone  $a$  non varia a causa dell'aggiornamento, allora  $\Delta E = 0$  e  $E'(\mathbf{s}) = E(\mathbf{s})$ ;
- Se l'uscita del neurone  $a$  varia a causa dell'aggiornamento e diventa 0, allora  $\Delta E = 0$  e  $E'(\mathbf{s}) = E(\mathbf{s})$ ;
- Se l'uscita del neurone  $a$  varia a causa dell'aggiornamento e diventa +1, allora  $\Delta E < 0$  e  $E'(\mathbf{s}) \leq E(\mathbf{s})$ ; e
- Se l'uscita del neurone  $a$  varia a causa dell'aggiornamento e diventa -1, allora  $\Delta E < 0$  e  $E'(\mathbf{s}) \leq E(\mathbf{s})$

Quindi, in sintesi, l'evoluzione dinamica dello stato della rete può solo fare decrescere o rimanere inalterata l'energia della rete, che quindi arriverà ad assestarsi asintoticamente ad un minimo locale.  $\square$

Si noti che, fissata una rete di Hopfield,  $E$  non è l'unica funzione che non cresce mai durante l'evoluzione dinamica della rete e, quindi, esistono altre funzioni che possono essere usate come energia della rete.

Una rete di Hopfield può essere usata come *memoria associativa* se, dato un insieme di vettori di ingresso detto training set, è possibile costruire una rete il cui stato evolva dinamicamente e si assesti ad uno dei vettori del training set *simile* al vettore di ingresso. In sostanza, quindi, lo stato della rete viene inizializzato con un vettore di ingresso e si prevede che la rete converga asintoticamente ad uno stato, tra quelli nel training set, per cui è alta la somiglianza con lo stato di ingresso.

Dato un training set, l'addestramento di una rete di Hopfield prevede di calcolare una matrice dei pesi e un vettore di bias che si comportino bene quando utilizzati con un adeguato test set. Si noti che, come si evince dal seguente teorema, l'addestramento di una rete di Hopfield non è *per rinforzo* perché non prevede che il training set venga proposto più volte alla rete. In più, l'addestramento è *non supervisionato* perché non si prevede che i vettori del training set siano associati a valori di uscita corretti.

**Teorema 3** (Di Addestramento con la Regola di Hebb). *Data una rete neurale di Hopfield con  $n \in \mathbb{N}_+$  neuroni, si consideri un training set  $T = \{\mathbf{x}_k\}_{k=1}^m$  formato da  $m \in \mathbb{N}_+$  vettori di  $\mathbb{R}^n$  tra loro mutuamente ortogonali e privi di componenti nulle, se la rete ha vettore di bias nullo e matrice dei pesi*

$$W = \frac{1}{m} \sum_{k=1}^m \mathbf{x}_k \mathbf{x}_k^\top - I_n \quad (46)$$

dove  $I_n$  è la matrice identità  $n \times n$ , allora l'energia della rete ammette minimi locali in corrispondenza dei vettori del training set.

*Dimostrazione.* Per prima cosa si noti che i vettori del training set sono vettori di  $\mathbb{R}^n$  e, quindi, sarà  $m \leq n$  perché è noto che un insieme di vettori mutuamente ortogonali è anche un insieme di vettori linearmente indipendenti. In più, si noti che la matrice considerata ha i seguenti elementi, per  $1 \leq i \leq n$  e  $1 \leq j \leq n$ ,

$$w_{i,j} = \frac{1}{m} \sum_{k=1}^m x_{k,i} x_{k,j} - \delta_{i,j} \quad (47)$$

dove  $\delta_{i,j}$  è la delta di Kronecker

$$\delta_{i,j} = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{se } i \neq j \end{cases} \quad (48)$$

Quindi, la matrice  $W$  è simmetrica e ha solo zeri sulla diagonale principale perché si è ipotizzato che i vettori del training set non abbiano componenti nulle. Si consideri la derivata dell'energia rispetto ad una generica componente dello stato della rete di indice  $1 \leq a \leq n$

$$\frac{\partial E}{\partial s_a}(\mathbf{s}) = -\frac{1}{2} \left( \sum_{i=1}^n w_{i,a} s_i + \sum_{j=1}^n w_{a,j} s_j \right) \quad (49)$$

dove si è sfruttato il fatto che  $w_{a,a} = 0$  e si è ricordato che, per ipotesi, il vettore di bias della rete è nullo. Siccome la matrice dei pesi considerata è simmetrica, vale

$$\frac{\partial E}{\partial s_a}(\mathbf{s}) = - \sum_{i=1}^n w_{i,a} s_i \quad (50)$$

e quindi

$$\frac{\partial E}{\partial s_a}(\mathbf{s}) = -\frac{1}{m} \sum_{k=1}^m \sum_{i=1}^n s_i x_{k,i} x_{k,a} + \sum_{i=1}^n \delta_{i,a} s_i \quad (51)$$

Si ricordi ora che i vettori del training set sono tra loro mutuamente ortogonali e quindi, per ogni  $1 \leq k \leq m$  e  $1 \leq h \leq m$ , con  $h \neq k$ , vale

$$\mathbf{x}_k \cdot \mathbf{x}_h = \sum_{r=1}^n x_{k,r} x_{h,r} = 0 \quad (52)$$

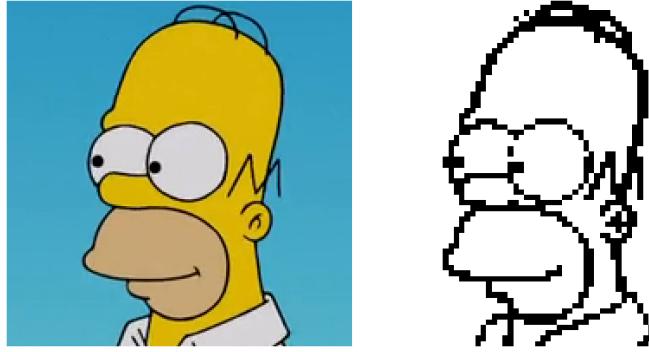


Figura 3: Un’immagine usata per alcuni esperimenti con una rete di Hopfield con  $n = 64 \times 64$  neuroni (sinistra) e il relativo vettore binario usato per l’addestramento della rete (destra)

Quindi, se la derivata dell’energia viene valutata in un generico vettore del training set di indice  $1 \leq h \leq m$ , vale

$$\frac{\partial E}{\partial s_a}(\mathbf{x}_h) = -\frac{1}{m} m x_{h,a} + x_{h,a} = 0 \quad (53)$$

perché, siccome i vettori del training set non hanno componenti nulle, vale che  $x_{h,i} x_{h,i} = 1$  per ogni  $1 \leq h \leq m$  e  $1 \leq i \leq n$ .  $\square$

Si noti che il teorema precedente non assicura che gli unici minimi locali della funzione energia siano quelli relativi ai vettori del training set e, quindi, fissato uno stato iniziale, non è garantito che la rete converga proprio ad uno dei vettori del training set. In generale, quindi, il *richiamo* di un vettore dalla rete, cioè la sua lettura a fronte di un vettore di ingresso, potrebbe essere non esatto. In sintesi, nonostante l’addestramento di una rete di Hopfield non sia per rinforzo, l’accuratezza nel richiamo dei vettori deve sempre essere quantificata mediante un’analisi con un opportuno test set.

Seguono alcuni esempi di richiamo di vettori mediante una rete addestrata con la *regola di Hebb*, che è la regola per la costruzione della matrice dei pesi del teorema precedente. Si noti che i vettori utilizzati per l’addestramento non sono mutuamente ortogonali, ma viene comunque usata la regola di Hebb ipotizzando di ottenere comunque buoni risultati.

Si consideri una rete di Hopfield in cui è stato memorizzato, mediante la regola di Hebb, il vettore mostrato nella parte destra della Figura 3. A fronte del vettore mostrato in Figura 4 la rete ha richiamato il vettore originale, come richiesto. Però, la stessa rete, una volta che è stato aumentato il numero di vettori memorizzati, come mostrato in Figura 5 inizia a comportarsi male e il suo comportamento peggiora al crescere del numero di vettori memorizzati. Si noti che gli errori nel richiamo dei vettori non sono necessariamente legati all’eccessivo numero di vettori memorizzati nella rete. Ad esempio, in questo caso, il numero di vettori memorizzati è sufficientemente piccolo da ritenere che gli errori non siano dovuti alla quantità di vettori memorizzati, ma alla loro dipendenza lineare.

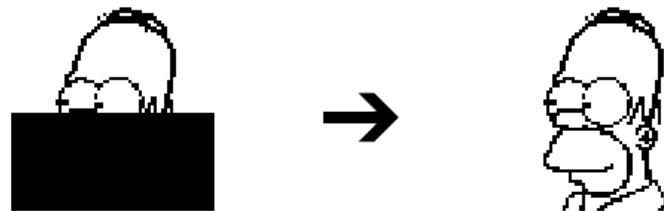


Figura 4: Un vettore usato come ingresso della rete di Hopfield addestrata con il vettore di Figura 3 (sinistra) e l'uscita ottenuta dalla rete (destra)

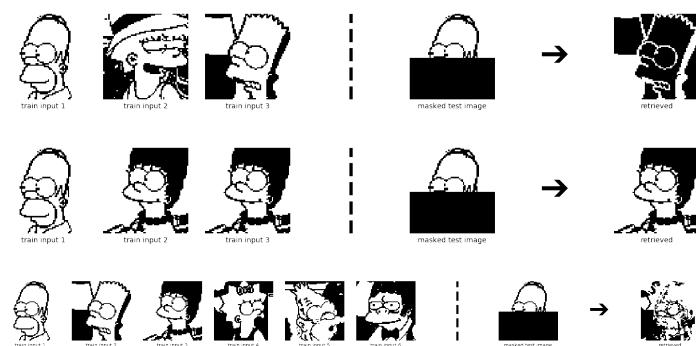


Figura 5: Alcuni esempi in cui la regola di Hebb per l'addestramento di una rete di Hopfield non consente di richiamare correttamente alcuni vettori

# Approfondimenti sui Problemi con Vincoli del Corso di Intelligenza Artificiale

Federico Bergenti

19 maggio 2023

## 1 Problemi di Soddisfacimento di Vincoli

Un *problema di soddisfacimento di vincoli* (*Constraint Satisfaction Problem, CSP*) è una tripla  $\langle V, D, C \rangle$  in cui

- $V \neq \emptyset$  è un insieme non vuoto e finito di simboli detti variabili con  $n = |V|$ ;
- $D \neq \emptyset$  è un insieme non vuoto di insiemi detti *domini* delle variabili con  $|D| \leq n$ ; e
- $C$  è un insieme finito di *vincoli*.

Detta  $dom : V \rightarrow D$  una funzione suriettiva totale che associa un dominio ad ogni variabile, si può parlare, per ogni variabile  $x \in V$ , del suo dominio  $dom(x)$ . In più, si suppone sempre che esista un ordinamento totale delle variabili e che questo ordinamento venga sempre usato in senso crescente quando vengono elencate le variabili. Normalmente, l'ordinamento utilizzato per le variabili è lasciato implicito ed evidente dal contesto. Utilizzando l'ordinamento delle variabili è possibile definire il dominio di un CSP  $\mathcal{P}$  come

$$dom(\mathcal{P}) = \prod_{x \in V} dom(x) \quad (1)$$

dove  $V$  è l'insieme delle variabili del CSP e le variabili vengono elencate nella costruzione del prodotto cartesiano in senso crescente secondo l'ordinamento scelto.

Dato un CSP con variabili in  $V$ , ogni vincolo  $c \in C$  è una coppia  $\langle V_c, \Delta_c \rangle$  dove  $V_c \subseteq V$  è un insieme di variabili del CSP e  $\Delta_c$  è

$$\Delta_c \subseteq \prod_{x \in V_c} dom(x) \quad (2)$$

dove le variabili vengono elencate nella costruzione del prodotto cartesiano in senso crescente secondo l'ordinamento scelto e  $\Delta_c$  viene detto insieme degli *assegnamenti (parziali) consistenti (o ammissibili)* del vincolo  $c$ . Se  $|V_c| = 1$ , allora  $c$  viene detto *unario*, se  $|V_c| = 2$ , allora  $c$  viene detto *binario*, mentre  $c$  viene detto *globale* in tutti

gli altri casi. Infine, si noti che  $vars(c) = V_c$  è un modo per fare riferimento alle variabili di un vincolo e

$$dom(c) = \prod_{x \in V_c} dom(x) \quad (3)$$

viene detto dominio del vincolo  $c$  se si assume che le variabili vengano elencate in senso crescente secondo l'ordinamento scelto.

Fissato un CSP  $\mathcal{P}$ , è sempre possibile estendere  $\Delta_c$  ad un sottoinsieme del dominio del CSP  $\Delta = dom(\mathcal{P})$  aggiungendo alle ennuple di  $\Delta_c$  le componenti mancanti. Siccome se una variabile non è coinvolta in un vincolo i suoi valori sono tutti e soli quelli del proprio dominio, le componenti aggiunte alle ennuple di  $\Delta_c$  potranno assumere un valore qualsiasi nei dominî delle rispettive variabili. Detta  $img : C \rightarrow \Delta$  una funzione suriettiva totale che associa ad ogni vincolo l'estensione a  $\Delta$  dell'insieme degli assegnamenti (parziali) consistenti, si può parlare, per ogni vincolo  $c \in C$ , del suo insieme degli *assegnamenti totali consistenti (o ammissibili)*  $img(c)$ .

Se tutti i dominî delle variabili di un vincolo sono finiti, allora è possibile descrivere il vincolo elencando tutti gli assegnamenti parziali consistenti in modo *estensionale* e il vincolo viene detto *tabellare*. La rappresentazione tabellare dei vincoli diventa impraticabile anche per dominî con pochi elementi e quindi spesso si preferisce la rappresentazione *intensionale*.

**Esempio 16.** Si consideri un CSP con tre variabili,  $x$ ,  $y$  e  $z$ , tali che  $dom(x) = dom(y) = dom(z) = [1..6]$ . L'insieme delle variabili del CSP è  $V = \{x, y, z\}$  e il dominio del CSP è  $\Delta = [1..6]^3$  dove, come di consueto, le variabili vengono ordinate in senso alfabetico. Il CSP contiene un vincolo  $c$  definito dalla proprietà

$$x \neq z \text{ devono essere diversi ed entrambi pari}$$

Quindi,  $c = \langle V_c, \Delta_c \rangle$  dove  $V_c = \{x, z\}$  e  $\Delta_c$  può essere espresso in forma estensionale come segue

$$\Delta_c = \{(2, 4), (2, 6), (4, 2), (4, 6), (6, 2), (6, 4)\} \quad (4)$$

dove, come di consueto, è stato utilizzato l'ordinamento alfabetico per le variabili. Si noti che l'estensione di  $\Delta_c$  ad un sottoinsieme di  $\Delta$  può essere espressa in forma estensionale come segue

$$img(c) = \{(2, 1, 4), (2, 2, 4), \dots, (6, 6, 4)\} \quad (5)$$

dove ogni elemento di  $\Delta_c$  è stato utilizzato per produrre 6 elementi di  $img(c)$ . L'insieme degli assegnamenti parziali consistenti di  $c$  può essere espresso in forma intensionale come segue

$$\Delta_c = \{(x, z) : x \in I \wedge z \in I \wedge x \neq z \wedge x \bmod 2 = 0 \wedge z \bmod 2 = 0\} \quad (6)$$

e quindi

$$img(c) = \{(x, y, z) : x \in I \wedge y \in I \wedge z \in I \wedge x \neq z \wedge x \bmod 2 = 0 \wedge z \bmod 2 = 0\} \quad (7)$$

La possibilità di estendere gli insiemi degli assegnamenti parziali consistenti ad insiemi di assegnamenti totali consistenti permette di definire in modo semplice

cosa si intenda per soluzione di un CSP. Dato un CSP  $\mathcal{P} = \langle V, D, C \rangle$ , l'insieme delle *soluzioni* di  $\mathcal{P}$  è

$$sol(\mathcal{P}) = \bigcap_{c \in C} img(c) \quad (8)$$

e il  $\mathcal{P}$  si dice *risolubile* (o *risolvibile*) se questo insieme non è vuoto. Dati due CSP  $\mathcal{P}_1$  e  $\mathcal{P}_2$  definiti sullo stesso insieme di variabili, si dice che  $\mathcal{P}_1$  è *equivalente* a  $\mathcal{P}_2$  se  $sol(\mathcal{P}_1) = sol(\mathcal{P}_2)$ .

Dato un CSP è interessante cercare delle trasformazioni in grado di generare dei CSP ad esso equivalenti ma che siano più utili per la ricerca delle soluzioni. In generale, però, sono interessanti anche le trasformazioni in grado di produrre dei CSP non equivalenti a quello di partenza ma che permettano, comunque, di studiare le proprietà di quest'ultimo.

**Esempio 17.** Si consideri un CSP  $\mathcal{P}$  con due variabili  $x$  e  $y$  entrambe con dominio  $[-10..10]$ . Il CSP ha un unico vincolo descritto dalla proprietà  $y = x^2$ . Si noti che il CSP è risolubile perché, ad esempio,  $x = -2$  e  $y = 4$  è una soluzione. Il CSP  $\mathcal{P}_1$  che restringe il dominio di  $y$  a  $[0..10]$  è equivalente a  $\mathcal{P}$  perché è facile vedere che entrambi ammettono lo stesso insieme di soluzioni. Viceversa, il CSP  $\mathcal{P}_2$  che restringe anche il dominio di  $x$  all'insieme  $[0..10]$  non è equivalente a  $\mathcal{P}$  perché, ad esempio,  $x = -2$  e  $y = 4$  è una soluzione di  $\mathcal{P}$  ma non è una soluzione di  $\mathcal{P}_2$ . Però, il CSP  $\mathcal{P}_2$ , che è ottenuto da  $\mathcal{P}$  con una trasformazione detta *di rottura della simmetria* (o *symmetry breaking*), è comunque interessante perché una volta risolto è possibile ottenere le soluzioni di  $\mathcal{P}$ . In più,  $\mathcal{P}_2$  ha un numero inferiore di elementi nel dominio di  $x$  e quindi la ricerca esaustiva delle soluzioni è facilitata.

Dato un CSP, ogni trasformazione che produce un CSP ad esso equivalente ma con meno vincoli o meno valori nei dominî viene detta *propagazione dei vincoli*. Se la trasformazione utilizza un sottoinsieme dei vincoli, allora viene detta *propagazione locale (dei vincoli)*. Se la trasformazione utilizza un sottoinsieme dei vincoli per rimuovere alcuni valori dai dominî delle variabili coinvolte nei vincoli considerati, allora viene detta *filtro (dei valori)*.

Normalmente, le trasformazioni tra CSP equivalenti vengono utilizzate per ottenere nuovi CSP con proprietà interessanti. Ad esempio, una trasformazione che spesso viene utilizzata permette di ottenere un CSP nodo-consistente equivalente ad un CSP dato. Si noti che un CSP si dice *nodo-consistente* (o *node-consistent*) se per ogni vincolo unario  $c$  vale la seguente proprietà

$$\forall v \in dom(x), v \in \Delta_c \quad (9)$$

dove  $x$  è l'unica variabile del vincolo  $c$  e  $\Delta_c$  è l'insieme degli assegnamenti parziali consistenti di  $c$ .

Un *algoritmo di nodo-consistenza* lavora su un CSP per produrre un CSP nodo-consistente ad esso equivalente eliminando dai dominî delle variabili tutti i valori che non rispettano la definizione precedente. In più, un algoritmo di nodo-consistenza rimuove anche tutti i vincoli unari dal CSP perché, una volta eliminati i valori inconsistenti dai dominî, i vincoli unari non sono più necessari.

