UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica

Un ambiente di programmazione simbolica per lo studio di MV insiemi

Pietro Codara

Relatore:

Prof. Ottavio M. D'Antona

Correlatore:

Dott. Vincenzo Marra

Anno Accademico 2002/2003

INDICE

INTRODUZIONE

Questa tesi predispone un ambiente di elaborazione simbolica per lo studio combinatorio degli MV set[1].

I primi capitoli espongono le linee essenziali degli MV set. Prendendo come esempio la cosiddetta Tavola Pitagorica dell'analisi combinatoria, elaborata da G-C. Rota sulla base della "metafora delle palline delle scatole", cercheremo di trasporre nel nostro caso il conteggio dei modi di disporre n palline distinguibili in x scatole distinguibili .

Le basi da cui si partirà sono quindi abbastanza semplici e fondate sulle seguenti [3]:

Proposizione 1 *In assenza di altri vincoli, il numero di modi di mettere n palline in x scatole è* x^n.

Proposizione 2 *Il numero di assegnamenti iniettivi di n palline in x scatole, ovvero il numero di modi di mettere n palline in x scatole in modo tale che in ogni scatola ci sia al più una pallina, è* $(x)_n$ [2].

Proposizione 3 *Se* $n < x$*, il numero di assegnamenti suriettivi di n palline distinguibili in x scatole distinguibili, ovvero il numero di modi di mettere n*

---

[1] Utilizzeremo sin da ora il termine "MV set" in luogo di "MV insiemi" per rappresentare questa nuova categoria di oggetti. MV sta per *"many-valued"*, ma si userà pronunciarlo esattamente come è scritto: si parlerà cioè di uno o più MV set.

[2] Con il simbolo $(x)_n$ indichiamo il *fattoriale decrescente*. Si ha: $(x)_0 = 1$ e, per n positivo, $(x)_n = x(x-1)\cdots(x-n+1)$.

palline in x scatole in modo tale che in ogni scatola ci sia almeno una pallina, è 0, altrimenti tale numero è $x!S(n, x)$ [3].

Da un punto di vista insiemistico, considerando i due insiemi delle palline e delle scatole, la funzione nella proposizione 1 conta il numero di funzioni tra i due insiemi, quella nella proposizione 2 conta il numero di funzioni iniettive e quella nella 3 conta il numero di funzioni suriettive.

In questo elaborato si affronterà in un primo tempo un'introduzione teorica agli MV set, che sfocerà nella stesura di formule per il calcolo combinatorio. In una seconda parte si affronterà lo sviluppo di un pacchetto *Mathematica*® che definirà il tipo di dato MV set e calcolerà su di esso alcune funzioni di base e le funzioni precedentemente sviluppate. Si procederà a fornire alcuni esempi di utilizzo del pacchetto; in particolare, oltre a descriverne l'utilizzo di base, si cercherà con esso di trovare delle applicazioni che diano risultati interessanti.

[3] Con il simbolo $S(n, x)$ indichiamo il numero di Stirling di seconda specie, ovvero il numero di partizioni di un insieme di ordine n in k blocchi.

Capitolo 1

NOZIONI DI BASE SUGLI MV SET

"Devono essere stati necessari molti secoli per scoprire che una coppia di fagiani ed un paio di giorni sono entrambi espressioni del numero 2; il grado di astrazione che questo comporta non è certo lieve. (…) Quanto allo 0, è una scoperta recentissima; i Greci ed i Romani non avevano questa cifra."

(Bertrand Russel)

Partendo da una semplice definizione degli MV set, verranno introdotti i concetti di morfismo, funzione iniettiva e suriettiva e cardinalità.

1.1 Gli MV set

Un MV set è una funzione

$$\alpha : A \to \mathbb{N}$$

dall'insieme finito A ai numeri naturali $\mathbb{N} = \{1, 2, \ldots\}$. Chiameremo $A = \text{Dom}(\alpha)$ *insieme supporto* di α o anche *riduzione booleana* di α. Si può notare come, nel caso il codominio si riduca all'insieme $\{0, 1\}$, α sia un insieme classico.