In modo simile, un *algoritmo di arco-consistenza* lavora su un CSP per produrre un CSP arco-consistente ad esso equivalente eliminando dai dominî delle variabili

tutti i valori che non rispettano la definizione di arco-consistenza. Un CSP si dice *arco-consistente* (o *arc-consistent*) se per ogni vincolo binario  $c$  valgono le seguenti proprietà

$$\forall v_x \in \text{dom}(x), \exists v_y \in \text{dom}(y) : (v_x, v_y) \in \Delta_c \quad (10)$$

$$\forall v_y \in \text{dom}(y), \exists v_x \in \text{dom}(x) : (v_x, v_y) \in \Delta_c \quad (11)$$

dove  $x$  e  $y$  sono le due variabili del vincolo  $c$  e  $\Delta_c$  è l'insieme degli assegnamenti parziali consistenti di  $c$ .

Si noti che esistono vari algoritmi di arco-consistenza che assumono che i dominî delle variabili siano finiti. L'algoritmo normalmente più utilizzato è AC-3, che ha una complessità temporale asintotica di caso pessimo di classe  $O(e k^3)$ , dove  $e$  è il numero di vincoli binari e  $k$  è la cardinalità massima dei dominî delle variabili coinvolte nei vincoli binari. Si noti, comunque, che la complessità temporale asintotica di caso pessimo di un algoritmo di arco-consistenza ottimo è di classe  $O(e k^2)$ , a cui si può arrivare con l'algoritmo AC-4 mediante un significativo incremento della memoria utilizzata rispetto ad AC-3.

In più, si noti che la proprietà di arco-consistenza può essere estesa a vincoli globali introducendo la cosiddetta *iperarco-consistenza*. Infine, si noti che la nodo-consistenza coinvolge una sola variabile mentre la arco-consistenza coinvolge due variabili. Quindi è ragionevole aspettarsi di poter definire la cosiddetta *consistenza di percorso* costruendo percorsi di più di due variabili collegate da vincoli binari.

## 2 Problemi con Vincoli Polinomiali

La consistenza di nodo e la consistenza di arco permettono di rimuovere valori dai dominî delle variabili e sono spesso efficaci, specialmente se i vincoli sono espressi in forma tabellare. Viceversa, se i vincoli sono espressi in forma intensionale, ad esempio perché i dominî delle variabili non sono finiti, vengono spesso utilizzate altre forme di consistenza. Un esempio interessante a questo riguardo è rappresentato dai vincoli espressi mediante equazioni, disequazioni e disuguaglianze tra polinomi a coefficienti di variabili reali.

Si consideri un CSP  $\mathcal{P}$  con  $n \in \mathbb{N}_+$  variabili in  $V = \{x_i\}_{i=1}^n$  alle quali sono associati i dominî  $\text{dom}(x_i) = [\underline{x}_i, \bar{x}_i]$  rappresentati da intervalli chiusi in  $\mathbb{R}$ . Il problema può contenere anche più variabili, ma quelle di  $V$  sono quelle interessanti per i vincoli espressi mediante polinomi. Un vincolo  $c$  sulle variabili  $V_c \subseteq V$  di  $\mathcal{P}$  si dice *polinomiale* se il suo insieme degli assegnamenti parziali consistenti può essere espresso in modo intensionale come

$$\Delta_c = \{\mathbf{x} \in \text{dom}(c) : p(\mathbf{x}) \odot q(\mathbf{x})\} \quad (12)$$

dove  $\odot \in \{<, \leq, =, \neq, \geq, >\}$ ,  $p : \mathbb{R}^k \rightarrow \mathbb{R}$  e  $q : \mathbb{R}^k \rightarrow \mathbb{R}$  sono due funzioni polinomiali e  $k = |V_c|$ .

Si noti subito che ogni vincolo polinomiale può essere ridotto ad una delle due seguenti forme

$$\{\mathbf{x} \in \text{dom}(c) : p(\mathbf{x}) \geq 0\} \quad \text{oppure} \quad \{\mathbf{x} \in \text{dom}(c) : p(\mathbf{x}) > 0\} \quad (13)$$

utilizzando semplici manipolazioni algebriche e sfruttando il fatto che, data una funzione polinomiale  $p : \mathbb{R}^k \rightarrow \mathbb{R}$ , vale, per ogni  $\mathbf{x} \in \mathbb{R}^k$ ,

$$p(\mathbf{x}) = 0 \iff p(\mathbf{x}) \leq 0 \wedge p(\mathbf{x}) \geq 0 \quad (14)$$

$$p(\mathbf{x}) \neq 0 \iff p^2(\mathbf{x}) > 0 \quad (15)$$

Infine, si noti che è sempre possibile riscrivere i vincoli in modo che tutte le variabili abbiano dominî definiti come intervalli chiusi in  $\mathbb{R}_+$  mediante opportune trasformazioni affini operate sulle variabili.

Un CSP si dice *consistente sugli intervalli* (o *bounds-consistent*) se per ogni vincolo polinomiale  $c$  vale che per ognuna delle sue variabili  $x$  con dominio  $[\underline{x}, \bar{x}]$  esiste almeno un assegnamento parziale consistente del vincolo che abbia il valore  $\underline{x}$  per  $x$  ed, eventualmente un altro, assegnamento parziale consistente che abbia il valore  $\bar{x}$  per  $x$ .

Quindi, indipendentemente dalla consistenza dei valori all'interno dell'intervallo  $[\underline{x}, \bar{x}]$ , la bounds consistency considera solo la consistenza dei valori ai due estremi di ogni intervallo coinvolto nel vincolo. Si noti che spesso si è interessati al più piccolo intervallo, nel senso del contenimento, che garantisca la bounds consistency di una variabile di un vincolo.

Un algoritmo di bounds consistency è un algoritmo che trasforma un CSP in un secondo CSP ad esso equivalente in cui gli intervalli che definiscono i dominî delle variabili garantiscono la bounds consistency. Come già discusso, un algoritmo di bounds consistency può considerare solo variabili definite su  $\mathbb{R}_+$  e vincoli in forma di disequazione, stretta o meno. In queste ipotesi, è possibile definire un opportuno algoritmo di bounds consistency sfruttando la seguente proposizione.

**Proposizione 11.** *Dato  $n \in \mathbb{N}$ , sia  $p : \mathbb{R}^n \rightarrow \mathbb{R}$  una funzione polinomiale a coefficienti reali positivi, se  $\underline{\mathbf{x}} \in \mathbb{R}_+^n$  e  $\bar{\mathbf{x}} \in \mathbb{R}_+^n$ , allora per ogni  $\mathbf{v} \in [\underline{\mathbf{x}}, \bar{\mathbf{x}}]$  vale*

$$p(\underline{\mathbf{x}}) \leq p(\mathbf{v}) \leq p(\bar{\mathbf{x}}) \quad (16)$$

*nell'ipotesi non restrittiva che per ogni  $1 \leq i \leq n$  valga  $\underline{x}_i \leq \bar{x}_i$ .*

La descrizione generale di un algoritmo di bounds consistency che sfrutti la proposizione precedente richiede l'introduzione della *notazione dei multi-indici*. Quindi, anziché descrivere un algoritmo di bounds consistency nella sua generalità, verrà descritto sommariamente un metodo che permette di ridurre i domini delle variabili sfruttando il fatto che queste siano definite su intervalli chiusi di  $\mathbb{R}_+$ .

Si consideri vincolo descritto dalla proprietà  $p(\mathbf{x}) \geq 0$  nel caso in cui  $p$  sia una funzione lineare a coefficienti reali di  $n \in \mathbb{N}_+$  variabili

$$p(\mathbf{x}) = \sum_{i=1}^n a_i x_i + b \quad (17)$$

con, per semplicità  $b \geq 0$ . La proprietà che definisce il vincolo può essere riscritta spostando a destra tutti i termini di  $p$  con coefficienti negativi e quindi

$$\sum_{a_i > 0} a_i x_i + b \geq \sum_{a_i < 0} -a_i x_i \quad (18)$$

Fissata una variabile di indice  $k$ , se  $a_k > 0$  è possibile scrivere

$$\sum_{i \neq k, a_i > 0} a_i x_i + a_k x_k + b \geq \sum_{a_i < 0} -a_i x_i \quad (19)$$

Sfruttando la proposizione precedente è possibile ottenere

$$\sum_{i \neq k, a_i > 0} a_i x_i \leq \sum_{i \neq k, a_i > 0} a_i \bar{x}_i = u \quad l = \sum_{a_i < 0} -a_i \underline{x}_i \leq \sum_{a_i < 0} -a_i x_i \quad (20)$$

e quindi la proprietà che descrive il vincolo impone che

$$x_k \geq \frac{l - u - b}{a_k} \quad (21)$$

per ogni variabile  $x_k$  che viene moltiplicata in  $p$  per un coefficiente positivo. Si noti subito che è semplice rimuovere l'ipotesi  $b \geq 0$  e che, ragionamenti simili a quelli appena visti, consentono di identificare una disequazione da utilizzare per le variabili che vengono moltiplicate per un coefficiente negativo in  $p$ . Entrambe queste disequazioni possono anche essere utilizzate per trattare vincoli descritti mediante disequazioni strette, previo cambiamento del simbolo di diseguaglianza. Tutte queste disequazioni possono essere utilizzate ripetutamente per restringere i dominî delle variabili del vincolo considerato fino al raggiungimento di una delle seguenti condizioni:

1. L'applicazione delle disequazioni non genera nuovi restringimenti dei dominî e quindi è possibile cercare le soluzioni del CSP nei nuovi dominî; oppure
2. L'applicazione delle disequazioni ha svuotato almeno uno dei dominî e quindi è stato dimostrato che il CSP non ammette soluzioni.

I due esempi che seguono mostrano come applicare questo metodo nei due casi appena descritti.

**Esempio 18.** Si consideri un CSP con tre variabili  $x$ ,  $y$  e  $z$  i cui dominî sono  $\text{dom}(x) = \text{dom}(y) = \text{dom}(z) = [1, 10]$ . Si consideri il vincolo definito dalla proprietà

$$2x + 3y - z \leq 1 \quad (22)$$

La proprietà che definisce il vincolo può essere riscritta come

$$z + 1 \geq 2x + 3y \quad (23)$$

e quindi

$$z + 1 \geq 2x + 3y \geq 5 \quad (24)$$

da cui si evince che  $z \geq 4$ . Ma, ragionando sulla variabile  $x$  si ottiene

$$11 \geq z + 1 \geq 2x + 3y \geq 2x + 3 \quad (25)$$

da cui si evince che  $x \leq 4$ . Infine, ragionando sulla variabile  $y$  si ottiene

$$11 \geq z + 1 \geq 2x + 3y \geq 2 + 3y \quad (26)$$

da cui si evince che  $y \leq 3$ . Si noti che questi ragionamenti possono essere iterati finché nessun dominio viene più modificato oppure finché almeno uno dei dominî non diventa l'insieme vuoto. Nel primo caso si è ottenuto un CSP bounds consistent equivalente al CSP iniziale, mentre nel secondo caso è stato dimostrato che il CSP iniziale non ammette soluzioni. Provando ad iterare i ragionamenti fatti si nota subito che i dominî non vengono ulteriormente ridotti e quindi è stato ottenuto un nuovo CSP, equivalente a quello iniziale ma bounds consistent, con tre variabili  $x$ ,  $y$  e  $z$  aventi dominî

$$dom(x) = [1..4] \quad dom(y) = [1..3] \quad dom(z) = [4..10] \quad (27)$$

e un solo vincolo che richiede che valga la proprietà  $2x + 3y - z \leq 1$ . Si noti che il vincolo non può essere rimosso perché la proprietà di bounds consistency non fornisce informazioni sulla consistenza dei valori interni agli intervalli considerati.

**Esempio 19.** Si consideri un CSP con tre variabili  $x$ ,  $y$  e  $z$  i cui dominî sono  $dom(x) = dom(y) = dom(z) = [1, 10]$ . Si consideri il vincolo definito dalla proprietà

$$12x + 8y - z \leq 1 \quad (28)$$

La proprietà che definisce il vincolo può essere riscritta come

$$z + 1 \geq 12x + 8y \quad (29)$$

e quindi

$$z + 1 \geq 12x + 8y \geq 20 \quad (30)$$

da cui si evince che il vincolo richiede che  $z \geq 19$ . Però,  $z \leq 10$  per ipotesi e quindi è stato facilmente dimostrato che il CSP non ammette soluzioni.

Si noti che il metodo per la riduzione dei dominî delle variabili descritto sommariamente nel caso di funzioni lineari può essere esteso facilmente al caso di funzioni polinomiali non lineari ricordando che, nelle ipotesi considerate, se una variabile  $x$  ha per dominio  $dom(x) = [\underline{x}, \bar{x}] \subseteq \mathbb{R}_+$ , allora se  $m \in \mathbb{N}_+$

$$\underline{x}^{m-1} x \leq x^m \leq \bar{x}^{m-1} x \quad (31)$$

e quindi i termini non lineari del tipo  $x^m$  possono essere ridotti a termini lineari. In modo simile, se una seconda variabile  $y$  ha dominio  $dom(y) = [\underline{y}, \bar{y}] \subseteq \mathbb{R}_+$ , allora

$$xy \leq \underline{x}\underline{y} \leq \bar{x}\bar{y} \quad e \quad yx \leq \bar{y}\bar{x} \leq \underline{y}\underline{x} \quad (32)$$

e quindi i termini non lineari del tipo  $xy$  possono essere ridotti a termini lineari.

I vincoli polinomiali vengono spesso utilizzati restringendo i dominî delle variabili a intervalli in  $\mathbb{Z}_+$  e richiedendo che i coefficienti delle funzioni polinomi siano in  $\mathbb{Z}$ . In questo caso si parla di *vincoli polinomiali a dominî finiti* ed è possibile sfruttare le particolarità di questo tipo di vincoli per definire degli algoritmi di propagazione dei vincoli efficaci ed efficienti. In particolare, il fatto che le variabili abbiano dominî in  $\mathbb{Z}_+$  e il fatto che i coefficienti delle funzioni polinomiali siano in  $\mathbb{Z}$  permette di ridurre lo studio di questo tipo di vincoli a sole disequazioni non strette perché vale

$$p(\mathbf{x}) > 0 \iff p(\mathbf{x}) - 1 \geq 0 \quad (33)$$

per una qualsiasi funzione polinomiale a coefficienti interi e variabili intere positive.

# Approfondimenti sulla Programmazione Logica del Corso di Intelligenza Artificiale

Federico Bergenti

19 maggio 2023

## 1 Paradigmi di Programmazione

Seguendo l'approccio introdotto da *Robert W. Floyd*, ogni linguaggio di programmazione può essere descritto nei termini di uno specifico *paradigma di programmazione* che ne riassume le peculiarità elencando in modo astratto le caratteristiche degli *esecutori* in grado di produrre una *computazione* partendo da un programma scritto nel linguaggio. Spesso i linguaggi di programmazione seguono più paradigmi, ma normalmente è possibile identificare un paradigma principale.

I paradigmi di programmazione che vengono tradizionalmente identificati sono:

1. Paradigma *imperativo*: in cui ogni esecutore è una macchina di Turing e un programma descrive in modo esplicito e dettagliato quali azioni deve compiere l'esecutore per risolvere un problema.
2. Paradigma *dichiarativo*: in cui ogni esecutore è in grado di trovare una soluzione ad una classe di problemi mediante una tecnica risolutiva di uso generale e un programma descrive in modo esplicito e dettagliato un problema da risolvere.

Quindi, nella programmazione imperativa, un programma descrive una procedura per risolvere una classe di problemi e l'esecutore si limita ad applicare la procedura per risolvere una specifica istanza del problema. Viceversa, nella programmazione dichiarativa, un programma descrive un problema da risolvere in modo che l'esecutore possa risolverlo mediante l'applicazione del metodo risolutivo che ha a disposizione. Usando uno slogan: *Declarative programming tells what to do. Imperative programming tells how to do it.*

Nell'ambito della programmazione imperativa, vengono normalmente identificati due paradigmi principali:

1. Paradigma *procedurale*: in cui i comandi (o, in modo improprio, *statement*) da svolgere vengono forniti ad un esecutore raggruppandoli in *procedure*. Ogni procedura manipola lo stato dell'intera computazione in modo esplicito.

2. Paradigma *object-oriented*: in cui sono presenti più esecutori detti *oggetti* che interagiscono mediante lo scambio di *messaggi*. I comandi da svolgere da parte di ogni esecutore vengono raggruppati in procedure che vengono eseguite a seguito della ricezione dei messaggi. Normalmente, ogni procedura manipola solo lo stato dell'esecutore che la esegue e non lo stato dell'intera computazione.

Nell'ambito della programmazione dichiarativa, vengono normalmente identificati due paradigmi principali:

1. Paradigma *funzionale*: in cui il problema da risolvere viene descritto mediante un insieme di oggetti e un insieme di *funzioni* tra gli oggetti (relazioni per cui ogni elemento del dominio viene associato ad un unico elemento del codominio) e un esecutore è in grado di ragionare sulle funzioni e sulla loro composizione per risolvere il problema.
2. Paradigma *logico*: in cui il problema da risolvere viene descritto mediante un insieme di oggetti e un insieme di relazioni tra gli oggetti e un esecutore è in grado di ragionare sulle relazioni per risolvere il problema.

La programmazione funzionale viene usata molto spesso nell'intelligenza artificiale, specialmente nella tradizione statunitense. I linguaggi di programmazione funzionale più tradizionali sono *Lisp* (da *LISt Processor*) e i suoi dialetti e derivati (ad esempio *Scheme*). Il linguaggio di programmazione funzionale oggi più utilizzato è sicuramente *Haskell* ([www.haskell.org](http://www.haskell.org)).

La programmazione logica viene usata molto spesso nell'intelligenza artificiale, specialmente nella tradizione europea e giapponese. I linguaggi di programmazione logica più tradizionali sono *Prolog* (da *PROgrammation en LOGique*) e i suoi dialetti e derivati. Il linguaggio di programmazione logica oggi più utilizzato è ancora Prolog, eventualmente arricchito dalle funzionalità della *programmazione logica con vincoli* (*CLP*, da *Constraint Logic Programming*), e l'implementazione più diffusa di Prolog è SWI-Prolog ([www.swi-prolog.org](http://www.swi-prolog.org)).

Dal punto di vista dei costrutti linguistici messi a disposizione dal linguaggio, la differenza principale tra un linguaggio che segue il paradigma imperativo e un linguaggio che segue il paradigma dichiarativo è l'assenza in quest'ultimo dell'*assegnazione* (*o assegnamento*) *distruttiva*. L'assegnazione distruttiva viene utilizzata nel paradigma imperativo per consentire al programma di modificare esplicitamente lo stato della computazione. L'assenza dell'assegnazione distruttiva limita la possibilità di realizzare le comuni strutture di controllo della programmazione imperativa e, di fatto, rende la ricorsione uno strumento imprescindibile della programmazione funzionale e della programmazione logica.

Si noti che molti linguaggi seguono più di un paradigma e quindi vengono detti *multi-paradigma*. Ad esempio, il linguaggio *Kotlin* ([www.kotlinlang.org](http://www.kotlinlang.org)) fonde in modo organico la programmazione object-oriented di Java con la programmazione funzionale ispirata ad Haskell.

Infine, si noti che il paradigma di programmazione seguito da un linguaggio non ne limita in alcun modo la potenza. Infatti, tutti i linguaggi menzionati sono *Turing equivalenti* (*o completi*) in quanto sono in grado di istruire i relativi esecutori a calcolare una qualsiasi funzione (Turing) computabile.

## 2 Il Linguaggio dei Termini

Si considerino due insiemi di simboli  $A$  e  $V$  non entrambi vuoti e tra loro disgiunti e, quindi,  $(A \neq \emptyset \vee B \neq \emptyset) \wedge A \cap B = \emptyset$ . L'insieme  $A$  viene detto insieme degli *atomi* (o dei *simboli atomici*) e l'insieme  $V$  viene detto insieme delle *variabili* (o dei *simboli di variabile*). L'insieme  $T$  dei *termini* (*del primo ordine*) ottenibili da  $A$  e da  $V$  è costruito secondo le regole:

1. Una variabile  $v \in V$  è un termine;
2. Un atomo  $a \in A$  è un termine;
3. Se  $f \in A$ ,  $k \in \mathbb{N}_+$  e  $\{t_i\}_{i=1}^k \subseteq T$ , allora  $f(t_1, t_2, \dots, t_k)$  è un termine; e
4. Nient'altro è un termine.

Quindi, il *linguaggio dei termini* (*del primo ordine*)  $T$  è il linguaggio che si appoggia all'alfabeto improprio (perché non necessariamente finito)  $A \cup V$  e si basa sulla seguente grammatica non contestuale:

$$\begin{array}{lcl} \text{Term} & \rightarrow & \text{Variable} \mid \text{Atom} \mid \text{Atom}(\text{Arguments}) \\ \text{Arguments} & \rightarrow & \text{Term} \mid \text{Term}, \text{Arguments} \\ \text{Variable} & \rightarrow & \{v \in V\} \\ \text{Atom} & \rightarrow & \{a \in A\} \end{array} \quad (1)$$

Un termine  $t$  si dice *non strutturato* se è formato unicamente da un atomo o da una variabile. Viceversa, tutti gli altri termini si dicono *strutturati*. L'atomo più a sinistra di un termine strutturato viene detto *testa* del termine e, si noti, non può essere una variabile. Il numero di argomenti di un termine strutturato si dice *arità* del termine. Si noti che non è prevista un'arità fissa associata ad un atomo usato come testa di un termine strutturato. Quindi, ad esempio, in uno stesso termine un atomo può essere usato con arità diverse senza generare ambiguità. Per convenzione, l'arità di un termine non strutturato vale zero, ma spesso si evita di attribuire un'arità alle variabili. In alcuni contesti, anche agli atomi non è attribuita un'arità e, quando viene fatta questa scelta, gli atomi vengono separati in due insiemi disgiunti chiamati, rispettivamente, insieme delle *costanti* (o dei *simboli di costante*) e insieme delle *funzioni* (o dei *simboli di funzione*) e questi insiemi vengono scelti disgiunti dall'insieme delle variabili.

Seguono alcuni esempi di termini (con  $A = \{a, b, f, g\}$  e  $V = \{x, y\}$ ):

- $a$  e  $x$  sono termini.
- $f(a)$ ,  $f(x)$ ,  $f(a, f(x))$ ,  $f(g(a), f(g(y, b)))$  sono termini.

Si noti che i termini strutturati possono essere usati per descrivere degli alberi e, spesso, vengono descritti mediante dei diagrammi. Ad esempio, Figura 1 riporta il diagramma che descrive l'albero relativo al termine  $f(g(a, b), f(x, g(c, y)))$  (con  $A = \{a, b, c, f, g\}$  e  $V = \{x, y\}$ ).

Dato un termine  $t \in T$ ,  $\text{vars}(t)$  è l'insieme delle variabili presenti in  $t$ . Se  $\text{vars}(t) = \emptyset$ , allora  $t$  viene detto termine *ground*. Dato un insieme di atomi  $A \neq \emptyset$ ,

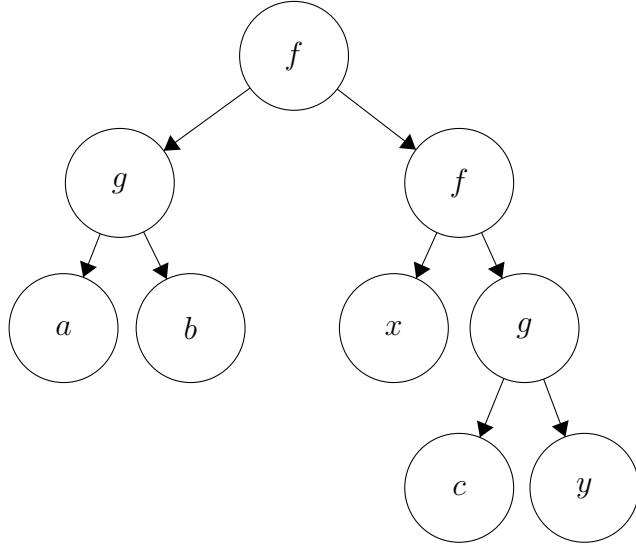


Figura 1: L’albero descritto dal termine  $f(g(a, b), f(x, g(c, y)))$ .

l’insieme dei termini ottenibili usando l’insieme di variabili  $V = \emptyset$  è un insieme di termini ground e viene detto *universo di Herbrand* ottenibile da  $A$ .

Dati due termini  $t_1 \in T$  e  $t_2 \in T$ , si dice che  $t_1$  è *sintatticamente equivalente* a  $t_2$  se  $t_1$  e  $t_2$  sono lo stesso elemento di  $T$ .

Quando si lavora con un insieme di termini è spesso utile permettere di trattare alcuni termini con varianti sintattiche speciali che ne semplificano la lettura e la scrittura oltre a renderne più esplicito il significato. Un caso tipico è quello che prevedere che l’insieme degli atomi contenga almeno due atomi:

1. Un atomo, normalmente indicato con il simbolo  $,$ , che viene utilizzato solo per formare termini strutturati con due argomenti; e
2. Un atomo, normalmente indicato con il simbolo  $[]$ , che viene utilizzato solo per formare termini non strutturati.

In questo caso, si utilizza la sintassi delle *liste* per semplificare la lettura e la scrittura di termini complessi. In particolare:

1. La lista  $[]$  è un termine che viene chiamato *lista vuota*;
2. La lista  $[h|r]$  è il termine  $.(h, r)$ , dove  $h$  è un termine che viene detto *testa della lista* e  $r$  è una lista detta *resto della lista*;
3. La lista  $[t_1, t_2, \dots, t_n]$  è il termine  $.(t_1, .(t_2, \dots, .(t_n, []) \dots))$ ; e
4. La lista  $[t_1, t_2, \dots, t_n|r]$  è il termine  $.(t_1, .(t_2, \dots, .(t_n, r) \dots))$ , dove  $r$  è una lista che, anche in questo caso, viene detta resto della lista.

Si noti però che se, ad esempio  $A = \{a, b\}$ , allora il termine  $[a|b]$  è un termine sintatticamente corretto ed equivale a  $.(a, b)$ , ma questo termine non viene considerata una lista perché si prevede che il resto di una lista sia a sua volta una lista, eventualmente vuota.

L'introduzione della sintassi delle liste non aggiunge niente all'insieme dei termini su cui si sta lavorando, ma permette di scrivere in modo semplice ed esplicito alcuni termini, quali, ad esempio (con  $A = \{a, b, c\}$  e  $V = \{x\}$ ):

- $[a, b, c]$  è la lista formata dagli atomi  $a$ ,  $b$  e  $c$ , in questo ordine.
- $[a|x]$  è una lista che inizia con l'atomo  $a$  e prosegue con un resto che viene associato alla variabile  $x$ .

Oltre alle liste, viene normalmente data la possibilità di introdurre degli operatori unari o binari per descrivere in modo semplice alcuni termini complessi. Quando viene data la possibilità di utilizzare operatori per costruire termini, viene anche data la possibilità di raggruppare gli operatori e i relativi argomenti mediante le parentesi tonde, come normalmente si fa nella scrittura delle espressioni matematiche.

Seguendo una tradizione consolidata, ogni operatore viene descritto mediante le seguenti tre proprietà:

1. L'atomo da utilizzare per indicare l'operatore e quindi, ad esempio,  $+$  o  $-$ ;
2. L'indice di precedenza dell'operatore; e
3. Il tipo di operatore, descritto mediante una delle sette tipologie ammesse.

L'indice di precedenza di un operatore è un numero naturale positivo con la convenzione che al crescere dell'indice di precedenza decresce la precedenza degli operatori. Infatti, viene utilizzata la convenzione che l'indice di precedenza delle variabili e degli atomi, eventualmente utilizzati come teste di termini strutturati, vale zero. In più, vale zero anche l'indice di precedenza dei termini racchiusi tra parentesi tonde.

Le sette tipologie di operatori ammesse sono descritte dalle seguenti etichette:

1.  $fx$ ,  $fy$  per gli operatori *unari prefissi*;
2.  $xfx$ ,  $xyf$ ,  $yfx$  per gli operatori *binari infissi*; e
3.  $xf$ ,  $yf$  per gli operatori *unari postfissi*.

Le etichette utilizzate per identificare le sette tipologie possono essere lette informalmente nel seguente modo:

1. Il simbolo  $f$  viene usato per individuare la posizione dell'operatore;
2. Il simbolo  $x$  viene letto: una variabile, un atomo, eventualmente usato come testa di un termine strutturato, un termine tra parentesi tonde o un operatore con indice di precedenza strettamente inferiore all'indice di precedenza dell'operatore che si sta descrivendo; e
3. Il simbolo  $y$  viene letto: una variabile, un atomo, eventualmente usato come testa di un termine strutturato, un termine tra parentesi tonde o un operatore con indice di precedenza minore o uguale all'indice di precedenza dell'operatore che si sta descrivendo.

Seguono alcune definizioni di operatori molto comuni indicando nelle triplette prima l'indice di precedenza, poi la tipologia e, per ultimo, l'atomo:

- $(700, xfx, <)$ ,  $(700, xfx, = <)$ ,  $(700, xfx, =)$ ,  $(700, xfx, >=)$ ,  $(700, xfx, >)$ .
- $(500, yfx, +)$ ,  $(500, yfx, -)$ .
- $(400, yfx, *)$ ,  $(400, yfx, /)$ .
- $(200, fy, +)$ ,  $(200, fy, -)$ .

L'introduzione della sintassi degli operatori unari e binari non aggiunge niente all'insieme dei termini su cui si sta lavorando ma permette di scrivere in modo semplice ed esplicito alcuni termini complessi. Seguono alcuni esempi dell'uso dei precedenti operatori per la costruzione di termini (con  $A = \{a, b, c, f\}$  e  $V = \{x, y\}$ ) in cui è stata usata la convenzione comune di racchiudere un atomo tra apici per garantire che non venga interpretato come un operatore:

1. Il termine  $f(x) + b$  equivale a  $' + '(f(x), b)$ ;
2. Il termine  $a + b - c$  equivale a  $' - ' + '(a, b), c)$ ;
3. Il termine  $a + b * c$  equivale a  $' + '(a, ' * '(b, c))$ ;
4. Il termine  $-a + b/y$  equivale a  $' + '(-'(a), '/(b, y))$
5. Il termine  $a + b = < c * a$  equivale a  $' = < ' + '(a, b), ' * '(c, a))$ .

Si noti che l'introduzione contemporanea di liste e operatori viene normalmente accompagnata dalla definizione dell'operatore  $\cdot$  come operatore unario postfisso con basso indice di precedenza (normalmente, 100). In questo caso, l'atomo  $\cdot$  utilizzato per le liste deve essere racchiuso tra apici per evitare che venga interpretato come operatore unario postfisso.

Si noti che le informazioni fornite per descrivere gli operatori consentono di identificare in modo univoco a quale termine si sta facendo riferimento utilizzando gli operatori, purché non vengano introdotti degli errori sintattici. L'introduzione degli operatori aumenta la possibilità di introdurre errori sintattici perché le sette tipologie di operatori vincolano i modi in cui gli operatori possono essere combinati. Ad esempio, i seguenti sono alcuni esempi di errori sintattici dovuti all'introduzione degli operatori per descrivere termini:

1. Il termine  $a + b >=$  utilizza l'operatore  $>=$  come unario postfisso anziché binario;
2. Il termine  $a = < b = < c$  utilizza l'operatore  $= <$  con un secondo operatore con lo stesso indice di precedenza a sinistra (o a destra); e
3. Il termine  $(a + b$  non ha le parentesi bilanciate.

Infine, conviene notare che anche se l'introduzione di operatori per usi specifici può semplificare la manipolazione di termini complessi, non conviene ricorrere troppo spesso a questi operatori.

### 3 Unificazione di Termini

Dato un insieme di termini  $T$  costruito su un insieme di atomi  $A$  e un insieme di variabili  $V$ , una *sostituzione* è un insieme finito ed eventualmente vuoto della forma

$$\{t_1/x_1, t_2/x_2, \dots, t_n/x_n\} \quad (2)$$

dove

1.  $x_1, x_2, \dots, x_n$  sono variabili;
2.  $t_1, t_2, \dots, t_n$  sono termini;
3. Per ogni  $1 \leq i \leq n$ ,  $t_i \neq x_i$ ; e
4. Per ogni  $1 \leq i \leq n$  e  $1 \leq j \leq n$ , se  $i \neq j$  allora  $x_i \neq x_j$ .

Se  $t$  è un termine e  $\theta$  è una sostituzione, allora  $t\theta$  è il termine che si ottiene sostituendo in  $t$  *simultaneamente* tutte le variabili nelle parti destre degli elementi della sostituzione  $\theta$  con i rispettivi termini.

Ad esempio, se  $\theta = \{f(z, z)/x, c/f\}$  (con  $A = \{c, f\}$  e  $V = \{x, z\}$ ) allora

$$p(f(x, y), x, g(z))\theta = p(f(f(z, z), y), f(z, z), g(c)) \quad (3)$$

Date due sostituzioni  $\theta = \{t_1/x_1, t_2/x_2, \dots, t_n/x_n\}$  e  $\sigma = \{u_1/y_1, u_2/y_2, \dots, u_m/y_m\}$  la *sostituzione composta*  $\theta \circ \sigma$  si ottiene dal seguente insieme

$$\{t_1\sigma/x_1, t_2\sigma/x_2, \dots, t_n\sigma/x_n, u_1/y_1, u_2/y_2, \dots, u_m/y_m\} \quad (4)$$

eliminando tutti gli elementi

1.  $t_j\sigma/x_j$  se vale  $t_j\sigma = x_j$ ; e
2.  $u_j/y_j$  se  $y_j \in \{x_1, x_2, \dots, x_n\}$ .

Quindi, ad esempio, se  $\theta = \{f(y)/x, z/y\}$  e  $\sigma = \{a/x, b/y, y/z\}$  (con  $A = \{a, b, f\}$  e  $V = \{x, y, z\}$ ) allora

$$\theta \circ \sigma = \{f(b)/x, y/z\} \subset \{f(b)/x, y/y, a/x, b/y, y/z\} \quad (5)$$

La proprietà fondamentale della composizione di sostituzioni, che ne giustifica il nome, è che per ogni coppia di sostituzioni  $\theta, \sigma$  vale

$$t(\theta \circ \sigma) = (t\theta)\sigma \quad (6)$$

per ogni termine  $t$ . Quindi, ad esempio, se  $\theta = \{f(y)/x, z/y\}$ ,  $\sigma = \{a/x, b/y, y/z\}$  e  $t = h(x, g(y), z)$  (con  $A = \{a, b, f, g, h\}$  e  $V = \{x, y, z\}$ ), allora

$$t\theta = h(f(y), g(z), z) \quad (t\theta)\sigma = h(f(b), g(y), y) \quad (7)$$

e, infatti,  $t(\theta \circ \sigma) = h(f(b), g(y), y)$ .

Una sostituzione  $\theta$  di dice *unificatore* per l'insieme di  $k \geq 2$  termini  $S = \{t_i\}_{i=1}^k$  se vale la seguente catena di uguaglianze

$$t_1\theta = t_2\theta = \cdots = t_k\theta \quad (8)$$

Un insieme di termini si dice *unificabile* se esiste almeno un unificatore per l'insieme. Si noti che un insieme di termini può ammettere più unificatori, ad esempio, perché alcuni unificatori utilizzano più variabili del necessario. L'insieme di due termini  $\{p(x), p(y)\}$  (con  $A = \{p\}$  e  $V = \{x, y, z\}$ ) ammette almeno tre unificatori:

$$\{x/y\} \quad \{y/x\} \quad \{z/x, z/y\} \quad (9)$$

dove i primi due unificatori possono essere resi uguali semplicemente cambiando in modo coerente i nomi delle variabili e il terzo unificatore è stato ottenuto mediante l'introduzione della variabile, non necessaria,  $z$ .

Viceversa, se un insieme di termini  $S$  è unificabile allora ammette un unico *unificatore più generale* (*Most General Unifier*, MGU), a meno di cambiamenti coerenti dei nomi delle variabili. Il MGU di un insieme di termini  $S$  è definito come un unificatore  $mgu(S)$  tale che qualsiasi sia un unificatore  $\theta$  di  $S$  esiste almeno una sostituzione  $\sigma$  tale che  $\theta = mgu(S) \circ \sigma$ . Quindi, a giustificazione del nome, il MGU di un insieme di termini non compie sostituzioni non necessarie ma compie solo le sostituzioni necessarie a rendere gli elementi dell'insieme congiuntamente uguali. Dei tre unificatori elencati nell'esempio precedente, solo i primi due sono MGU e, infatti, sono uguali a meno di cambiamenti coerenti dei nomi delle variabili.

Ad esempio, se  $S = \{p(x), p(f(y))\}$  allora  $\theta = \{f(a)/x, a/y\}$  (con  $A = \{a, f, p\}$  e  $V = \{x, y\}$ ) è un unificatore di  $S$ , infatti

$$p(x)\theta = p(f(a)) \quad p(f(y))\theta = p(f(a)) \quad (10)$$

ma  $\theta$  non è il MGU di  $S$  perché, per unificare gli elementi di  $S$ , è sufficiente  $\alpha = \{f(y)/x\}$  e, infatti,  $\theta = \alpha \circ \{a/y\}$ .

Dato un insieme di termini  $T$  costruito su un insieme di atomi  $A$  e un insieme di variabili  $V$ , un *problema di unificazione (sintattica)* è un insieme del tipo

$$\{l_1 \doteq r_1, l_2 \doteq r_2, \dots, l_n \doteq r_n\} \quad (11)$$

dove  $\{l_i\}_{i=1}^n \subseteq T$  e  $\{r_i\}_{i=1}^n \subseteq T$  e ogni elemento del problema viene detto *equazione*. Dato un problema di unificazione, si cerca una sostituzione  $\theta$  tale che valga  $l_i\theta = r_i\theta$  per ogni  $1 \leq i \leq n$ . Se una tale sostituzione esiste, allora il problema viene detto risolubile e la sostituzione trovata ne è una soluzione.

Si noti che vengono definite le seguenti funzioni per un problema di unificazione  $S$ , generalizzando opportunamente le relative funzioni definite per i termini:

1.  $vars(S)$  è l'insieme delle variabili contenute nelle parti sinistre e nelle parti destre delle equazioni di  $S$ ; e
2. Se  $\theta$  è una sostituzione,  $S\theta$  è il problema di unificazione ottenuto applicando la sostituzione alle parti sinistre e alle parti destre di tutte le equazioni in  $S$ .

Il seguente algoritmo è in grado di stabilire se un problema di unificazione è risolubile e, in caso, di trovarne una soluzione. L'algoritmo (*di unificazione di Martelli e Montanari*) parte da un problema di unificazione  $S$  e termina producendo uno dei seguenti risultati:

1. Il simbolo  $\perp$  se il problema di unificazione non è risolubile; oppure
2. Un problema di unificazione equivalente ad  $S$  ma con solo variabili nelle parti sinistre delle equazioni, e quindi nella forma  $\{x_1 \doteq t_1, x_2 \doteq t_2, \dots, x_m \doteq t_m\}$ , da cui è possibile costruire una soluzione  $\{t_1/x_1, t_2/x_2, \dots, t_m/x_m\}$ .

L'algoritmo può essere descritto come la seguente *funzione non deterministica unify* che ha come unico argomento un problema di unificazione:

1.  $\text{unify}(G \cup \{f(l_1, l_2, \dots, l_m) \doteq f(r_1, r_2, \dots, r_m)\}) = \text{unify}(G \cup \{l_1 \doteq r_1, l_2 \doteq r_2, \dots, l_m \doteq r_m\})$  (caso *decompose*);
2.  $\text{unify}(G \cup \{x \doteq t\}) = \text{unify}(G\{t/x\} \cup \{x \doteq t\})$  se  $x \in \text{vars}(G)$  e  $x \notin \text{vars}(t)$  (caso *eliminate*);
3.  $\text{unify}(G \cup \{f(l_1, l_2, \dots, l_m) \doteq x\}) = \text{unify}(G \cup \{x \doteq f(l_1, l_2, \dots, l_m)\})$  se  $x \in V$  (caso *swap*);
4.  $\text{unify}(G \cup \{t \doteq t\}) = \text{unify}(G)$  (caso *delete*);
5.  $\text{unify}(G \cup \{f(l_1, l_2, \dots, l_m) \doteq g(r_1, r_2, \dots, r_k)\}) = \perp$  se  $f \neq g \vee m \neq k$  (caso *conflict*);
6.  $\text{unify}(G \cup \{x \doteq f(t_1, t_2, \dots, t_m)\}) = \perp$  se  $x \in \text{vars}(f(t_1, t_2, \dots, t_m))$  (caso *check*).