Utilizzeremo spesso le lettere greche α e γ ad indicare due MV set aventi insiemi supporto, rispettivamente, $A = \{a_1, \ldots, a_n\}$ e $C = \{c_1, \ldots, c_x\}$.

Un'altro modo di scrivere gli MV set sarà:

$$\alpha = \{a_1^{m_1}, \ldots, a_n^{m_n}\}$$

dove $\alpha(a_i) = m_i$ per $i \in \{1, \ldots, n\}$. Chiameremo m_i la *molteplicità* dell'elemento a_i. Scriveremo inoltre:

$$a_1^{m_1} \in \alpha$$

per dire che $a_1^{m_1}$ è un MV elemento di α. Si potrà anche scrivere $a_i \in \alpha$ per indicare che a_i è un elemento dell'insieme supporto A di α.

Sarà inoltre possibile, e lo faremo in seguito, definire la MV cardinalità di un MV set. Ci limitiamo per ora a dire che con il simbolo $|\alpha|$ indichiamo la cardinalità in senso classico, ovvero la cardinalità dell'insieme supporto, ovvero $|A|$.

È importante osservare che l'utilizzo del prefisso MV sarà molto diffuso. Si parlerà, per esempio, di *MV elementi*, *MV funzioni*, *MV suriezioni*, etc: questa notazione favorirà la distinzione con elementi, funzioni, suriezioni, etc intesi in senso classico.

1.2 MV funzioni

Dati due MV set α e γ, aventi insiemi supporto, rispettivamente, A e C, un *morfismo* f tra α e γ, chiamato anche *MV funzione* da α a γ o, in simboli, $f : \alpha \to \gamma$, è una funzione $f : A \to C$ tale che, per ogni $a \in A$,

$$\gamma \circ f(a) \text{ div } \alpha(a)$$

dove *x div y* vuol dire *x divide y*. Si possono effettuare composizioni di MV funzioni. Date $f : \alpha \to \xi$ e $g : \xi \to \gamma$, $\quad g \circ f : \alpha \to \gamma$ denoterà la MV funzione ottenuta componendo f e g.

Una MV funzione da α a se stesso è una *funzione identità* su α, indicata con $\mathbf{1}_\alpha$, *se e solo se* è una funzione identità sull'insieme supporto.

Una MV funzione $f : \alpha \to \gamma$ è un *MV isomorfismo*, o una *MV biezione*, *se e solo se* esiste $g : \gamma \to \alpha$ tale che $g \circ f = \mathbf{1}_\alpha$ e $f \circ g = \mathbf{1}_\gamma$. È evidente che $f : \alpha \to \gamma$ è una MV biezione *se e solo se* $f : A \to C$ è una biezione che

preserva le molteplicità, cioè se $f : A \to C$ è un isomorfismo (nella categoria degli insiemi) e $\gamma \circ f = \alpha$.

Esempio 1.1 *Siano dati due MV set* $\alpha = \{a_1^2, a_2^5\}$ *e* $\gamma = \{c_1^1, c_2^2\}$ *, aventi insiemi supporto* $A = \{a_1, a_2\}$ *e* $C = \{c_1, c_2\}$ [1].

- *La funzione* $f : A \to C$ *tale che*[2] $a_1 \mapsto c_2$ *e* $a_2 \mapsto c_1$ *è una MV funzione , perché* 2 div 2 *e* 1 div 5.

- *La funzione* $f : A \to C$ *tale che* $a_1 \mapsto c_1$ *e* $a_2 \mapsto c_2$ *non è una MV funzione, perché* 2 non divide 5.

- α *e* γ *non sono isomorfi (non esiste un MV isomorfismo tra* α *e* γ*) , perché nessuna delle biezioni (classiche) tra A e C preserva le molteplicità.*

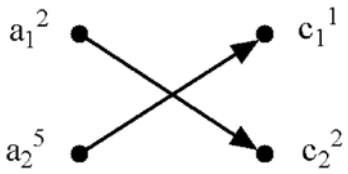

Figura 1.1: MV funzione.