Si noti che l'algoritmo è utilizzabile per calcolare il MGU di due termini  $l$  e  $r$  tra loro unificabili. Infatti, l'applicazione dell'algoritmo al problema di unificazione  $\{l \doteq r\}$  consente di trovare immediatamente  $\text{mgu}(\{l, r\})$ .

In più, si noti che il caso check aumenta la complessità computazionale temporale asintotica di caso pessimo dell'algoritmo e quindi, alle volte, si preferisce omettere questo caso per ottenere una procedura di unificazione *senza occurs check*. In questo caso è anche necessario rimuovere le restrizioni dal caso eliminate per garantire di potere elaborare problemi tipo  $\{x \doteq f(x)\}$  e, quindi, non sarà più possibile garantire la terminazione della procedura (e non più algoritmo) di unificazione.

Infine, si noti che i problemi di unificazione possono essere generalizzati a *problemi di unificazione modulo teoria* se la relazione  $\doteq$  ammette delle proprietà aggiuntive fornite dalla teoria considerata oltre ad essere una relazione di equivalenza tra termini. In questi casi, l'algoritmo visto deve essere opportunamente esteso per tenere conto delle proprietà fornite dalla teoria considerata. Ad esempio, se l'atomo  $f$  denota un'operazione commutativa della teoria considerata, allora il problema di unificazione  $\{f(a, b) \doteq f(x, y)\}$  dovrà ammettere due soluzioni  $\{a/x, b/y\}$  oppure  $\{b/x, a/y\}$ . Quindi, non solo si dovrà modificare l'algoritmo di unificazione per tenere conto della commutatività dell'operazione denotata da  $f$  ma si dovrà anche ammettere la possibilità che due termini unificabili possano avere più MGU tra loro effettivamente diversi.

## 4 Sintassi di Prolog<sub>0</sub>

Si consideri un insieme di atomi  $A$  e un insieme di variabili  $V$  tali che:

1.  $A$  è formato da tutte e sole le parole che iniziano con una lettera minuscola e proseguono con, eventualmente zero, lettere, numeri e underscore; e
2.  $V$  è formato da tutte e sole le parole che iniziano con una lettera maiuscola o un underscore e proseguono con, eventualmente zero, lettere, numeri e underscore senza però essere formate unicamente da underscore.

Si noti che  $A \neq \emptyset$ ,  $V \neq \emptyset$ ,  $A \cap V = \emptyset$  ed entrambi gli insiemi sono infiniti numerabili. In più, si noti che non sono presenti simboli di punteggiatura nei due insiemi.

Si consideri il linguaggio  $\text{Prolog}_0$  descritto dalla seguente grammatica che usa l'alfabeto improprio  $A \cup V$  e riprende la grammatica dei termini nella parte finale:

$$\begin{aligned}
\text{Program} &\rightarrow \text{Clauses} \\
\text{Clauses} &\rightarrow \text{Clause} \mid \text{Clause Clauses} \\
\text{Clause} &\rightarrow \text{Fact} \mid \text{Rule} \\
\text{Fact} &\rightarrow \text{Head}. \\
\text{Rule} &\rightarrow \text{Head} :- \text{Body}. \\
\text{Head} &\rightarrow \text{Atom} \mid \text{Atom}(\text{Arguments}) \\
\text{Body} &\rightarrow \text{Conjuncts} \\
\text{Conjuncts} &\rightarrow \text{Conjunct} \mid \text{Conjunct, Conjuncts} \\
\text{Conjunct} &\rightarrow \text{Atom} \mid \text{Atom}(\text{Arguments}) \\
\text{Term} &\rightarrow \text{Variable} \mid \text{Atom} \mid \text{Atom}(\text{Arguments}) \\
\text{Arguments} &\rightarrow \text{Term} \mid \text{Term, Arguments} \\
\text{Variable} &\rightarrow \{v \in V\} \\
\text{Atom} &\rightarrow \{a \in A\}
\end{aligned} \tag{12}$$

Il linguaggio  $\text{Prolog}_0$  è un dialetto minimalista (ma non troppo) del linguaggio Prolog. La seguente nomenclatura è definita per  $\text{Prolog}_0$  ma vale anche per Prolog:

1. Gli elementi derivabili dalla produzione  $\text{Program}$ , e quindi gli elementi  $\pi$  tali che  $\text{Program} \xrightarrow{*} \pi$ , si chiamano *programmi*.
2. Gli elementi derivabili dalla produzione  $\text{Clause}$ , e quindi gli elementi  $\gamma$  tali che  $\text{Clause} \xrightarrow{*} \gamma$ , si chiamano *clausole definite*.
3. Gli elementi derivabili dalla produzione  $\text{Fact}$ , e quindi gli elementi  $\lambda$  tali che  $\text{Fact} \xrightarrow{*} \lambda$ , si chiamano *fatti* e, per ogni fatto, è definita una *testa*.
4. Gli elementi derivabili dalla produzione  $\text{Rule}$ , e quindi gli elementi  $\rho$  tali che  $\text{Rule} \xrightarrow{*} \rho$ , si chiamano *regole* e, per ogni regola, è definita una *testa* ed è definito un *corpo*.
5. Gli elementi derivabili dalla produzione  $\text{Conjunct}$ , e quindi gli elementi  $\alpha$  tali che  $\text{Conjunct} \xrightarrow{*} \alpha$ , si chiamano *congiunti* e ogni congiunto è un termine che non può essere una variabile isolata.

Infine, si noti che gli elementi derivabili da *Conjuncts* sono sequenze di congiunti e quindi è possibile estendere ad un'intera sequenza una qualsiasi operazione che coinvolga un congiunto semplicemente applicando l'operazione ad ogni congiunto della sequenza e separando i risultati con delle virgole. L'utilizzo delle virgole per separare i risultati consente di ottenere ancora un elemento derivabile da *Conjuncts*. In particolare, se  $\theta$  è una sostituzione, e  $Conjuncts \xrightarrow{*} \beta$ , allora è possibile parlare di  $\beta \theta$  applicando la sostituzione ad ogni congiunto di  $\beta$  e separando i risultati ottenuti con delle virgole.

Ad esempio, dato  $\{a, b, p, q\} \subseteq A$  e  $\{X, Y\} \subseteq V$ , il seguente è un programma scritto in Prolog<sub>0</sub>:

```
p(a).
p(X).
p(b, X) :- q(Y, X).
p(X, b) :- q(X, a), p(a, X).
```

Dato un programma scritto in Prolog<sub>0</sub> è possibile ottenere una computazione abbinando al programma un *goal*. Una computazione così ottenuta, se produce un risultato, produce una o più sostituzioni, eventualmente vuote, che coinvolgono un sottoinsieme delle variabili del goal. Non tutte le computazioni, però, producono un risultato perché, per alcune computazioni, il risultato non è definito mentre, per altre computazioni, non necessariamente diverse dalle precedenti, la computazione non termina.

Dal punto di vista sintattico, un goal è descritto da una regola privata della testa. Un elemento dell'insieme delle clausole definite unito all'insieme dei goal viene detto *clausola di Horn*. Si noti che i goal non possono essere utilizzati nel testo di un programma Prolog<sub>0</sub> mentre possono essere utilizzati nel testo di un programma Prolog con l'ovvio significato di attivare una computazione per ogni goal incontrato durante il caricamento in memoria del programma.

Ad esempio, si consideri il seguente programma *Prolog<sub>0</sub>*:

```
m(a).
m(b).

f(c).
f(d).

c(X) :- m(X).
c(X) :- f(X).
```

Questo programma può essere utilizzato con i seguenti goal, com'è semplice verificare utilizzando SWI-Prolog o la sua interfaccia Web *SWISH* ([swish.swi-prolog.org](http://swish.swi-prolog.org)). Si noti che, per fornire un goal all'interfaccia interattiva di SWI-Prolog o a SWISH, deve essere omesso il simbolo `:-` iniziale. In più, si noti che i risultati successivi al primo vengono ottenuti premendo ; quando si usa l'interfaccia interattiva di SWI-Prolog mentre vengono ottenuti con gli appositi pulsanti grafici quando si usa SWISH. Infine, si noti che viene stampato **true** quando il risultato della computazione è la sostituzione  $\emptyset$  e viene stampato **false** quando non è definito un risultato per la computazione.

```

:- m(a).
true

:- m(b).
true

:- m(w).
false.

:- f(X).
X=c ; X=d

:- c(X).
X=a ; X=b ; X=c ; X=d

```

La sintassi del linguaggio Prolog<sub>0</sub> può essere estesa sfruttando la possibilità di introdurre liste e operatori come varianti sintattiche del linguaggio dei termini, eventualmente estendendo l'insieme degli atomi in modo opportuno. Infatti, il linguaggio Prolog<sub>0</sub><sup>+</sup> estende Prolog<sub>0</sub> mediante l'introduzione di liste e operatori nelle teste di fatti e regole, nei congiunti e nei termini. Ovviamente, anche i goal da utilizzare con i programmi Prolog<sub>0</sub><sup>+</sup> potranno sfruttare liste e operatori.

Ad esempio, il seguente è un programma scritto in Prolog<sub>0</sub><sup>+</sup> sfruttando le liste:

```

m(H, [H | X]). 
m(H, [Y | R]) :- 
    m(H, R).

```

Il programma può essere utilizzato, ad esempio, con il goal `m(X, [a, b, c]).`

Si noti che l'introduzione di liste e operatori non estende in modo significativo il linguaggio Prolog<sub>0</sub> perché liste e operatori non sono altro che varianti sintattiche del linguaggio dei termini. Viceversa, le due seguenti estensioni introdotte in Prolog<sub>0</sub><sup>+</sup> non si limitano a varianti sintattiche, ma hanno anche un aspetto semantico. La prima di queste estensioni prevede che sia sempre disponibile in Prolog<sub>0</sub><sup>+</sup> un operatore definito dalla tripla  $(700, xfx, =)$ , richiedendo quindi che l'atomo `=` sia presente nell'insieme degli atomi di Prolog<sub>0</sub><sup>+</sup>. La semantica di questo operatore verrà discussa in seguito. La seconda di queste estensioni prevede che l'insieme delle variabili di Prolog<sub>0</sub><sup>+</sup> contenga anche un elemento formato unicamente da un underscore, elemento che è esplicitamente escluso dall'insieme delle variabili di Prolog<sub>0</sub>. Questa variabile, che viene detta *dummy* (o *meta*), ha un significato particolare in quanto si prevede che venga sostituita con una nuova variabile non presente nel programma o nel goal tutte le volte che viene incontrata.

Ad esempio, l'utilizzo della variabile dummy e dell'operatore `=` permette di riscrivere il programma dell'esempio precedente come segue:

```

m(H, [X | _]) :- 
    X = H.
m(H, [_ | R]) :- 
    m(H, R).

```

## 5 Semantica di Prolog<sub>0</sub>

La *sintassi* di Prolog<sub>0</sub> è definita dalla grammatica che viene utilizzata per costruire l'insieme dei programmi partendo dall'insieme degli atomi  $A$  e dall'insieme delle variabili  $V$ . La *semantica* di Prolog<sub>0</sub> viene definita nel seguito esplicitando formalmente la computazione che si ottiene da un programma utilizzato insieme ad un opportuno goal.

La sintassi di Prolog<sub>0</sub><sup>+</sup> viene definita mediante semplici regole sintattiche che permettono di rimuovere liste, operatori e variabili dummy. Quindi, la semantica di Prolog<sub>0</sub><sup>+</sup> può essere espressa, ad eccezione che per la semantica dell'operatore  $=$ , sfruttando la semantica dei programmi e dei relativi goal scritti in Prolog<sub>0</sub> dopo la rimozione di operatori, liste e variabili dummy. Quindi, dopo aver definito in modo formale la semantica di Prolog<sub>0</sub> sarà sufficiente definire la semantica dell'operatore  $=$  per completare la semantica di Prolog<sub>0</sub><sup>+</sup>.

Dato un programma  $\pi$  scritto in Prolog<sub>0</sub> e un goal  $\nu$ , la definizione formale della semantica del programma  $\pi$  quando viene utilizzato per *soddisfare* il goal  $\nu$  viene descritta mediante una funzione non deterministica  $\sigma^\pi$ . La funzione  $\sigma^\pi$  riceve come unico argomento il goal  $\nu$  e produce, in modo non deterministico, zero o più sostituzioni che coinvolgono le variabili di  $\nu$ . Se almeno una sostituzione viene calcolata, anche se vuota, allora il goal si dice *soddisfacibile*. Viceversa, il goal si dice *insoddisfacibile*. Si noti, però, che non è possibile garantire che il calcolo di  $\sigma^\pi(\nu)$  termini per qualsiasi coppia programma  $\pi$  e goal  $\nu$  e, quindi, è possibile che in alcune situazioni l'enumerazione dei risultati del calcolo di  $\sigma^\pi(\nu)$  possa non essere completata.

Prima di dettagliare  $\sigma^\pi$  è però necessario introdurre alcune funzioni che verranno utilizzate nella definizione di  $\sigma^\pi$ . Dato un programma  $\pi$  scritto in Prolog<sub>0</sub>, la funzione  $head_P(\pi)$  produce la prima clausola definita del programma, che è sempre presente visto che la grammatica di Prolog<sub>0</sub> non ammette programmi vuoti. In più, la funzione  $rest_P(\pi)$  produce  $\pi$  privato della prima clausola definita, se il programma contiene almeno due clausole definite, oppure  $\perp$  se il programma è formato unicamente da una clausola definita.

In modo simile, dato un goal  $\nu$  scritto in Prolog<sub>0</sub>, la funzione  $head_G(\nu)$  produce il primo congiunto di  $\nu$ , che è sempre presente visto che la grammatica delle regole di Prolog<sub>0</sub> non ammette regole senza corpo. In più, la funzione  $rest_G(\nu)$  produce  $\perp$  se il goal è formato unicamente da un congiunto, oppure la sequenza dei congiunti di  $\nu$  a cui è stato rimosso il primo congiunto, separando i congiunti con delle virgolette.

Infine, la funzione non deterministica  $(.)'$ , data una clausola definita, produce una nuova clausola definita, detta sua *variante fresh* (*o fresca*), ottenuta applicando una sostituzione che associa ad ogni variabile della clausola definita una nuova variabile non ancora utilizzata nella computazione. Quindi, ad esempio, una variante fresh del fatto  $h(X, Y, X)$ . è

$$(h(X, Y, X).)' = h(X1, Y1, X1). \quad (13)$$

nell'ipotesi che le variabili  $X1$  e  $Y1$  non siano ancora state usate nella computazione. In modo simile, una variante fresh della regola  $h(X, Y) :- f(X), g(Y, X)$ . è

$$(h(X, Y) :- f(X), g(Y, X).)' = h(X1, Y1) :- f(X1), g(Y1, X1). \quad (14)$$

sempre nell'ipotesi che le variabili  $X1$  e  $Y1$  non siano ancora state usate nella computazione. Si noti che la funzione  $(.)'$  è una funzione non deterministica perché la scelta delle nuove variabili è del tutto arbitraria, purché queste non siano ancora state utilizzate nella computazione. In più, si noti che la produzione della variante fresh di una regola richiede di memorizzare le variabili utilizzate durante la computazione. Però, per non appesantire troppo la notazione, si preferisce lasciare implicito l'insieme delle variabili utilizzate durante una computazione senza, comunque, perdere di generalità o di correttezza.

Dato un programma  $\pi$  scritto in Prolog<sub>0</sub> e un goal  $\nu$ , la funzione non deterministica  $\sigma^\pi$  può essere esplicitata come segue:

$$\sigma^\pi(\nu) = \begin{cases} \sigma_G^\pi(g, \pi) & \text{se } g = \text{head}_G(\nu) \wedge \perp = \text{rest}_G(\nu) \\ \theta \circ \sigma^\pi(:- r\theta.) & \text{se } g = \text{head}_G(\nu) \wedge r = \text{rest}_G(\nu) \wedge r \neq \perp \wedge \theta = \sigma_G^\pi(g, \pi) \end{cases}$$

Quindi, il calcolo di  $\sigma^\pi(\nu)$  si riduce a due casi non deterministici disgiunti. Se il goal è formato da un unico congiunto, allora viene usata la funzione non deterministica  $\sigma_G^\pi$  descritta nel seguito. Questa funzione, se è applicabile agli specifici argomenti e se termina, produce la sostituzione cercata. La stessa funzione viene utilizzata per produrre una sostituzione nel caso in cui il goal sia formato da almeno due congiunti. In questo caso, la sostituzione ottenuta da  $\sigma_G^\pi$  viene applicata alla parte non ancora elaborata del goal per applicare poi ricorsivamente  $\sigma^\pi$ . Si noti che la natura non deterministica di  $\sigma^\pi$  deriva unicamente dalla natura non deterministica di  $\sigma_G^\pi$  perché la sintassi del linguaggio garantisce che i due casi considerati nella definizione di  $\sigma^\pi$  siano disgiunti.

La funzione  $\sigma_G^\pi$  riceve come primo argomento un congiunto e come secondo argomento un programma, cioè una sequenza di clausole definite. Se effettivamente applicabile agli argomenti e nell'ipotesi che termini,  $\sigma_G^\pi$  produce in modo non deterministico delle sostituzioni. In particolare, la funzione  $\sigma_G^\pi$  è definita come segue:

$$\sigma_G^\pi(g, \pi) = \begin{cases} \sigma_F(g, c) & \text{se } c = \text{head}_P(\pi) \wedge c = h. \\ \sigma_R^\pi(g, c) & \text{se } c = \text{head}_P(\pi) \wedge c = h :- b. \\ \sigma_G^\pi(g, r) & \text{se } r = \text{rest}_P(\pi) \wedge r \neq \perp \end{cases} \quad (15)$$

La funzione è definita da tre casi non deterministici. Il primo descrive il comportamento della funzione se la prima clausola definita è un fatto. Il secondo descrive il comportamento della funzione se la prima clausola definita è una regola. L'ultimo caso descrive il comportamento della funzione se non sono state elaborate tutte le clausole definite del programma. Quindi, i due casi più interessanti sono il primo e il secondo perché il terzo si limita a proseguire il calcolo nel tentativo di coinvolgere tutte le clausole definite del programma.

La funzione  $\sigma_F$  riceve come primo argomento un congiunto di un goal e come secondo argomento un fatto è definita come segue:

$$\sigma_F(g, f) = \theta \quad \text{se } f' = h'. \wedge \theta = \text{mgu}(\{g, h'\}) \wedge \theta \neq \perp \quad (16)$$

Quindi, se è possibile unificare la testa del fatto con il goal, allora il MGU ottenuto dal goal e dalla testa di una variante fresh del fatto ricevuto come argomento è il risultato cercato.

In modo simile, la funzione  $\sigma_R^\pi$  riceve come primo argomento un congiunto di un goal e come secondo argomento una regola è definita come segue:

$$\sigma_R^\pi(g, r) = \theta \circ \sigma^\pi(:- b'\theta.) \quad \text{se } r' = h' : b'. \wedge \theta = \text{mgu}(\{g, h'\}) \wedge \theta \neq \perp \quad (17)$$

Quindi, se è possibile unificare la testa della regola con il goal, allora il MGU ottenuto dal goal e dalla testa di una variante fresh della regola ricevuta come argomento è una parte del risultato cercato. Infatti, una volta identificato il MGU, lo si applica al corpo della variante fresh della regola utilizzata in modo da ottenere un nuovo goal da soddisfare mediante l'applicazione di  $\sigma^\pi$ .

Come già chiarito, la semantica di Prolog<sub>0</sub><sup>+</sup> è ottenuta mediante la semantica di Prolog<sub>0</sub> dopo aver rimosso liste, operatori e variabili dummy in modo puramente sintattico. Per completare la semantica di Prolog<sub>0</sub><sup>+</sup> è però necessario definire una semantica per l'operatore  $=$ , che è sempre ritenuto disponibile. La semantica di questo operatore può essere definita prevedendo che venga aggiunto un *prologo* ad ogni programma Prolog<sub>0</sub><sup>+</sup>. Questo prologo contiene la definizione del comportamento dell'operatore  $=$  mediante il fatto  $X = X$ . che, appunto, stabilisce che l'operatore  $=$  faccia riferimento all'uguaglianza sintattica.

**Esempio 20.** Dato il programma  $\pi = p(a). p(b).$ , si vuole verificare se il goal  $\nu = :- p(b).$  è soddisfacibile. In particolare,

$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma_G^\pi(p(b), \pi) \stackrel{1}{=} \sigma_F(p(b), p(a).) = \times \quad \text{mgu}(\{p(b), p(a)\}) = \perp$$

$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma_G^\pi(p(b), \pi) \stackrel{2}{=} \times \quad \text{non applicabile}$$

$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma_G^\pi(p(b), \pi) \stackrel{3}{=} \sigma_G^\pi(p(b), p(b).) \stackrel{1}{=} \sigma_F(p(b), p(b).) = \text{mgu}(\{p(b), p(b)\}) = \emptyset$$

quindi il goal è soddisfacibile perché è stata trovata una sostituzione.

**Esempio 21.** Dato il programma  $\pi = p(a). p(b).$ , si vuole verificare se il goal  $\nu = :- p(c).$  è soddisfacibile. In particolare,

$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma_G^\pi(p(c), \pi) \stackrel{1}{=} \sigma_F(p(c), p(a).) = \times \quad \text{mgu}(\{p(c), p(a)\}) = \perp$$

$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma_G^\pi(p(c), \pi) \stackrel{2}{=} \times \quad \text{non applicabile}$$

$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma_G^\pi(p(c), \pi) \stackrel{3}{=} \sigma_G^\pi(p(c), p(b).) \stackrel{1}{=} \sigma_F(p(c), p(b).) = \times \quad \text{mgu}(\{p(c), p(b)\}) = \perp$$

$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma_G^\pi(p(c), \pi) \stackrel{3}{=} \sigma_G^\pi(p(c), p(b).) \stackrel{2,3}{=} \times \quad \text{non applicabili}$$

$$\sigma^\pi(\nu) \stackrel{2}{=} \times \quad \text{non applicabile}$$

quindi il goal non è soddisfacibile.

**Esempio 22.** Dato il programma  $\pi = p(a). p(b).$ , si vuole verificare se il goal  $\nu = :- p(X).$  è soddisfacibile. In particolare,

$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma_G^\pi(p(X), \pi) \stackrel{1}{=} \sigma_F(p(X), p(a).) = \text{mgu}(\{p(X), p(a)\}) = \{a/X\} \quad (18)$$

quindi il goal è soddisfacibile. Continuando con l'applicazione non deterministica di  $\sigma^\pi$  si ottiene:

$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma_G^\pi(p(X), \pi) \stackrel{2}{=} \times \quad \text{non applicabile}$$

$$\sigma^\pi(\nu) \stackrel{1}{=} \sigma_G^\pi(p(X), \pi) \stackrel{3}{=} \sigma_G(p(X), p(b).) \stackrel{1}{=} \sigma_F(p(X), p(b).) = \{b/X\}$$

e quindi il goal è soddisfacibile anche dalla sostituzione  $\{b/X\}$ .

**Esempio 23.** Dato il programma  $\pi = p(a). q(X) :- p(X).$ , si vuole verificare se il goal  $\nu = :- q(X)$ . è soddisfacibile. In particolare,

$$\begin{aligned}\sigma^\pi(\nu) &\stackrel{1}{=} \sigma_G^\pi(q(X), \pi) \stackrel{1}{=} \sigma_F^\pi(q(X), p(a).) = \times \quad mgu(\{q(X), p(a)\}) = \perp \\ \sigma^\pi(\nu) &\stackrel{1}{=} \sigma_G^\pi(q(X), \pi) \stackrel{2}{=} \times \quad \text{non applicabile} \\ \sigma^\pi(\nu) &\stackrel{1}{=} \sigma_G^\pi(q(X), \pi) \stackrel{3}{=} \sigma_G^\pi(q(X), q(X) :- p(X).) \stackrel{1}{=} \times \quad \text{non applicabile} \\ \sigma^\pi(\nu) &\stackrel{1}{=} \sigma_G^\pi(q(X), \pi) \stackrel{3}{=} \sigma_G^\pi(q(X), q(X) :- p(X).) \stackrel{2}{=} \sigma_R^\pi(q(X), q(X) :- p(X).)\end{aligned}$$

ma

$$\sigma_R^\pi(q(X), q(X) :- p(X).) = \{X/X1\} \circ \sigma^\pi(:- p(X1)\{X/X1\}.) = \{X/X1\} \circ \sigma^\pi(:- p(X).)$$

e

$$\sigma^\pi(:- p(X).) \stackrel{1}{=} \sigma_G^\pi(p(X), \pi) \stackrel{1}{=} \sigma_F(p(X), p(a).) = mgu(\{p(X), p(a)\}) = \{a/X\}$$

e quindi il goal è soddisfacibile perché è stata trovata la sostituzione  $\{a/X1, a/X\}$ .

**Esempio 24.** Dato il programma  $\pi = q(X) :- q(X).$ , si vuole verificare se il goal  $\nu = :- q(X)$ . è soddisfacibile. In particolare,

$$\begin{aligned}\sigma^\pi(\nu) &\stackrel{1}{=} \sigma_G^\pi(q(X), \pi) \stackrel{1}{=} \times \quad \text{non applicabile} \\ \sigma^\pi(\nu) &\stackrel{1}{=} \sigma_G^\pi(q(X), \pi) \stackrel{2}{=} \sigma_R^\pi(q(X), q(X) :- q(X).)\end{aligned}$$

ma

$$\sigma_R^\pi(q(X), q(X) :- q(X).) = \{X/X1\} \circ \sigma^\pi(:- q(X1)\{X/X1\}.) = \{X/X1\} \circ \sigma^\pi(:- q(X).)$$

e quindi è dimostrato che la computazione di  $\sigma^\pi$  non termina perché viene richiesto di soddisfare ciclicamente il goal  $:- q(X)$ . indefinitamente.

## 6 Il Linguaggio Prolog!

Il linguaggio Prolog<sup>+</sup>, e quindi il linguaggio Prolog<sub>0</sub>, è sufficiente per realizzare programmi che possono essere utilizzati per risolvere problemi interessanti. Ad esempio, Prolog<sup>+</sup> è stato utilizzato per realizzare un programma per descrivere un grafo diretto aciclico che è poi stato usato per soddisfare vari goal interessanti. Però, Prolog<sup>+</sup> non è ancora sufficiente perché non offre alcun modo per bloccare le scelte non deterministiche. Si consideri, ad esempio, il grafo diretto (non aciclico) riportato in Figura 2. Il grafo può essere descritto in un programma Prolog<sup>+</sup> come segue:

```
arc(a, a).
arc(a, b).
arc(b, c).
arc(c, a).

path(X, X, [X]).
path(X, Y, [X | R]) :-
    arc(X, Z),
    path(Z, Y, R).
```

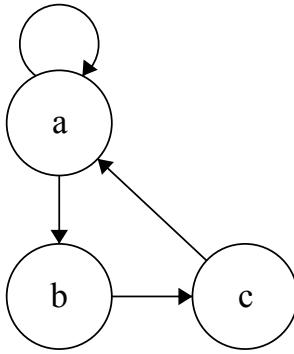


Figura 2: Un semplice grafo diretto con un autoanello e un ciclo.

dove le regole che descrivono `path` sono state introdotte per calcolare il percorso tra due nodi. Quindi, il seguente goal può essere utilizzato per elencare i percorsi nel grafo tra il nodo `a` e il nodo `c`:

`path(a, c, P).`

Però, la soluzione proposta presenta due problemi:

1. La presenza dell'autoanello sul nodo `a` innesca immediatamente un ciclo infinito che non permette di ottenere alcune soluzioni al goal; e
2. Anche dopo aver rimosso l'autoanello, ad esempio rimuovendo o commentando il primo fatto, il goal viene soddisfatto infinite volte nonostante i percorsi prodotti siano solo ripetizioni cicliche dello stesso percorso `[a, b, c]`.

Entrambi questi problemi potrebbero essere risolti richiedendo che un percorso non possa contenere due volte lo stesso nodo. Quindi, in presenza di un ipotetico operatore di negazione, sarebbe sufficiente verificare che un nodo non sia presente nel percorso corrente e, solo in questo caso, aggiungerlo.

Però, la negazione è uno strumento complesso da realizzare nell'ambito della programmazione logica soprattutto a causa della presenza di *variabili libere*, che sono variabili non ancora sostituite da termini durante la computazione. Quindi, anziché estendere Prolog<sub>0</sub><sup>+</sup> con una negazione complessa da realizzare e da utilizzare, è più utile estendere Prolog<sub>0</sub><sup>+</sup> con uno strumento in grado di inibire alcuni rami della computazione non deterministica.

Il linguaggio Prolog<sub>!</sub> è il linguaggio Prolog<sub>0</sub> arricchito di un atomo che può essere utilizzato come un congiunto e, quindi, nei goal e nei corpi delle regole. Questo operatore viene indicato con il simbolo `!` (letto *cut* o *simbolo di cut*) e viene usato solo con arità zero. Quindi, la produzione che descrive i congiunti in Prolog<sub>!</sub> diventa:

$$\text{Conjunct} \rightarrow \text{Atom} \mid \text{Atom}(\text{Arguments}) \mid ! \quad (19)$$

Lo strumento del *cut* viene spesso criticato perché è in grado di introdurre dei comportamenti inattesi se non viene usato con estrema cautela. Quindi, conviene subito sottolineare il fatto che l'uso del *cut* deve essere limitato allo stretto necessario.

Informalmente, quando viene incontrato un cut nel corpo di una regola  $h :- b$ . vengono scartate:

1. Tutte le scelte non deterministiche attivate da quando l'esecutore ha iniziato a soddisfare la regola  $e$ , quindi, tutte le scelte non deterministiche legate ai congiunti di  $b$  che precedono il cut; e
2. Tutte le scelte non deterministiche attivate per cercare un fatto o una regola con testa che unifichi  $h$  e, quindi, tutte le scelte non deterministiche legate ai fatti e alle regole che seguono la regola corrente nel testo del programma e che hanno teste che unificano con  $h$ .

Si noti che queste scelte non deterministiche vengono scartate anche se la regola in cui è presente il cut fallisce. Quindi, contrariamente agli altri effetti legati al soddisfacimento dei goal, gli effetti di un cut non vengono inibiti da un fallimento.

Ad esempio, il cut presente nel seguente programma:

```
p(a).
p(X) :- q(X), !, r(X).
p(b).
p(c).

q(y).
q(z).

r(y).
r(z).
```

garantisce che, una volta incontrato, vengano scartate le scelte non deterministiche legate a  $q(X)$ ,  $p(b)$  e  $p(c)$ . Quindi, il goal  $:- p(W)$ . viene soddisfatto unicamente dalle sostituzioni  $W=a$  e  $W=y$  perché  $q(z)$ ,  $p(b)$  e  $p(c)$  vengono inibite dal cut.

In modo del tutto analogo, quando viene incontrato un cut in un goal, vengono scartate tutte le scelte non deterministiche attivate da quando l'esecutore ha iniziato a soddisfare il goal  $e$ , quindi, tutte le scelte non deterministiche legate ai congiunti del goal che precedono il cut. Quindi, il goal  $:- p(W)$ ,  $!$ . applicato al programma precedente viene soddisfatto unicamente dalla sostituzione  $W=a$  perché la sostituzione  $W=y$  viene inibita dal cut.

Normalmente, l'introduzione del cut viene accompagnata dai seguenti fatti e regole che vengono aggiunte al prologo utilizzato implicitamente con i programmi scritti in Prolog<sup>+</sup>:

```
true.

fail :-
    a = b.

X \= X :-
    !,
    fail.
X \= Y.
```

Il primo fatto introduce `true`, che serve per esprimere dei goal che non falliscono mai. La seconda regola serve per introdurre `fail`, che è un goal che fallisce sempre. Infine, il terzo predicato binario, espresso mediante un operatore infisso, serve per definire cosa significhi diverso in Prolog<sup>+</sup> dicendo che due termini sono diversi tra loro solo se non possono essere unificati. Quindi, oltre a variabili dummy, liste e operatori, Prolog<sup>+</sup> aggiunge a Prolog<sub>I</sub> gli operatori = e \=, il fatto `true` e le regole per definire `fail`.

L'introduzione del cut permette di risolvere i problemi dell'esempio discusso in precedenza e relativo ad un grafo diretto non aciclico. In particolare, con la seguente soluzione è possibile impedire che un percorso contenga più volte lo stesso nodo:

```

node(a).
node(b).
node(c).

arc(a, a).
arc(a, b).
arc(b, c).
arc(c, a).

path(X, Y, P) :-
    path1(X, Y, [], P).

path1(_, X, V, _) :-
    member(X, V),
    !,
    fail.
path1(X, X, V, [X | V]).
path1(Y, X, V, R) :-
    arc(Z, X),
    path1(Y, Z, [X | V], R).

```

L'introduzione del cut in Prolog<sub>I</sub> e, equivalentemente nel linguaggio esteso Prolog<sup>+</sup>, permette di modificare la computazione svolta dall'esecutore e, quindi, in generale, cambia l'insieme delle sostituzioni che risolvono un goal. Però, alle volte, l'utilizzo del cut non ha effetto sull'insieme delle sostituzioni che soddisfano un goal, ma si limita, eventualmente, solo a cambiare l'ordine con cui le soluzioni al goal vengono calcolate o il numero di volte in cui una soluzione viene prodotta. Dato un programma e un goal, se il cut viene utilizzato per modificare il programma o il goal senza però modificare l'insieme delle soluzioni al goal, allora viene detto *green cut*. Viceversa, in tutti gli altri casi, si parla di *red cut*. I green cut vengono utilizzati solo per migliorare le prestazioni dell'esecutore nel risolvere alcuni goal e, quindi, non cambiano in modo sostanziale la semantica del programma e del goal. Viceversa, i red cut vengono utilizzati per ottenere programmi o goal che si comportino in modo sostanzialmente diverso da quelli che non usano il cut. Normalmente, i green cut vengono considerati utili e non particolarmente pericolosi dal punto di vista della correttezza dei programmi realizzati. Viceversa, i red cut sono considerati uno strumento che andrebbe utilizzato poco e con estrema cautela.

L'introduzione del cut permette di spiegare perché la negazione è ritenuta problematica nella programmazione logica. Si consideri, ad esempio, il problema di stabilire se un elemento non è membro di una lista. Una possibile soluzione scritta in Prolog<sup>†</sup> è la seguente:

```
nonmember(_, []).
nonmember(X, [X | _]) :- !,
    fail.
nonmember(X, [_ | R]) :- nonmember(X, R).
```

Però, questa soluzione, come tutte le soluzioni che coinvolgono la negazione, ha dei comportamenti anomali in presenza di variabili libere. Ad esempio:

```
:-
    nonmember(R, [a, b, c]).
    false

:- nonmember(z, [a, b, R]).
    false
```

Si noti che non è possibile ovviare a questi problemi anche utilizzando il linguaggio Prolog. Infatti, la negazione offerta da Prolog viene detta *negazione per fallimento* e viene realizzata soddisfacendo un goal negato, per fallimento, solo quando il relativo goal affermato non è soddisfacibile. Quindi, l'utilizzo opportuno del cut consente sempre di evitare l'utilizzo della negazione per fallimento ed è preferibile perché le situazioni anomale introdotte dal cut risultano più evidenti.

La semantica di Prolog<sub>!</sub> può essere introdotta formalmente andando ad estendere la semantica di Prolog<sub>0</sub>. L'estensione richiede di definire la funzione non deterministica  $\sigma^\pi$  che definisce la semantica di Prolog<sub>!</sub> estendendo quella di Prolog<sub>0</sub> per produrre come risultato una coppia  $(\theta, \gamma)$  in cui  $\theta$  è la sostituzione che rappresenta il risultato della computazione mentre  $\gamma$  viene utilizzata per gestire il cut. In particolare,  $\gamma$  può valere  $!$  o  $\perp$  e, se vale,  $!$  significa che la computazione è stata alterata dall'utilizzo di un cut, mentre se vale  $\perp$  significa che la computazione non ha coinvolto alcun cut. Naturalmente, questo richiede di modificare anche la funzione  $\sigma_G$  che definisce il comportamento dell'esecutore durante il soddisfacimento dei goal. Questa funzione estende quella di Prolog<sub>0</sub> prevedendo la presenza di un nuovo tipo di congiunto. Questo nuovo tipo di congiunto, il cut, non fallisce mai, non produce una sostituzione ma solo un valore per  $\gamma$  e la funzione è fatta in modo che gli effetti della presenza di un cut si propaghino anche se i goal che seguono il cut falliscono. Quest'ultima parte è particolarmente insidiosa e complica la funzione in modo sensibile, ma è necessaria per permettere di realizzare, ad esempio, i predicati  $\backslash=$  e **nonmember**. Infatti, entrambi questi predicati utilizzano il goal  $!, \text{fail}$  per garantire che non venga preso un percorso non deterministico in cui viene richiesto che i goal falliscano. Lo stesso comportamento viene tenuto dal predicato **path** dell'esempio discusso in precedenza per garantire che, se un nodo è già presente in un percorso, la costruzione del percorso venga interrotta.

## 7 Prolog e Linguaggio Naturale

I primi esperimenti realizzati da *Alain Colmerauer*, che hanno poi dato vita alla programmazione logica e a Prolog, avevano come obiettivo la realizzazione di strumenti per l'*elaborazione (automatica) del linguaggio naturale* (*Natural Language Processing* o *NLP*). Le tecniche introdotte con questi esperimenti sono ancora un valido approccio alla maggior parte dei problemi di NLP e, quindi, ancora oggi il legame tra Prolog e NLP è molto stretto. Nonostante questo legame, però, gli approcci più recenti al NLP hanno anche esplorato strade diverse, riconoscendo il legame forte tra NLP e apprendimento automatico. Quindi, oggi, i problemi di NLP possono essere affrontati con *approcci simbolici*, principalmente basati su Prolog, e *approcci sub-simbolici*, principalmente basati su reti neurali.

Seguendo le linee di ricerca rese popolari da *A. Noam Chomsky*, è sicuramente possibile dire che uno dei problemi fondamentali del NLP è quello di spezzare una *frase* in parti costitutive che permettano di verificarne la correttezza e che permettano di attribuirle un significato. Questo problema viene normalmente chiamato di *identificazione delle parti del discorso* (o *Part Of Speech, POS*).

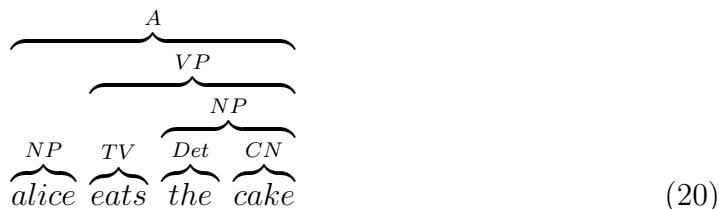
Ognuna di queste parti costitutive di una frase viene chiamata *sintagma* (o *phrase*) ed è una sezione continua di parole di una frase. I sintagmi vengono classificati in base alla tipologia della prima parola *rilevante* che contengono, che viene normalmente chiamata *testa* del sintagma. Si noti che i sintagmi di una frase non sono necessariamente disgiunti. Anzi, normalmente, un sintagma contiene altri sintagmi in modo che una frase possa essere associata ad un *albero sintattico* in cui i nodi interni sono etichettati con nomi di sintagmi e le foglie sono etichettate con le singole parole della frase.