Esempio 1.2 *Siano dati* $\alpha = \{a_1^1, a_2^2, a_3^4\}$ *e* $\gamma = \{c_1^1, c_2^3\}$.

- *Esiste una sola MV funzione da* α *a* γ*: la funzione* $f : A \to C$ *tale che* $a_1 \mapsto c_1$ *,* $a_2 \mapsto c_1$ *e* $a_3 \mapsto c_1$.

---

[1] Si è voluto, solo in questo primo esempio, specificare la natura degli insiemi supporto. In seguito gli insiemi supporto verranno omessi, in quanto direttamente, da definizione, ricavabili dall' MV set e naturalmente correlati ad esso.Allo stesso modo si eviterà spesso di scrivere *l' MV set* α, perché già l'uso di lettere greche minuscole come α e γ indica che si sta parlando di MV set.

[2] $a_i \mapsto c_j$ indica che l'elemento a_i *è mappato nell'elemento* c_j

- *Non esistono MV isomorfismi tra α e γ , perché non ci sono biezioni da A a C.*

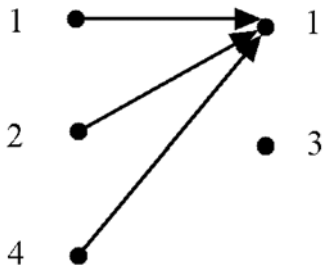

Figura 1.2: MV funzione.

1.3 MV iniezioni e MV suriezioni

Mentre nella categoria degli insiemi si parlava di funzioni iniettive e suriettive, nella categoria degli MV set il discorso è più complesso e ci saranno iniezioni (suriezioni) deboli e forti.

Definizione 1.1 *Sia $f : \alpha \to \gamma$ una MV funzione. Diciamo che:*

1. *f è debolmente iniettiva sse $f : A \to C$ è iniettiva.*

2. *f è debolmente suriettiva sse $f : A \to C$ è suriettiva.*

3. *f è fortemente iniettiva sse $f : A \to C$ è iniettiva e preserva le molteplicità, ovvero se è iniettiva e $\gamma \circ f = \alpha$.*

4. *f è fortemente suriettiva sse $f : A \to C$ è suriettiva e, per ogni $c \in C$, vale*

$$\gamma(c) = MCD\,\{\,\alpha(a) \mid f(a) = c\,\}.$$

Si può immediatamente verificare che le iniezioni forti e deboli, come le suriezioni deboli, sono chiuse sulla composizione di funzioni. La chiusura delle suriezioni forti è altresì verificabile, a partire dall'associatività dell'operatore MCD.

Nel seguito verrà utilizzata la notazione $f : \alpha \twoheadrightarrow \gamma$ per indicare che f è una suriezione forte. La notazione $f : \alpha \rightarrowtail \gamma$ indicherà invece che f è una iniezione forte.

Esempio 1.3 *Siano dati* $\alpha = \{a_1^2, a_2^5\}$ *e* $\gamma = \{c_1^1, c_2^2, c_3^5\}$.

- *La MV funzione* $f : \alpha \to \gamma$ *, tale che* $a_1 \mapsto c_2 \quad e \quad a_2 \mapsto c_1$ *è debolmente iniettiva (perché* $f : A \to C$ *è iniettiva), ma non fortemente iniettiva.* f*, inoltre, non è debolmente suriettiva.*

- *La MV funzione* $f : \alpha \to \gamma$ *, tale che* $a_1 \mapsto c_2 \quad e \quad a_2 \mapsto c_3$ *è fortemente iniettiva, perché* $f : A \to C$*, oltre ad essere iniettiva, preserva le molteplicità.*