Facendo riferimento alla lingua inglese, e semplificando un poco alcuni dettagli di interesse rilevante prettamente per uno studio avanzato della *linguistica computazionale*, è possibile identificare i seguenti tipi di sintagmi:

1. Un *sintagma nominale* (o *Noun Phrase, NP*) è un sintagma in cui la testa è un *nome* (*comune* o *proprio*) o un *pronomе*;
2. Un *sintagma verbale* (o *Verb Phrase, VP*) è un sintagma in cui la testa è un *verbo*; e
3. Un *sintagma preposizionale* (o *Prepositional phrase, o PP*) è un sintagma in cui la testa è una *preposizione*.

Si noti che gli stessi tipi di sintagmi sono presenti in varie altre lingue, tra le quali l'italiano. In più, si noti che le abbreviazioni e gli acronimi adottati sono ormai di uso comune e consolidato.

Quindi, ad esempio, la frase *alice eats the cake* può essere ricondotta al seguente albero sintattico:



dove conviene notare che:

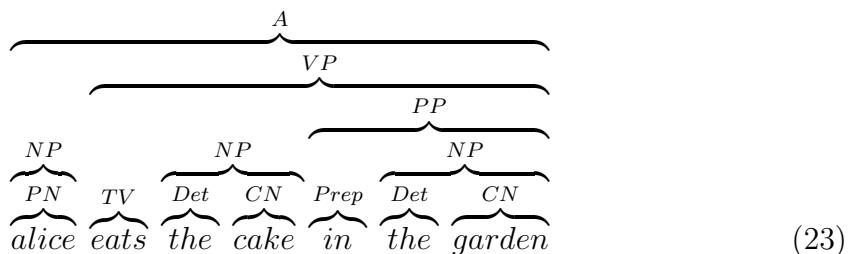
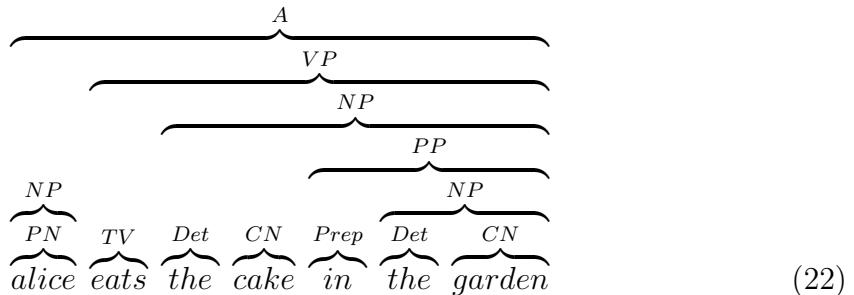
- La lettera *A* è stata usata perché la frase è un'*asserzione*;
- L'acronimo *TV* è stato usato perché *eats* è una voce del *verbo transitivo* (*Transitive Verb*, *TV*) *to eat*;
- L'abbreviazione *Det* è stata usata perché *the* è un *determinante* (o *determiner*) come *a*, *this*, *that* e altri; e
- L'acronimo *CN* (da *Common Name*) è usato perché *cake* è un nome comune.

Per risolvere il problema dell'identificazione della parti del discorso è possibile utilizzare le grammatiche formali. Ad esempio, la seguente grammatica non contestuale in formato BNF descrive delle semplici asserzioni in un inglese semplificato:

$$\begin{aligned}
 A &\rightarrow NP VP \\
 NP &\rightarrow PN \mid Det CN PP^* \\
 VP &\rightarrow IV PP^* \mid TV NP PP^* \\
 PP &\rightarrow Prep NP \\
 Det &\rightarrow \{\text{the}\} \\
 Prep &\rightarrow \{\text{in, near}\} \\
 PN &\rightarrow \{\text{alice, bob, london}\} \\
 CN &\rightarrow \{\text{book, cake, garden, house, kid, lake}\} \\
 TV &\rightarrow \{\text{eats, reads}\} \\
 IV &\rightarrow \{\text{lives, runs}\}
 \end{aligned} \tag{21}$$

dove *IV* sta per *verbo intransitivo* (o *Intransitive Verb*, *IV*) e *PN* sta per *nome proprio* (o *Proper Name*, *PN*).

Si noti subito che, come sempre accade quando si lavora nell'ambito del NLP, la grammatica precedente è ambigua perché, ad esempio, ammette più derivazioni per la frase *alice eats the cake in the garden*:



La prima derivazione significa che Alice mangia la torta che è in giardino e, mentre la seconda significa che Alice mangia la torta mentre si trova in giardino. Entrambe le derivazioni sono corrette ed esemplificano un'ambiguità estremamente comune nelle lingue tipo la lingua inglese. Queste ambiguità sono intrinseche alla lingua considerata e non possono essere eliminate a meno di non semplificare ancora di più il linguaggio descritto. Comunque, avendo a disposizione uno strumento di calcolo non deterministico, le ambiguità verranno gestite come risultati alternativi e quindi non sarà necessario cercare di eliminarle durante l'analisi delle frasi.

Infine, si noti che la grammatica descrive un linguaggio che include anche frasi senza senso. Ad esempio, *alice reads the garden* oppure *alice eats london* sono frasi previste dalla grammatica. L'identificazione che queste frasi non hanno senso, però, richiederebbe di approfondire la semantica delle frasi analizzate e quindi è un problema che va oltre il problema dell'identificazione delle parti del discorso. Si noti, comunque, che anche un'analisi semantica del linguaggio descritto non sarà in grado di identificare che le frasi menzionate sono insensate a meno di non inserire sufficiente *conoscenza comune (common knowledge)*.

Per realizzare un *analizzatore sintattico* (o *parser*) della grammatica precedente in Prolog sarà sufficiente:

1. Ipotizzare che la frase da analizzare sia fornita come una lista di atomi, ognuno dei quali rappresenti un simbolo terminale della grammatica;
2. Realizzare un predicato con arità due per ogni simbolo non terminale;
3. Realizzare i predicati associati ai simboli non terminali in modo che il primo argomento sia una lista che rappresenti la parte della frase da analizzare e il secondo argomento rappresenti la parte della frase che rimarrà da analizzare.

L'esempio `nlp1.pl` è un semplice parser del linguaggio descritto dalla grammatica dell'inglese estremamente semplificato che è stata considerata in precedenza.

L'utilizzo delle grammatiche non contestuali per risolvere il problema dell'identificazione delle parti del discorso si scontra con la necessità di introdurre molti dettagli linguistici che tendono a fare crescere velocemente il numero di produzioni necessarie per descrivere un linguaggio naturale moderatamente realistico. Quindi, spesso si abbandona lo strumento delle grammatiche non contestuali per utilizzare altri strumenti. Prolog è uno di questi strumenti e, per come verrà utilizzato in seguito, viene spesso reinterpretato mediante una sintassi alternativa detta delle *grammatiche a clausole definite (Definite Clause Grammar, DCG)*. Si noti che l'utilizzo delle DCG non aggiunge nulla all'utilizzo che verrà descritto di Prolog e, quindi, le DCG non verranno ulteriormente approfondite.

Come esempio delle possibilità offerte dall'utilizzo di Prolog per superare i limiti delle grammatiche non contestuali, si vuole estendere la grammatica sviluppata:

1. Introducendo i plurali dei nomi comuni;
2. Aggiungendo la persona (prima, seconda o terza) e il numero (singolare o plurale) ai sintagmi nominali; e
3. Garantendo la concordanza di persona e numero tra soggetto e verbo.

Si noti, comunque, che si sta lavorando solo con il presente indicativo di verbi regolari e quindi è sufficiente trattare in modo specifico solo la terza persona singolare. L'esempio `nlp2.p1` è un'estensione dell'esempio `nlp1.p1` che include le nuove caratteristiche nella grammatica dell'inglese semplificato che si sta considerando.

Partendo da questa seconda grammatica è possibile ottenere una terza grammatica, il cui analizzatore sintattico è disponibile nell'esempio `nlp3.p1`, in cui vengono anche introdotte tre tipi di domande:

1. Domande *chi*, ad esempio, *who eats the cake*
2. Domande *sì/no*, ad esempio, *does alice eat the cake*
3. Domande *dove*, ad esempio, *where does alice eat the cake*

Si noti che, a livello grammaticale, il punto interrogativo finale, così come il punto finale delle asserzioni, non sono considerati. In più, si noti che sono state introdotti nuovi predicati: il predicato *s*, da *sentence*, il predicato *aux*, da *auxiliary* (perché è presente l'*ausiliare do*) e *qwho*, *qyn* e *qwhere* per le domande.

A questo punto è abbastanza semplice estendere l'analizzatore sintattico che è stato realizzato per attribuire un significato alle asserzioni e alle domande espresse nel linguaggio descritto. Nonostante il linguaggio descritto sia molto semplice, si vuole comunque realizzare una *funzione semantica* che attribuisca un *valore semantico* (o *significato*) ad ogni sintagma e che calcoli il valore semantico di una frase componendo i valori semantici dei sintagmi. Questo approccio consente di estendere il linguaggio e la relativa semantica in modo semplice ed efficace. Naturalmente, il linguaggio è così semplice che approcci *ad hoc* sarebbero comunque stati efficaci.

Per la definizione della funzione semantica, conviene ricordare le seguenti note relative all'interpretazione che normalmente viene fatta di un discorso. Un *discorso* è una sequenza di *frasi* eventualmente proferite da più *attori*. Una frase, sia essa un'affermazione o una domanda, fa sempre riferimento ad un *mondo* fatto da *entità*. Gli attori sono alcune di queste entità.

Alcune entità sono identificate univocamente nell'ambito del discorso dai relativi *nomi propri*. Le entità sono divise in classi, non necessariamente disgiunte, che vengono identificate dai *nomi comuni* messi a disposizione dal linguaggio. Le *preposizioni* vengono utilizzate per specificare in modo dettagliato a quali entità di una specifica classe si sta facendo riferimento. In particolare, le preposizioni sono in grado di descrivere delle relazioni tra le entità del mondo.

Alcune entità, non necessariamente diverse dagli attori, sono in grado di compiere delle *azioni* e quindi vengono chiamate *agenti*. Le frasi a cui siamo interessati, siano esse asserzioni o domande, parlano di agenti e delle loro azioni. Le azioni sono identificate univocamente dai *verbi* messi a disposizione dal linguaggio e, eventualmente in modo implicito, fanno riferimento agli agenti che le compiono e che, quindi, prendono il nome di *soggetti* delle azioni. I *verbi transitivi* identificano compiutamente un'azione solo se vengono proferiti insieme ad un'entità che prende il nome di *oggetto* dell'azione. I *verbi intransitivi* non necessitano di questa seconda entità. Le preposizioni vengono utilizzate per specificare in modo dettagliato le caratteristiche di un'azione. In particolare, le preposizioni sono in grado di descrivere delle relazioni tra un'azione e le entità del mondo.

Si noti che, grazie all'approccio adottato e all'attento uso della ricorsione nella scrittura del programma in Prolog, i predicati realizzati sono *invertibili* e quindi possono essere utilizzati per:

1. Verificare che una frase è corretta e calcolarne il relativo valore semantico; e
2. Ottenere una frase che abbia un valore semantico noto.

Si noti che, per come è scritto il programma in Prolog, la verifica della correttezza di una frase e il calcolo del relativo valore semantico terminano sempre, anche nel caso di frasi che non rispettano la grammatica. Viceversa, gli stessi predicati, se utilizzati con valori semanticici non riconducibili a frasi del linguaggio, spesso non terminano. Quindi, è importante garantire che i valori semanticici utilizzati per costruire frasi siano effettivamente riconducibili a frasi del linguaggio.

Partendo da quanto realizzato è semplice scrivere un programma che si comporti come un *chatbot* in grado di dialogare con l'utente utilizzando il linguaggio descritto. Il chatbot ha a disposizione una *base di conoscenza* (*knowledge base*) inizialmente vuota in cui vengono memorizzate tutte le asserzioni dell'utente. Si noti che ogni asserzione può avere più valori semanticici perché la grammatica è ambigua e il chatbot si limita a scegliere il primo dei valori semanticici possibili. Quando l'utente pone una domanda, il chatbot cerca nella sua base di conoscenza tutte le asserzioni che possano essere utilizzate per rispondere alla domanda e, in modo non deterministico, le utilizza tutte per rispondere all'utente.

## 8 Problemi di Pianificazione

Dato un *agente* in grado di compiere *azioni* nell'*ambiente* in cui è immerso, un *problema di pianificazione* riguarda la scelta di un *piano* di azioni che l'agente potrà svolgere per portare il *mondo* (ambiente e agente) dallo *stato attuale* in uno degli stati di *goal* che l'agente cerca di raggiungere. Per semplicità, la trattazione seguente è limitata a problemi di pianificazione simili a quelli supportati dallo *Stanford Research Institute Problem Solver (STRIPS)*, che rappresentano un particolare tipo di *problemi di pianificazione proposizionale*. In particolare, è possibile definire un problema di pianificazione come una quintupla:

$$\langle P, A, I, G, \tilde{G} \rangle \tag{24}$$

dove:

1.  $P \neq \emptyset$  è un insieme finito di *proposizioni* (della logica proposizionale);
2.  $A \neq \emptyset$  è un insieme finito di *azioni*;
3.  $I \neq \emptyset$  è un sottoinsieme finito di  $P$  detto *stato iniziale* (del mondo);
4.  $G \neq \emptyset$  è un sottoinsieme finito di  $P$  detto *goal asserito*; e
5.  $\tilde{G} \neq \emptyset$  è un sottoinsieme finito di  $P$  detto *goal negato*.

In ogni istante, il mondo è caratterizzato da uno *stato* che viene descritto da un sottoinsieme di  $P$  che contiene tutte e sole le proposizioni che risultano vere nello stato. Quindi, ogni proposizione non contenuta nella descrizione di uno stato risulta falsa nello stato stesso. In particolare,  $I$  contiene tutte e sole le proposizioni ritenute vere nello stato iniziale del problema di pianificazione. Si noti, però, che  $G$  e  $\tilde{G}$  non sono descrizioni di uno stato, ma sono, rispettivamente, l'insieme delle proposizioni che devono essere vere e quelle che devono essere false in uno stato che possa essere considerato uno *stato di goal* del problema di pianificazione. Una soluzione del problema di pianificazione è una sequenza finita di azioni che, una volta eseguite dall'agente, portino il mondo dallo stato iniziale  $I$  ad uno stato di goal  $F$  che rispetti i requisiti espressi mediante  $G$  e  $\tilde{G}$  e, quindi, tale che:

$$G \subseteq F \quad \wedge \quad \tilde{G} \cap F = \emptyset \quad (25)$$

Ogni elemento di  $A$  è un'azione ed è una quintupla:

$$\langle n, R, \tilde{R}, T, \tilde{T} \rangle \quad (26)$$

dove:

1.  $n$  è un simbolo detto *nome (o identificativo) dell'azione* e determina univocamente ogni elemento di  $A$ ;
2.  $R$  è un sottoinsieme di  $P$  detto *precondizione asserita*;
3.  $\tilde{R}$  è un sottoinsieme di  $P$  detto *precondizione negata* tale che  $R \cap \tilde{R} = \emptyset$ ;
4.  $T$  è un sottoinsieme di  $P$  detto *postcondizione asserita*; e
5.  $\tilde{T}$  è un sottoinsieme di  $P$  detto *postcondizione negata* tale che  $T \cap \tilde{T} = \emptyset$ .

Un'azione  $a \in A$ , con  $a = \langle n, R, \tilde{R}, T, \tilde{T} \rangle$ , può essere compiuta dall'agente se il mondo si trova in uno stato  $S$  tale che:

$$R \subseteq S \quad \wedge \quad \tilde{R} \cap S = \emptyset \quad (27)$$

Se l'agente compie l'azione  $a$  nello stato  $S$ , che soddisfa le precondizioni precedenti per ipotesi, allora il mondo si porta nello stato  $S'$  tale che:

$$S' = (S \setminus \tilde{T}) \cup T \quad (28)$$

Per risolvere un problema di pianificazione è possibile operare una *ricerca nello spazio degli stati* che parta dallo stato iniziale  $I$  e proceda fino all'identificazione delle soluzioni al problema, se esistono. La ricerca avviene mediante un insieme di *nodi*, ognuno dei quali contiene uno stato  $S$ , l'insieme degli stati attraversati per arrivare dallo stato iniziale fino allo stato  $S$  e la sequenza di azioni che ha permesso di arrivare fino allo stato  $S$ . Si noti che l'insieme degli stati attraversati per arrivare fino allo stato  $S$  è importante perché permette di capire se la ricerca sta per proseguire in uno stato già attraversato e, in questo caso, bloccarla in modo da garantire che la ricerca termini sempre. I nodi vengono organizzati in un *albero di ricerca* che ha

come radice il nodo corrispondente allo stato  $I$ , il quale ha un insieme vuoto di stati già traversati e viene raggiunto con una sequenza di azioni vuota. I nodi figli di un nodo  $N$  sono ottenuti andando a compiere tutte le azioni possibili partendo dal nodo  $N$ . Fissato un percorso nell'albero di ricerca, l'insieme dei nodi costruiti e non ancora scartati viene detto *frangia* (o *fringe*).

In sintesi, la ricerca nello spazio degli stati può essere riassunta come segue:

1. La frangia viene inizializzata con lo stato iniziale  $I$ , un insieme di stati attraversati vuoto e la sequenza di azioni vuota;
2. Se la frangia è vuota, allora è stato provato che non esiste una soluzione al problema di pianificazione;
3. Viceversa, viene prelevato il nodo  $H$  alla testa della frangia e viene indicato con  $S$  il corrispondente stato, con  $L$  l'insieme degli stati attraversati e con  $P$  la corrispondente sequenza di azioni;
4. Se  $S$  è uno stato di goal, allora viene ritornato il piano ottenuto dalla sequenza di azioni  $P$ ;
5. Viceversa, lo stato  $S$  viene *espanso* generando l'insieme degli stati raggiungibili da  $S$  mediante l'applicazione di tutte le azioni disponibili;
6. Per ognuno degli stati  $S'$  raggiungibili da  $S$  mediante l'applicazione di un'azione  $a$ , se  $S' \in L$  allora lo stato viene scartato;
7. Viceversa, viene costruito un nodo relativo ad  $S'$  usando come insieme degli stati attraversati  $L \cup S$  ed estendendo la sequenza di azioni  $P$  con l'azione  $a$ , quindi il nuovo nodo viene aggiunto alla frangia; e
8. Si ritorna al punto 2.

Il metodo così sommariamente descritto ha almeno un punto in cui può essere raffinato. In particolare, non viene deciso in quale posizione della frangia verrà aggiunto un nuovo nodo ottenuto dall'espansione di un altro nodo. Se ogni nuovo nodo viene:

- Aggiunto in fondo alla frangia, allora l'algoritmo di ricerca nello spazio degli stati viene detto *in ampiezza* (*Breadth-First Search* o *BFS*).
- Aggiunto in testa alla frangia, allora l'algoritmo di ricerca nello spazio degli stati viene detto *in profondità* (*Depth-First Search* o *DFS*).
- Aggiunto in modo che gli stati più vicini agli stati di goal siano più vicini alla testa della frangia, allora l'algoritmo di ricerca nello spazio degli stati viene detto *informato*, per distinguerlo dai precedenti due algoritmi che vengono detti *non informati*.

Si noti che tutti i metodi elencati hanno complessità computazionale temporale asintotica di caso pessimo esponenziale di ordine  $O(b^d)$ , dove  $b = |A|$  e  $d$  è il numero di azioni necessarie per raggiungere, nel caso pessimo, uno stato di goal. Però, gli algoritmi hanno caratteristiche diverse da almeno altri due punti di vista:

1. BFS ha complessità computazionale spaziale asintotica di caso pessimo di ordine  $O(b^d)$  e garantisce che, per problemi risolubili, venga ottenuta una soluzione composta dal numero minimo di azioni; ma
2. DFS ha complessità computazionale spaziale asintotica di caso pessimo di ordine  $O(d)$  pur non garantendo, per problemi risolubili, che venga ottenuta una soluzione composta dal numero minimo di azioni.

Per ottenere un buon compromesso tra la BFS e la DFS, di solito si introduce la *ricerca ad approfondimenti successivi (della ricerca in profondità) (Iterative Deepening Depth-First Search, ID-DFS)* che lavora nel seguente modo:

1. Viene inizializzato il *parametro di profondità massima N* al valore uno;
2. Viene applicata la DFS fino alla costruzione di un albero di ricerca con non più di  $N$  livelli;
3. Se viene trovata una soluzione, allora viene immediatamente ritornata, senza necessariamente completare l'albero di ricerca; ma
4. Se non viene trovata una soluzione, allora viene incrementato  $N$  di uno e si ritorna al punto 2.

La ID-DFS ha la complessità computazionale spaziale asintotica di caso pessimo della DFS, ma garantisce che, per problemi risolubili, venga trovata una soluzione composta dal numero minimo di azioni, come accade per la BFS. Si noti che, a causa degli incrementi di  $N$ , la DFS viene applicata ad alberi che condividono la parte più vicina alla radice. Questo prevede che vengano costruiti più volte i nodi dei primi livelli degli alberi, senza però peggiorare la complessità computazionale temporale asintotica. Infatti, il numero di nodi di un albero finito è sempre di ordine  $O(b^d)$ , dove  $b$  è il numero massimo di figli di ogni nodo e  $d$  è la profondità dell'albero. Quindi, l'ordine del numero dei nodi di un albero di profondità finita dipende unicamente dal numero dei nodi nella frangia dell'albero.

Si noti che, normalmente, la ricerca informata richiede che:

1. Il costo sia *definito positivo* e, quindi, che non sia mai nullo o negativo, e che sia *additivo*; e
2. Venga definita una *funzione di costo*  $c = f(N)$  per ogni nodo  $N$  che quantifichi il costo minimo necessario per partire dalla radice, passare per  $N$  e arrivare ad uno stato di goal.

In queste ipotesi, dato un nodo  $N$  ottenuto durante la costruzione dell'albero di ricerca, vale:

$$f(N) = h(N) + g(N) \quad (29)$$

dove  $g(N)$  è il costo necessario per raggiungere il nodo  $N$  partendo da  $I$  e  $h$  è il costo necessario per raggiungere il più vicino stato di goal. Però, spesso, la funzione  $h$  non è disponibile e quindi si ricorre alla funzione  $h^*(N)$  tale che:

$$0 \leq h^*(N) \leq h(N) \quad (30)$$

per ogni nodo  $N$ . La funzione  $h^*$  viene detta *funzione euristica* e se i nodi vengono inseriti nella frangia in senso crescente di  $f^*(N) = h^*(N) + g(N)$ , allora l'algoritmo di ricerca viene chiamato  $A^*$ . Si può dimostrare che  $A^*$  è ottimo nel senso che richiede il numero minimo di espansioni di nodi per ottenere una soluzione a costo minimo, se esiste.

Infine, si noti che la ricerca nello spazio degli stati che è stata descritta viene detta *in avanti* (o *forward chaining*) perché parte dallo stato iniziale fino a raggiungere uno stato di goal. Viceversa, è possibile modificare la ricerca in modo che parta dagli stati di goal e prosegua verso lo stato iniziale facendo, quindi, una ricerca *all'indietro* (o *backward chaining*), sostanzialmente, invertendo il ruolo di precondizioni e postcondizioni.

## 9 Programmazione Logica con Vincoli

La programmazione logica realizzata mediante Prolog<sub>0</sub> e le sue varianti ed estensioni può essere ulteriormente estesa per essere resa più efficace nell'affrontare problemi comuni quali, ad esempio, i problemi di calcolo. La *programmazione logica con vincoli* (*Constraint Logic Programming, CLP*) estende la programmazione logica vista finora in modo molto generale e applicabile a vari contesti. Quindi, conviene subito notare che la programmazione logica con vincoli non viene introdotta unicamente per trattare problemi di calcolo mediante la programmazione logica. Anzi, è ormai così comune che quando si parla di programmazione logica, normalmente, si intende programmazione logica con vincoli. Infatti, tutti i linguaggi di programmazione logica sufficientemente maturi offrono sempre un supporto integrato e ben sviluppato alla programmazione logica con vincoli.

Per estendere Prolog<sub>0</sub> e renderlo un linguaggio di programmazione logica con vincoli è per prima cosa necessario introdurre i linguaggi di vincoli. Un *linguaggio di vincoli* è una ennupla:

$$\langle D, C, A, V, \text{sat}, \text{post}, \text{label} \rangle \quad (31)$$

dove:

1.  $D \neq \emptyset$  è il *dominio* del linguaggio di vincoli;
2.  $C \neq \emptyset$  è un insieme di *simboli di vincolo*;
3.  $A \neq \emptyset$  e  $V \neq \emptyset$ , con  $A \cap V = \emptyset$  sono, rispettivamente, un insieme di atomi e un insieme di variabili utilizzati per costruire termini; e
4.  $\text{sat}$ ,  $\text{post}$  e  $\text{label}$  sono tre funzioni descritte nel seguito.

L'insieme  $D$  può essere utilizzato per costruire dei CSP che abbiano come variabili gli elementi di un sottoinsieme finito di  $V$ , ammettendo anche il caso degenero in cui il sottoinsieme scelto sia vuoto. Quindi, dato un insieme finito di variabili  $V_P \subseteq V$  è possibile costruire un CSP  $\langle V_P, \{D\}, C_P \rangle$  che abbia  $V_P$  come insieme di variabili,  $D$  come dominio di tutte le variabili in  $V_P$  e  $C_P$  come vincoli.

Dato uno di questi CSP, la funzione *sat* dello specifico linguaggio di vincoli è in grado di verificare la soddisfabilità del CSP. In particolare, il risultato dell'applicazione della funzione *sat* ad un CSP è  $\perp$  se la funzione è in grado di stabilire che il CSP è insoddisfacibile. Viceversa, se la funzione *sat* non è in grado di garantire l'insoddisfabilità, allora il risultato è  $\top$ . Quindi, se il risultato della funzione *sat* applicata ad un CSP è  $\perp$ , allora il CSP è sicuramente insoddisfacibile. Viceversa, se il risultato è  $\top$ , allora il CSP potrebbe ancora essere soddisfacibile.

La funzione *post* serve per manipolare i CSP che è possibile costruire con il linguaggio di vincoli a disposizione. Infatti, dato un CSP  $\langle V_P, \{D\}, C_P \rangle$ , vale:

$$post(\langle V_P, \{D\}, C_P \rangle, c(t_1, t_2, \dots, t_n)) = \langle V_P \cup vars(t_1, t_2, \dots, t_n), \{D\}, C_P \cup \{\gamma\} \rangle$$

dove:

1.  $c \in C$  è un simbolo di vincolo a cui è attribuita arită  $n \in \mathbb{N}_+$ ;
2. I termini in  $\{t_i\}_{i=1}^n$  sono costruiti utilizzando  $A$  e  $V$ ; e
3.  $\gamma$  è un vincolo che viene aggiunto al CSP.

In sintesi, la funzione *post* riceve come argomento un CSP e un'espressione che denota un vincolo  $\gamma$  e aggiunge il vincolo, insieme alle sue variabili, al CSP. Normalmente, la funzione *post* propaga il nuovo vincolo in modo da ridurre, se possibile, i valori ammissibili per le variabili. Comunque, non viene richiesto che la *post* propaghi i vincoli e, anche in caso lo facesse, non è previsto che la propagazione identifichi se il nuovo CSP prodotto è insoddisfacibile, anche perché non è previsto che la funzione *post* studi la soddisfabilità dei CSP che estende. Quindi, partendo da un CSP improprio, perché senza variabili e senza vincoli, è possibile costruire nuovi CSP mediante la funzione *post*. Si noti che *post* non è definita per tutte le espressioni che può ricevere come secondo argomento. Infatti, la funzione *post* dello specifico linguaggio di vincoli a disposizione identifica anche quali sono le espressioni per le quali è possibile costruire nuovi CSP aggiungendo vincoli a CSP disponibili.

Infine, la funzione *label* ha un comportamento simile alla funzione *post*, ma è non deterministica. In particolare, dato un CSP  $\langle V_P, \{D\}, C_P \rangle$ , vale:

$$label(\langle V_P, \{D\}, C_P \rangle, x) = \begin{cases} (\langle V_P, \{D\}, C_P \cup \{\gamma\} \rangle, t) & \text{se il CSP ottenuto è} \\ & \text{soddisfacibile per } t \\ \perp & \text{altrimenti} \end{cases} \quad (32)$$

dove:

1.  $x \in V_P$  è una variabile del problema;
2.  $t$  è un termine costruibile utilizzando  $A$  e  $V$  tale che esista un unico  $d \in D$  associato a  $t$ ; e
3.  $\gamma$  è un nuovo vincolo che impone che la variabile  $x$  valga  $d \in D$ , con  $d$  valore univocamente associato a  $t$ .

Quindi, la funzione *label* riceve come primo argomento un CSP e come secondo argomento una variabile del CSP. La funzione costruisce un nuovo CSP e, se è soddisfacibile, lo ritorna insieme ad un termine. In particolare, il nuovo CSP impone che la variabile valga  $d \in D$ , dove  $d$  è univocamente identificato dal termine  $t$  che viene ritornato insieme al CSP.

Si noti che alcuni linguaggi di vincoli forniscono una funzione *label* in grado di verificare la soddisfacibilità del nuovo CSP ottenuto almeno per un valore  $d$ , univocamente associato a  $t$ , se esiste. Quindi, contrariamente a quanto accade utilizzando *sat*, la funzione *label* di questi linguaggi consente di verificare la soddisfacibilità di un CSP andando ad assegnare un valore alla variabile  $x$  che renda il CSP soddisfacibile, se almeno un valore con questa proprietà esiste. Comunque, anche se la funzione *label* non ha questa proprietà, è previsto che la funzione *label* possa sempre stabilire la soddisfacibilità del CSP su cui lavora nel momento in cui viene utilizzata per assegnare l'ultima variabile ancora libera.

Si noti che la funzione *label* è non deterministica perché utilizza in modo non deterministico tutti i valori  $d$ , ed i corrispondenti termini  $t$ , in grado di rendere soddisfacibile il CSP ottenuto. Viceversa, per tutti i valori  $\tilde{d} \in D$  per cui il corrispondente vincolo  $\gamma$  rende insoddisfacibile il CSP vengono scartati.

**Esempio 25.** Si consideri un linguaggio di vincoli con  $D = \mathbb{Z}$  che metta a disposizione un termine numerico per ogni elemento di  $D$ . Sia  $c$  il vincolo definito dalla proprietà:

$$x^2 - 4 \geq 0 \quad (33)$$

Se  $V_P = \{x\}$  è un sottoinsieme dell'insieme delle variabili del linguaggio di vincoli considerato, allora vale:

$$\text{label}(\langle \{x\}, \{\mathbb{Z}\}, \{c\}, x \rangle) = (P_L, t) \quad (34)$$

dove  $t$  vale, in modo non deterministico, tutti i termini numerici univocamente associati ai valori nell'insieme:

$$\{z \in \mathbb{Z} : |z| \geq 2\} = \{\pm 2, \pm 3, \dots\} \quad (35)$$

Quindi, se applicata più volte, la funzione *label* avrà come risultato  $t$  un termine tipo  $2, -2, 3, -3, \dots$  ottenendo termini diversi per ogni applicazione.

Dato un linguaggio di vincoli  $\mathcal{D} = \langle D, C, A, V, \text{sat}, \text{post}, \text{label} \rangle$  è possibile estendere Prolog<sub>0</sub> al linguaggio CLP<sub>0</sub>( $\mathcal{D}$ ) ottenuto permettendo di utilizzare i vincoli come se fossero dei predicati. In particolare, dal punto di vista sintattico:

1. L'insieme degli atomi di CLP<sub>0</sub>( $\mathcal{D}$ ) contiene gli atomi di Prolog<sub>0</sub>, i simboli di vincolo in  $C$ , gli atomi in  $A$  e la parola *label*; e
2. L'insieme delle variabili di CLP<sub>0</sub>( $\mathcal{D}$ ) contiene tutte le variabili di Prolog<sub>0</sub> e tutte le variabili in  $V$ .

Si noti che non è richiesto che gli insiemi che arricchiscono CLP<sub>0</sub>( $\mathcal{D}$ ) rispetto a Prolog<sub>0</sub> siano disgiunti dai rispettivi insiemi di Prolog<sub>0</sub>. Quindi, normalmente, il linguaggio CLP<sub>0</sub>( $\mathcal{D}$ ) ha tipicamente lo stesso insieme di variabili di Prolog<sub>0</sub>.

Dal punto di vista semantico, le funzioni che calcolano il valore semantico di un programma scritto in  $\text{CLP}_0(\mathcal{D})$  per un particolare goal vengono arricchite di un nuovo argomento, che è un CSP costruito mediante il linguaggio di vincoli  $\mathcal{D}$ . Oltre a questo nuovo argomento, le funzioni hanno anche un nuovo risultato, che è anch'esso un CSP costruito mediante il linguaggio di vincoli  $\mathcal{D}$ . Infatti, dato un programma e un goal scritti in  $\text{CLP}_0(\mathcal{D})$ , il valore semantico calcolato è una coppia in cui la prima componente è una sostituzione e la seconda componente è un CSP che esprime i vincoli accumulati durante la computazione e relativi, almeno, alle variabili della sostituzione.

In particolare, la funzione  $\sigma_G$  è estesa in modo che tratti i congiunti che hanno come testa un simbolo di vincolo in modo specifico. Anziché cercare calcolare il valore semantico di un congiunto di questo tipo facendo ricorso all'unificazione, come avviene in Prolog<sub>0</sub>, vengono utilizzate le funzioni *sat* e *post*. Quindi, dato un congiunto che abbia come testa un simbolo di vincolo, viene applicata *post* al congiunto e al CSP che viene ricevuto come nuovo argomento di  $\sigma_G$ . Se la funzione *post* è effettivamente definita per il congiunto, allora viene verificata la soddisfacibilità del risultato della *post* mediante *sat*. La funzione  $\sigma_G$  produce un risultato, in questo caso deterministico, solo nel caso in cui la funzione *sat* indichi che non è stata in grado di determinare che il CSP ottenuto dalla *post* è insoddisfacibile. In questo caso, il risultato prodotto dalla  $\sigma_G$  è la sostituzione vuota insieme al nuovo CSP.

Si noti che, siccome la funzione *sat* non è sempre in grado di determinare se un CSP è insoddisfacibile,  $\text{CLP}_0(\mathcal{D})$  può produrre risultati anche per CSP insoddisfacibili. Però, è possibile garantire uno studio completo della soddisfacibilità mediante la funzione *label* che associa un valore alla variabile che riceve come argomento solo quando il CSP è soddisfacibile, nel caso pessimo al momento dell'assegnazione dell'ultima variabile libera. Infatti,  $\text{CLP}_0(\mathcal{D})$  include nella propria semantica la possibilità di utilizzare la funzione *label* mediante un predicato che porta lo stesso nome e che può essere applicato ad una lista di variabili che vengono assegnate nell'ordine in cui sono presenti nella lista.

Infine, si noti che la semantica di  $\text{CLP}_0(\mathcal{D})$  è coerente con l'unificazione. In particolare, se il linguaggio di vincoli  $\mathcal{D}$  mette a disposizione un vincolo per imporre l'uguaglianza tra due valori, allora il vincolo è soddisfatto se e soltanto se le i termini corrispondenti ai due valori sono unificabili. In più, l'unificazione tra due variabili garantisce che entrambe le variabili siano utilizzate in modo indistinguibile nei rispettivi vincoli. Quindi, in sostanza, l'unificazione permette di riferirsi alle variabili di un CSP con nomi alternativi.

Un linguaggio di vincoli molto usato è *FD* (da *Finite Domains*), che è un linguaggio di vincoli in cui:

1. Il dominio  $D$  è composto da tutti i sottoinsiemi finiti di  $\mathbb{Z}$ , da cui il nome;
2. Tra gli atomi sono presenti i simboli univocamente associati ai numeri interi e, quindi, 1, 2, . . . ;
3. Tra gli atomi sono presenti i simboli univocamente associati alle operazioni aritmetiche tra interi e, quindi, +, - e altri;

4. Tra i simboli di vincolo sono presenti i simboli `#=`, `#\=`, `#<`, `#=<` e altri, che permettono di imporre i comuni vincoli di confronto tra numeri interi e vengono di solito utilizzati come operatori infissi;
5. Tra i simboli di vincolo è presente `in`, tipicamente usato come operatore infisso, che permette di attribuire un dominio ad una variabile esprimendo il dominio come un'unione di intervalli espressi usando `..` e `\/`, rispettivamente, per denotare gli intervalli e farne l'unione;
6. Tra i simboli di vincolo è presente il simbolo `ins`, tipicamente usato come operatore infisso, che permette di attribuire un dominio, esattamente come `in`, però ad un insieme di variabili elencate in una lista; e
7. Tra i simboli di vincolo sono presenti simboli per esprimere vincoli di utilizzo comune, tra i quali il simbolo `all_distinct` che permette di imporre che la lista di variabili utilizzate come argomento sia formata unicamente da variabili associate a termini tra loro tutti diversi.

Il linguaggio CLP<sub>0</sub>(*FD*) estende Prolog<sub>0</sub> mediante i vincoli del linguaggio di vincoli *FD*. In modo analogo, il linguaggio CLP(*FD*) estende Prolog mediante *FD*. Per utilizzare SWI-Prolog come CLP(*FD*) è sufficiente includere il relativo modulo mediante il goal

```
:– use_module(library(clpfd)).
```

che viene tipicamente posizionato all'inizio del programma. L'utilizzo di CLP(*FD*) permette di scrivere un predicato che sia soddisfatto da una lista e dalla relativa lunghezza:

```
length([], 0).
length([_ | L], Length) :-
    Length #>= 1,
    N1 #= Length - 1,
    length(L, N1).
```

Si noti che SWI-Prolog non permette di ridefinire questo predicato perché è uno dei predicatori *built-in* di basso livello del linguaggio.

**Esempio 26.** Il seguente programma CLP(*FD*) permette di enumerare tutte le soluzioni del problema di criptoaritmetica *send + more = money*:

```
:– use_module(library(clpfd)).

send(Vars) :-
    Vars = [S,E,N,D,M,O,R,Y],
    Vars ins 0..9,
    M #\= 0, S #\= 0,
    all_distinct(Vars),
    S*1000 + E*100 + N*10 + D +
    M*1000 + O*100 + R*10 + E #=
    M*10000 + O*1000 + N*100 + E*10 + Y,
    label(Vars).
```

**Esempio 27.** Il seguente programma CLP(*FD*) permette di enumerare tutte le soluzioni del problema delle *n*-regine:

```

:- use_module(library(clpf)).
queens(N, Queens) :-  

    length(Queens, N),  

    Queens ins 1..N,  

    safe_queens(Queens),  

    label(Queens).
safe_queens([]).
safe_queens([Queen | QueenRest]) :-  

    safe_queens(QueenRest, Queen, 1),
    safe_queens(QueenRest).
safe_queens([], _, _).
safe_queens([Queen | QueenRest], Queen0, D0) :-  

    Queen0 #\= Queen,
    abs(Queen0 - Queen) #\= D0,
    D1 #= D0 + 1,
    safe_queens(QueenRest, Queen0, D1).

```

Un altro linguaggio di vincoli interessante è *Dif*. Questo linguaggio offre un solo vincolo **dif** che permette di imporre che il termine a primo argomento sia sempre diverso dal termine a secondo argomento, intendendo che due termini sono diversi se non sono unificabili. Nonostante questo linguaggio di vincoli sia minimale, è particolarmente rilevante perché CLP(*Dif*) permette spesso di ovviare ai noti problemi dei predicati che implicano una negazione.