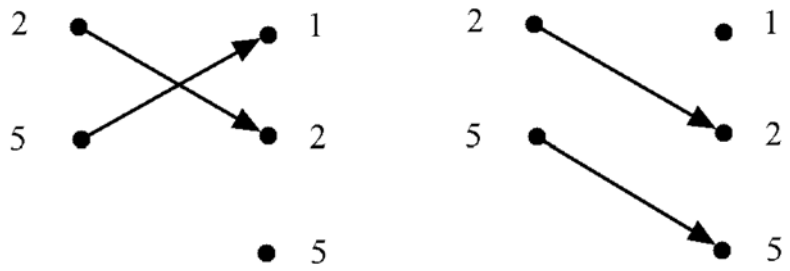

Figura 1.3: Iniettività debole (a sinistra) e forte (a destra).

Esempio 1.4 *Siano dati* $\alpha = \{a_1^2, a_2^6, a_3^3\}$ *e* $\gamma = \{c_1^2, c_2^3\}$.

- *La MV funzione* $f : \alpha \to \gamma$ *, tale che* $a_1 \mapsto c_1 \quad , \quad a_2 \mapsto c_2 \quad e \quad a_3 \mapsto c_2$ *è fortemente suriettiva, perché* $f : A \to C$ *è suriettiva e* $MCD\{6, 3\} = 3$ *(e, ovviamente,* $2 = 2$*).*

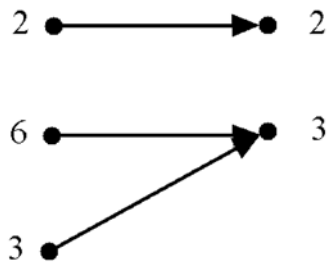

Figura 1.4: MV funzione fortemente suriettiva.

1.4 MV cardinalità

Come accennato nel primo paragrafo, è possibile definire una MV cardinalità di un MV set α, concetto ben più ampio della cardinalità dell'insieme supporto $|\alpha|$, poco esplicativa nella categoria che stiamo analizzando.
Iniziamo ricordando il concetto di *partizione di un intero*:

> la partizione di un intero non negativo $q \in \mathbb{Z}^+ = \{0, 1, \ldots, n\}$ è , per $q \neq 0$, una sequenza finita λ debolmente decrescente di numeri naturali $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_t$, $t \in \mathbb{N}$, tale che $q = \sum_{i=1}^{t} \lambda_i$; se $q = 0$, q ammette per definizione la sola partizione **0**.

Si scrive $\lambda \vdash q$ per indicare che λ è una partizione di q. λ_i si dice *addendo*, o *parte*, della partizione. Ricordiamo inoltre la funzione $p(q)$, che conta il numero di partizioni di un intero $q \in \mathbb{Z}^+$, e la funzione $p_k(q)$ che conta il numero di partizioni di q in non più di k parti (non necessariamente distinte). Nel seguito, oltre al più "classico" λ, verranno utilizzate le lettere ν e χ per indicare le partizioni di interi, mentre **0** e **1** indicheranno, rispettivamente, le partizioni (uniche) di 0 e 1.
Introduciamo ora la *funzione di distribuzione di frequenza* di una partizione,

concetto che ci sarà immediatamente utile al fine di definire un nuovo insieme: gli *MV numeri naturali*.

Definizione 1.2 *Sia* λ *la partizione* $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_t$. *La* distribuzione di frequenza *di* λ *è la funzione a valori interi*

$$s_i^\lambda = |\{\, j \in \mathbb{N} | \lambda_j = i \,\}|, \qquad i \in \mathbb{N}.$$

Per $i \in \mathbb{N}$, *il* cumulante i-esimo *di* λ *è*

$$S_i^\lambda = \sum_{j \text{ div } i} s_j^\lambda \quad ,$$

dove $j \in \mathbb{N}$.

Ci capiterà di scrivere s_i e S_i al posto di s_i^λ e S_i^λ quando λ è conosciuto.
Sia $i_1 < i_2 < \cdots < i_r$ la lista degli indici j tali che $s_j^\lambda > 0$, scriveremo allora λ come $\left\langle i_1^{s_{i_1}}, \cdots, i_r^{s_{i_r}} \right\rangle$.
Siamo così pronti per definire il nostro nuovo insieme, che chiameremo $\mathbb{MVN}$.

Definizione 1.3 *Un* MV numero naturale *è una partizione di un numero naturale. Useremo* $\mathbb{MVN}$ *per indicare l'insieme di tutti gli MV numeri naturali.* $\mathbb{MVZ}^+$ *sarà l'insieme di tutti gli MV numeri naturali unito allo* **0**, *unica partizione di* 0.

Si noti che $\mathbb{N}$ è proiettabile in $\mathbb{MVN}$ attraverso l'unica mappatura che fa corrispondere a $n \in \mathbb{N}$ la partizione, anch'essa unica, μ avente frequenze $s_1^\mu = n$,$s_i^\mu = 0$ per $i \neq 1$. L' $\mathbb{MVN}$ numero naturale corrispondente a questa partizione è $\langle 1^n \rangle$.
Un MV numero naturale $\nu = \left\langle i_1^{s_{i_1}}, \cdots, i_r^{s_{i_r}} \right\rangle$ descrive un unico MV set a meno di isomorfismi: l' MV set avente s_{i_j} elementi di molteplicità i_j , $j \in \{1, \ldots, r\}$.
Viceversa, α determina univocamente l'unica partizione $\nu \vdash \sum_{i=1}^n \alpha(a_i)$ avente frequenze $s_i^\nu = |\{a \in A | \alpha(a) = i\}|$.
Questa corrispondenza ci porta ad una naturale definizione di *MV cardinalità*.

Definizione 1.4 *La* MV cardinalità *di $\alpha \neq \emptyset$, indicata con $\|\alpha\|$, è quell'unico MV numero naturale che descrive, a meno di isomorfismi, l' MV set α , ovvero il numero*

$$\left\langle i_1^{s_{i_1}}, \cdots, i_r^{s_{i_r}} \right\rangle \in \mathbb{MVN} \quad ,$$

dove $i_1 < \cdots < i_r$ rappresentano il codominio di $\alpha : A \to \mathbb{N}$, e $s_{i_j} = |\{a \in A | \alpha(a) = i_j\}|$.
*Per $\alpha = \emptyset$, poniamo $\|\emptyset\| = \mathbf{0}$. Ogni α tale che $\|\alpha\| = \nu$ è un ν-*set.

Così come esiste un isomorfismo tra l'insieme $\mathbb{N}$ dei numeri naturali e la cardinalità degli insiemi finiti, esiste un isomorfismo tra l'insieme $\mathbb{MVN}$ degli MV numeri naturali e la MV cardinalità degli MV set. Ancora, come si può affermare:

> si dice *numero naturale* la cardinalità di un insieme finito

possiamo anche affermare:

> si dice *MV numero naturale* la MV cardinalità di un MV set (avente insieme supporto finito).

Qualche esempio ci sarà utile a chiarire il concetto di $\mathbb{MVN}$ e la corrispondenza tra MV set e MV numeri naturali.

Esempio 1.5 *Sia $\lambda \vdash 21$ la partizione* $5, 4, 4, 4, 2, 2$.

- *Calcoliamo la distribuzione di frequenza di λ:*
  $s_1^\lambda = |\{\, j \in \mathbb{N} | \lambda_j = 1 \,\}| = 0$
  $s_2^\lambda = 2$; $s_3^\lambda = 0$; $s_4^\lambda = 3$; $s_5^\lambda = 1$; $s_6^\lambda = s_7^\lambda = \cdots = 0$.