**Esempio 28.** Il seguente predicato permette di imporre che il primo argomento non sia mai incluso nella lista che viene utilizzata come secondo argomento:

```

:- use_module(library(dif)).
nevermember(_, []).
nevermember(X, [H | R]) :-  

    dif(X, H),
    nevermember(X, R).

```

## 10 Prolog e Logica

Nonostante il linguaggio Prolog sia l'esempio più tipico di un linguaggio di programmazione logica, non è ancora stato esplorato il legame tra Prolog e la Logica. In particolare, non è ancora stato affrontato il problema di capire se i risultati prodotti da un programma scritto in Prolog possono essere considerate conseguenze logiche valide dei fatti e delle regole incluse nel programma. Per affrontare questo

problema, seguendo la strada tracciata da *Robert A. Kowalski*, è però prima necessario introdurre i linguaggi della logica dei predicati e descrivere il metodo della risoluzione.

Un linguaggio della *logica dei predicati* (o della *logica del primo ordine*) è un linguaggio che può essere costruito partendo da una *signature*  $\langle P, C, F, V \rangle$  dove:

1.  $P \neq \emptyset$  è un insieme di *simboli di predicato* (o *predicati*), ognuno dei quali è associato ad un'arità;
2.  $C \neq \emptyset$  è un insieme di *simboli di costante* (o *costanti*);
3.  $F$  è un insieme di *simboli di funzione* (o *funzioni*), ognuno dei quali è associato ad un'arità; e
4.  $V \neq \emptyset$  è un insieme infinito numerabile di *simboli di variabile* (o *variabili*).

Si noti che viene richiesto che  $V \cap C = \emptyset$  e che gli insiemi non contengano le parentesi tonde, il punto e alcuni simboli che vengono comunemente utilizzati nei linguaggi di questo tipo per altri scopi, quali, ad esempio,  $\wedge$ ,  $\vee$  e  $\top$ .

Partendo da una signature  $\langle P, C, F, V \rangle$  è possibile costruire il relativo linguaggio della logica dei predicati mediante una struttura a due livelli. Il primo livello definisce la forma dei *termini* (*del primo ordine*) come segue:

1. Ogni simbolo di variabile è un termine;
2. Ogni simbolo di costante è un termine; e
3. Se  $t_1, t_2, \dots, t_n$  sono termini e  $f$  è un simbolo di funzione di arità  $n \in \mathbb{N}_+$ , allora  $f(t_1, t_2, \dots, t_n)$  è un termine; e
4. Nient'altro è un termine.

Per un termine  $t$  è definita la funzione  $vars(t)$  che costruisce l'insieme delle variabili contenute nel termine nel seguente modo:

1. Se  $t$  è una variabile, allora  $vars(t) = \{t\}$ ;
2. Se  $t$  è una costante, allora  $vars(t) = \emptyset$ ; e
3. Se  $t = f(t_1, t_2, \dots, t_n)$  per un qualche simbolo di funzione  $f$  di arità  $n \in \mathbb{N}_+$  e per i termini  $t_1, t_2, \dots, t_n$ , allora  $vars(t) = vars(t_1) \cup vars(t_2) \cup \dots \cup vars(t_n)$ .

Un termine  $t$  viene detto *chiuso* (o *ground*) se  $vars(t) = \emptyset$ .

Partendo dai termini così definiti, è possibile costruire il secondo livello della struttura del linguaggio. Questo livello contiene l'insieme delle *formule ben formate* (o *formule ben formulate*) del linguaggio. Le formule ben formulate di un linguaggio di questo tipo vengono chiamate *proposizioni* e vengono descritte come segue:

1. Se  $t_1, t_2, \dots, t_n$  sono termini e  $p$  è un simbolo di predicato di arità  $n \in \mathbb{N}_+$ , allora  $p(t_1, t_2, \dots, t_n)$  è una proposizione che viene comunemente detta *letterale affermato*;

2. I simboli  $\top$  (letto *top*) e  $\perp$  (letto *bottom*) sono proposizioni;
3. Se  $A$  è una proposizione, allora  $\neg(A)$  è una proposizione e se  $A$  è un letterale affermato, allora  $\neg(A)$  è un *letterale negato*;
4. Se  $A$  e  $B$  sono proposizioni, allora  $(A \wedge B)$ ,  $(A \vee B)$ ,  $(A \implies B)$ , and  $(A \iff B)$  sono proposizioni;
5. Se  $A$  è una proposizione e  $x \in V$  è una variabile, allora  $\exists x.(A)$  e  $\forall x.(A)$  sono proposizioni; e
6. Nient'altro è una proposizione.

Si noti che è consentito rimuovere le parentesi tonde se una proposizione è composta da un unico predicato o dai simboli  $\top$  e  $\perp$  e quando viene adottata la seguente precedenza per gli operatori binari:  $\wedge, \vee, \implies, \iff$ .

I quantificatori definiscono un  (o *scope*) per una variabile e, data una proposizione, ogni occorrenza di una variabile nel campo di azione di un quantificatore viene detta *occorrenza vincolata*. Viceversa, ogni occorrenza di una variabile in una proposizione che non sia nel campo d'azione di un quantificatore viene detta *occorrenza libera*. Una proposizione viene dette *chiusa* se ha unicamente variabili con occorrenze vincolate.

Dato un insieme  $D \neq \emptyset$ , detto *dominio del linguaggio*, e una signature  $\langle P, C, F, V \rangle$  è possibile associare un *valore di verità* (o *valore semantico*) alle proposizioni del linguaggio costruito mediante la signature a disposizione scegliendo una *funzione di interpretazione*  $I$  con le seguenti caratteristiche:

1. Per ogni  $c \in C$  esiste un elemento  $I(c) \in D$  chiamato interpretazione del simbolo di costante  $c$ ;
2. Per ogni  $f \in F$  di arità  $n \in \mathbb{N}_+$  esiste una funzione  $I(f) : D^n \rightarrow D$  chiamata interpretazione del simbolo di funzione  $f$ ; e
3. Per ogni  $p \in P$  di arità  $n \in \mathbb{N}_+$  esiste una relazione di arità  $n$  definita su  $D$ ,  $I(p) \subseteq D^n$ , chiamata interpretazione del simbolo di predicato  $p$ .

Fissata una funzione di interpretazione e un'*assegnazione*  $s : V \rightarrow D$  è possibile associare un'interpretazione ai termini come segue:

1. L'interpretazione di un simbolo di costante  $c \in C$  (per l'assegnazione  $s$ ) è  $I_s(c) = I(c)$ ;
2. L'interpretazione di un simbolo di variabile  $x \in V$  per l'assegnazione  $s$  è  $I_s(x) = s(x)$ ; e
3. L'interpretazione di un *termine strutturato*  $t = f(t_1, t_2, \dots, t_n)$  per un assegnazione  $s$  è  $I_s(t) = I(f)(I_s(t_1), I_s(t_2), \dots, I_s(t_n))$ .

Fissata un'interpretazione  $I$  e un'assegnazione  $s$  è possibile dire quando l'interpretazione  $I$  soddisfa una proposizione  $A$  secondo l'assegnazione  $s$ ,  $(I, s) \models A$ . Si noti subito che, a seconda dell'assegnazione considerata, una singola proposizione potrà essere soddisfatta da un'interpretazione  $I$  o meno. Per verificare se un'interpretazione  $I$  soddisfa una proposizione  $A$  secondo un'assegnazione  $s$  si utilizzano le seguenti regole, dove  $s_{x \mapsto d}$  è l'assegnazione identica ad  $s$  tranne che per la variabile  $x$  che viene assegnata a  $d$ :

1.  $(I, s) \models p(t_1, t_2, \dots, t_n)$  se e soltanto se  $(I_s(t_1), I_s(t_2), \dots, I_s(t_n)) \in I(p)$
2.  $(I, s) \models \top$  e  $(I, s) \not\models \perp$ ;
3.  $(I, s) \models \neg A$  se e soltanto se  $(I, s) \not\models A$ ;
4.  $(I, s) \models A \wedge B$  se e soltanto se  $(I, s) \models A$  e  $(I, s) \models B$ ;
5.  $(I, s) \models A \vee B$  se e soltanto se  $(I, s) \models A$  oppure  $(I, s) \models B$ ;
6.  $(I, s) \models A \implies B$  se e soltanto se  $(I, s) \not\models A$  oppure  $(I, s) \models B$ ;
7.  $(I, s) \models A \iff B$  se e soltanto se  $(I, s) \models A$  e  $(I, s) \models B$ , oppure  $(I, s) \not\models A$  e  $(I, s) \not\models B$ ;
8.  $(I, s) \models \exists x.A$  se e soltanto se esiste un  $d \in D$  tale che  $(I, s_{x \mapsto d}) \models A$ ; e
9.  $(I, s) \models \forall x.A$  se e soltanto se per ogni  $d \in D$  vale  $(I, s_{x \mapsto d}) \models A$ .

Si noti che  $(I, s) \models A$  viene anche letto:  $A$  è vera in  $I$  secondo l'assegnazione  $s$ . In particolare, si dirà che  $A$  è vera in  $I$ ,  $I \models A$ , se  $A$  è vera in  $I$  per ogni sostituzione  $s$ . In questo caso,  $I$  viene detta *modello* di  $A$ . Infine, si dirà che  $A$  non è vera in  $I$ ,  $I \not\models A$ , se non esiste alcuna sostituzione  $s$  tale per cui  $A$  è vera in  $I$  per  $s$ . In questo caso,  $I$  viene detta *contromodello* di  $A$ .

Si consideri una *teoria (del primo ordine)*  $T$ , che non è altro che un insieme di proposizioni. La teoria si dice vera in un'interpretazione  $I$  e secondo un'assegnazione  $s$ ,  $(I, s) \models T$ , se e soltanto se per ogni proposizione  $P$  in  $T$  vale  $(I, s) \models P$ . Una teoria si dice vera in un'interpretazione  $I$ ,  $I \models T$ , se la teoria è vera per ogni assegnazione. In questo caso, l'interpretazione  $I$  si dice modello della teoria. Analogamente è possibile definire quando un'interpretazione non è vera in una teoria e quindi definire i contromodelli di una teoria. Data una teoria  $T$  e una proposizione  $A$ , non necessariamente contenuta in  $T$ , si dice che  $A$  è *conseguenza logica* di  $T$ ,  $T \models A$ , se per ogni interpretazione  $I$  e sostituzione  $s$  che rendono vera la teoria  $T$  vale anche che  $(I, s) \models A$ . Quindi, per ogni interpretazione e assegnazione che rendono vera la teoria anche la conseguenza logica risulta vera. Naturalmente, non viene richiesto che la conseguenza logica sia vera unicamente per le interpretazioni e le sostituzioni che rendono vera  $T$ .

Date una teoria  $T$  e una proposizione  $A$  è possibile studiare se  $T \models A$  mediante il *metodo della risoluzione*, che è un particolare tipo di *metodo di refutazione* (o *metodo di riduzione all'assurdo*). I metodi di refutazione si basano sul fatto che  $T \models A$  se e soltanto se non esistono un'interpretazione  $I$  e una sostituzione  $s$  tali che  $(I, s) \models T \cup \{\neg A\}$ .

Il metodo di risoluzione può essere applicato per studiare la *soddisfacibilità* di un insieme generico di proposizioni, quindi l'esistenza di un modello per l'insieme di proposizioni, mediante il metodo dovuto a *Albert Thoralf Skolem*. Però, nell'ambito della programmazione logica, si prevede che le teorie siano formate unicamente da proposizioni di un particolare tipo detto *clausole di Horn*, dal nome di *Alfred Horn*. Una clausola di Horn è una proposizione in cui:

1. Non sono ammessi i quantificatori esistenziali;
2. I quantificatori universali, se presenti, sono unicamente nella parte iniziale della proposizione;
3. La parte della proposizione che segue i quantificatori universali è formata unicamente da una disgiunzione di letterali; e
4. Non più di un letterale può essere positivo.

Quindi, una clausola di Horn è una proposizione del tipo:

$$\forall x_1. \forall x_2. \dots \forall x_n. (L_1 \vee L_2 \vee \dots \vee L_m) \quad (36)$$

e non più di uno dei letterali in  $\{L_i\}_{i=1}^m$  è positivo. Una clausola di Horn si dice *clausola definita* se ammette un solo letterale positivo. Mentre una clausola di Horn si dice *clausola goal* se ammette zero letterali positivi. Si noti che si ammette, impropriamente, la possibilità di avere una *clausola (di Horn) vuota*, cioè con zero letterali, per indicare  $\perp$ . Infine, conviene notare che se viene tralasciato il vincolo che prevede la presenza di non più di un letterale positivo in una clausola di Horn, allora le proposizioni vengono dette semplicemente *clausole*.

Si consideri una teoria  $T$  formata unicamente da clausole di Horn, se esistono due clausole nella teoria

$$\forall x_1. \forall x_2. \dots \forall x_n. (L_1 \vee L_2 \vee \dots \vee L_m) \quad (37)$$

$$\forall y_1. \forall y_2. \dots \forall y_r. (M_1 \vee M_2 \vee \dots \vee M_s) \quad (38)$$

che non condividono variabili e tali che esistano  $1 \leq i \leq n$  e  $1 \leq j \leq s$  per cui  $L_i \theta = \neg M_j \theta$ , con  $\theta = mgu(L_i, M_j)$ , allora è possibile costruire la *clausola risolvente*  $R$  di  $L_i$  ed  $M_j$ :

1. Applicando la sostituzione  $\theta$  ad ognuno dei letterali delle due clausole, esclusi  $L_i$  ed  $M_j$ ;
2. Unendo tutti i letterali ottenuti dall'applicazione della sostituzione  $\theta$ , e quindi escludendo  $L_i$  ed  $M_j$ , mediante delle disgiunzioni; e
3. Quantificando universalmente su tutte le variabili contenute nei nuovi letterali ottenuti dall'applicazione della sostituzione.

La clausola risolvente  $R$  è stata costruita con la cosiddetta *regola di risoluzione* e si può dimostrare che la clausola risolvente  $R$  è conseguenza logica di  $T$ . Si noti che la regola di risoluzione può produrre la clausola vuota e, in questo caso, il risultato

viene più propriamente indicato con  $\perp$ . In più, si noti che la clausola risolvente è ottenuta da due clausole che non condividono variabili, quindi se si considerano clausole che condividono variabili sarà sufficiente sostituire queste variabili in una delle due clausole con delle nuove variabili per poter applicare la regola. Infine, si noti che è stata applicata l'unificazione al linguaggio di termini formato da un alfabeto che contiene anche i simboli di predicato.

Data una teoria  $T$  formata unicamente da clausole definite è possibile dimostrare che una proposizione  $G$  del tipo:

$$\exists x_1. \exists x_2. \dots \exists x_n. L_1 \wedge L_2 \wedge \dots \wedge L_m \quad (39)$$

dove i letterali in  $\{L_i\}_{i=1}^m$  sono tutti affermati, è conseguenza logica di  $T$  cercando di applicare iterativamente la regola di risoluzione alla teoria  $T \cup \{\neg G\}$  fino ad ottenere la clausola vuota. Infatti, è vero che se  $T \cup \{\neg G\} \vdash_{RES} \perp$  allora  $T \cup \{\neg G\} \models \perp$  e quindi  $T \models G$ . Si noti che l'applicazione iterata della regola di risoluzione fa aumentare il numero di clausole contenute nella teoria perché, per ogni applicazione della regola, si estende la teoria con la clausola risolvente ottenuta.

**Esempio 29.** Si consideri un linguaggio della logica dei predici con simboli di predicato in  $P = \{p\}$ , simboli di costante in  $C = \{a, b, e\}$ , simboli di funzione in  $F = \{f\}$  e simboli di variabile in  $V$  che contiene tutte le parole che iniziano con una lettera maiuscola. Partendo dalla seguente teoria espressa in forma di clausole, dove, come di consueto, sono stati lasciati impliciti i quantificatori:

$$\begin{aligned} (1) \quad & p(e, X, X) \\ (2) \quad & p(f(X, Y), Z, f(X, W)) \vee \neg p(Y, Z, W) \end{aligned}$$

si vuole dimostrare che esiste un  $T$  tale che  $p(f(a, e), T, f(a, f(b, e)))$ . Per farlo, è possibile procedere *in avanti* (*o forward chaining*), come segue:

$$(3) \quad p(f(X, e), S, f(X, S)) \quad RES(1, 2) \quad \{S/W, e/Y, S/Z\}$$

da cui è possibile ottenere che la sostituzione  $\{f(b, e)/S, a/X\}$  applicata a (3) porta al risultato cercato applicando la sostituzione  $\{f(b, e)/T\}$  alla proposizione goal che si vuole dimostrare. In alternativa è possibile procedere *all'indietro* (*backward chaining*) e quindi per refutazione. Per prima cosa è necessario negare la proposizione goal che si vuole dimostrare ottenendo la la seguente clausola goal:

$$(3) \quad \neg p(f(a, e), T, f(a, f(b, e)))$$

che viene aggiunta alla teoria. A questo punto è possibile procedere come segue:

$$\begin{aligned} (4) \quad & \neg p(e, U, f(b, e)) \quad RES(2, 3) \quad \{e/Y, U/Z, f(b, e)/W\} \\ (5) \quad & \perp \quad RES(1, 4) \end{aligned}$$

da cui si evince, procedendo a ritroso con le sostituzioni, che la conseguenza logica della teoria iniziale si ottiene applicando la sostituzione  $\{f(b, e)/T\}$  alla proposizione goal che si vuole dimostrare.

Il legame tra la programmazione in linguaggio Prolog e il metodo della risoluzione risulta evidente riscrivendo le clausole definite come:

$$\forall x_1. \forall x_2. \dots \forall x_n. (P \iff \exists y_1. \exists y_2. \dots \exists y_r. L_1 \wedge L_2 \wedge \dots \wedge L_s) \quad (40)$$

dove  $P$  è l'unico letterale affermato della clausola definita considerata, e dipende solo dalle variabili in  $\{x_i\}_{i=1}^n$ , e sono state sfruttate alcune *equivalenze logiche* che riguardano i quantificatori, la negazione e l'implicazione per operare la riscrittura. Infatti:

$$\forall x_1. \forall x_2. \dots \forall x_n. \forall y_1. \forall y_2. \dots \forall y_r. (P \vee \neg L_1 \vee \neg L_2 \vee \dots \vee \neg L_s) \quad (41)$$

può essere riscritta come:

$$\forall x_1. \forall x_2. \dots \forall x_n. (P \vee \forall y_1. \forall y_2. \dots \forall y_r. (\neg L_1 \vee \neg L_2 \vee \dots \vee \neg L_s)) \quad (42)$$

e quindi:

$$\forall x_1. \forall x_2. \dots \forall x_n. (P \iff \neg(\forall y_1. \forall y_2. \dots \forall y_r. (\neg L_1 \vee \neg L_2 \vee \dots \vee \neg L_s))) \quad (43)$$

da cui si ottiene:

$$\forall x_1. \forall x_2. \dots \forall x_n. (P \iff \exists y_1. \exists y_2. \dots \exists y_r. \neg(\neg L_1 \vee \neg L_2 \vee \dots \vee \neg L_s)) \quad (44)$$

e, infine, risulta:

$$\forall x_1. \forall x_2. \dots \forall x_n. (P \iff \exists y_1. \exists y_2. \dots \exists y_r. (L_1 \wedge L_2 \wedge \dots \wedge L_s)) \quad (45)$$

Quindi, un programma scritto in Prolog non è altro che una teoria logica espressa mediante clausole definite. I fatti sono clausole definite senza letterali negati e le regole sono clausole definite con almeno un letterale negato. Il goal necessario per attivare la computazione di un programma scritto in Prolog non è altro che una proposizione per la quale si vuole verificare se sia conseguenza logica della teoria. La funzione semantica che descrive la computazione svolta dal programma a fronte di un goal non è altro che l'applicazione della regola di risoluzione secondo una strategia, rigida ma efficiente, che prende il nome di *SLD* (da *Selective Linear Definite clause resolution*). Quindi, qualsiasi sostituzione ottenuta da un programma scritto in Prolog a fronte di un goal è una conseguenza logica valida dei fatti e delle regole scritte nel programma.