- *Calcoliamo ora i cumulanti:*
  $S_1^\lambda = \sum_{j \text{ div } 1} s_j^\lambda = s_1^\lambda = 0$
  $S_2^\lambda = s_1^\lambda + s_2^\lambda = 2$; $S_3^\lambda = s_1^\lambda + s_3^\lambda = 0$; $S_4^\lambda = 5$; $S_5^\lambda = 1$; $\ldots$

- *Possiamo scrivere* λ *come* $\left\langle i_1^{s_{i_1}}, \cdots, i_r^{s_{i_r}} \right\rangle$ *, dove* $i_1 < i_2 < \cdots < i_r$ *è la lista degli indici* j *tali che* $s_j^{\lambda} > 0$*. Si ha:* $\lambda = \left\langle 2^2, 4^3, 5^1 \right\rangle$.

Esempio 1.6 *Sia dato* $\nu \in \mathbb{MVN}$, $\nu = \left\langle 1^1, 3^1, 4^2 \right\rangle$.

- ν *descrive l'MV set* $\alpha = \{a_1^1, a_2^3, a_3^4, a_4^4\}$ *: l' MV set* α *ha MV cardinalità* ν.
- *Anche l' MV set* $\gamma = \{c_1^1, c_2^4, c_3^3, c_4^4\}$ *ha MV cardinalità* ν .
- α *e* γ *sono MV set* isomorfi .

Esempio 1.7 *Sia dato l' MV set* $\alpha = \{a_1^2, a_2^2, a_3^3\}$.

- *La* MV cardinalità *di* α *è* $\nu = \|\alpha\| = \left\langle 2^2, 3^1 \right\rangle$.
- *La* cardinalità *di* α *è* $n = |\alpha| = |A| = |\{a_1, a_2, a_3\}| = 3$.

Esempio 1.8 *Sia dato* $n = 5$.

- *È possibile una proiezione di* $n \in \mathbb{N}$ *in* $\nu \in \mathbb{MVN}$, $\nu = \left\langle 1^5 \right\rangle$.
- ν *descrive l' MV set* $\alpha = \{a_1^1, a_2^1, a_3^1, a_4^1, a_5^1\}$ *: l' MV set* α *ha MV cardinalità* $\left\langle 1^5 \right\rangle$.
- *La* cardinalità *di* α *è* $|\alpha| = |A| = |\{a_1, a_2, a_3, a_4, a_5\}| = 5 = n$.

Capitolo 2

PRINCIPI DI CALCOLO COMBINATORIO SU MV SET

"Se un uomo che non sa contare fino a quattro trova un quadrifoglio, ha diritto alla fortuna?"

(Stanislaw Lec)

Questo secondo capitolo introduce le prime nozioni, e le prime formule, di calcolo combinatorio nella categoria degli MV set. In particolare, con riferimento alla "tavola pitagorica dell'analisi combinatoria" [1], proveremo a trasporre la prima riga, quella relativa al caso "DD" (palline distinguibili in scatole distinguibili),nella categoria in questione. Procederemo quindi elencando e cercando di spiegare le varie formule per il conteggio di funzioni e funzioni iniettive e suriettive (nei vari casi) tra MV set.
Cercheremo, da qui in avanti, di mantenere la stessa notazione. In particolare:

1. α e γ , come nel primo capitolo, saranno, rispettivamente, dominio e codominio di funzioni tra MV set.

2. Le cardinalità di α e γ saranno $|\alpha| = n$ e $|\gamma| = x$.

3. Le MV cardinalità di α e γ saranno $\|\alpha\| = \nu$ e $\|\gamma\| = \chi$.

4. Il numero di funzioni da α a γ sarà indicato con χ^{ν} .

5. L'insieme delle MV permutazioni di α , ovvero delle MV biezioni da α in se stesso, sarà indicato con $\mathcal{S}_{\alpha}$. Il numero delle MV permutazioni sarà invece $\nu!$.

---

[1] Si veda l'introduzione.

6. Il numero di iniezioni deboli da α a γ si scriverà $((\chi))_\nu$ e verrà chiamato *fattoriale decrescente debole*.

7. Il numero di iniezioni forti da α a γ si scriverà $(\chi)_\nu$ e verrà chiamato *fattoriale decrescente forte*.

Va notato come ogni simbolo utilizzato racchiuda in sé un significato che rievoca quanto studiato nell'analisi combinatoria "classica".

2.1 Conteggio delle MV funzioni

Iniziamo da una semplice osservazione: sotto l'azione di $f : \alpha \to \gamma$ un elemento $a^{km} \in \alpha$ può essere mappato in ogni elemento $c^m \in \gamma$. Essendo f arbitraria, elementi diversi sono trattati in modo indipendente. Consideriamo per esempio $\alpha = \{a_1^{m_1}, \ldots, a_n^{m_n}\}$ e poniamo che $a_1^{m_1}$ possa essere mappato in i_1 elementi di γ, $a_2^{m_2}$ in i_2 elementi di γ, e così via. Allora, per quanto detto, il numero di possibili funzioni da α a γ sarà dato dal prodotto $i_1 i_2 \cdots i_n$. Abbiamo così trovato una prima formula, utile per il computo delle funzioni da un MV set α a un MV set γ, che in simboli può essere espressa come:

$$\chi^\nu = \prod_{a \in \alpha} \sum_{c \in \gamma} [\gamma(c) \text{ div } \alpha(a)]$$

dove $[\gamma(c) \text{ div } \alpha(a)]$ prende valore 1 se è vero che $\gamma(c)$ div $\alpha(a)$ e 0 altrimenti[2]. Da quanto detto, considerando le funzioni per il calcolo di distribuzione di frequenza e cumulante *i*-esimo, descritte nel capitolo precedente, possiamo arrivare alla seguente proposizione:

Proposizione 2.1 *Il numero di MV funzioni da α a γ è dato da*

$$\chi^\nu = \prod_{i \in \mathbb{N}} \left(S_i^\chi\right)^{s_i^\nu} .$$

[2]La notazione $[P]$ indica il valore di verità del predicato P ed è dovuta a D.E. Knuth.

Dimostrazione : Dalle osservazioni fatte all'inizio di questo paragrafo abbiamo che

$$\chi^{\nu} = \prod_{a\in\alpha} \sum_{c\in\gamma} [\gamma(c) \text{ div } \alpha(a)] = \prod_{i\in\mathbb{N}} \left(\sum_{c\in\gamma} [\gamma(c) \text{ div } i] \right)^{s_i^{\nu}} =$$

$$= \prod_{i\in\mathbb{N}} \left(\sum_{j\in\mathbb{N}} s_j^{\chi} [j \text{ div } i] \right)^{s_i^{\nu}} = \prod_{i\in\mathbb{N}} \left(\sum_{j \text{ div } i} s_j^{\chi} \right)^{s_i^{\nu}} = \prod_{i\in\mathbb{N}} \left(S_i^{\chi} \right)^{s_i^{\nu}} .$$

□

Concludiamo il paragrafo con un ulteriore formula utile a contare le permutazioni su un MV set α, ovvero il numero di MV funzioni da α in se stesso. Vale la seguente:

Proposizione 2.2 *Il numero di MV permutazioni di α è*

$$\nu! = \prod_{i\in\mathbb{N}} s_i^{\nu}! .$$

Dimostrazione : Sia A_i , per $i \in \mathbb{N}$, il sottoinsieme di A costituito dagli elementi di A aventi molteplicità i. Una MV permutazione $f : \alpha \to \alpha$ induce una permutazione di A_i, per ogni $i \in \mathbb{N}$. Viceversa, fissare una permutazione di A_i per ogni $i \in \mathbb{N}$ corrisponde a specificare esattamente una MV partizione di α. Dato che vale $|A_i| = s_i^{\nu}$, ne segue quanto affermato nella proposizione.

□

Esempio 2.1 *Siano dati gli MV set* $\alpha = \{a_1^3, a_2^6\}$ *e* $\gamma = \{c_1^1, c_2^2, c_3^3\}$.

- *Osserviamo α e γ. Per mezzo di una MV funzione (si ricordi la definizione):*

  l'elemento a_1^3 *può essere mappato nei soli elementi* c_1^1, c_3^3 ,

  l'elemento a_2^6 *può essere mappato negli elementi* c_1^1, c_2^2, c_3^3 .

  Il numero di possibili funzioni da α a γ sarà quindi pari a $2 \cdot 3 = 6$.

- *Utilizziamo la prima formula che abbiamo ricavato. Secondo tale formula, il numero di MV funzioni da α a γ è:*

  $\chi^{\gamma} = \prod_{a\in\alpha} \sum_{c\in\gamma}[\gamma(c) \text{ div } \alpha(a)] =$

  $= ([3 \text{ div } 1] + [3 \text{ div } 2] + [3 \text{ div } 3])([6 \text{ div } 1] + [6 \text{ div } 2] + [6 \text{ div } 3]) =$

  $= 2 \cdot 3 = 6.$

- *Sfruttiamo, infine, le distribuzioni di frequenza:*

  $\chi^{\gamma} = \prod_{i\in\mathbb{N}} \left(S_i^{\chi}\right)^{s_i^{\gamma}} = 1^0 \cdot 2^0 \cdot 2^1 \cdot 2^0 \cdot 1^0 \cdot 3^1 = 6\,.$

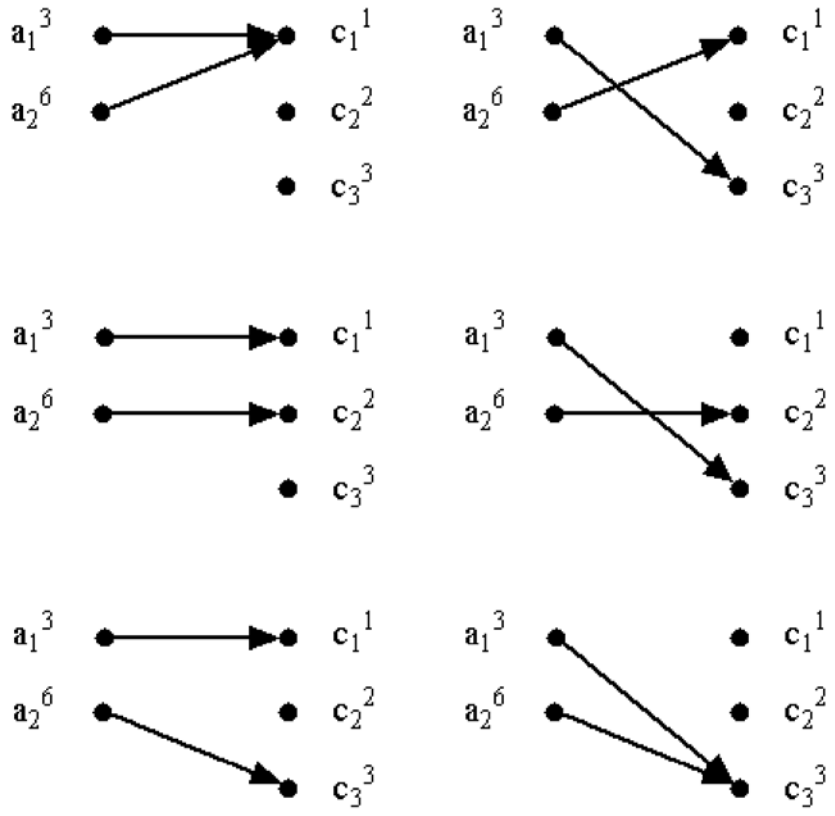

Figura 2.1: Le sei MV funzioni da $\alpha = \{a_1^3, a_2^6\}$ a $\gamma = \{c_1^1, c_2^2, c_3^3\}$.

Esempio 2.2 *Sia dato* $\alpha = \{a_1^2, a_2^2, a_3^4\}$.

- *Il numero di MV permutazioni su* α *è calcolabile con la formula* $\nu! = \prod_{i\in\mathbb{N}} s_i^{\nu}!$ *e vale* $2!1! = 2$.

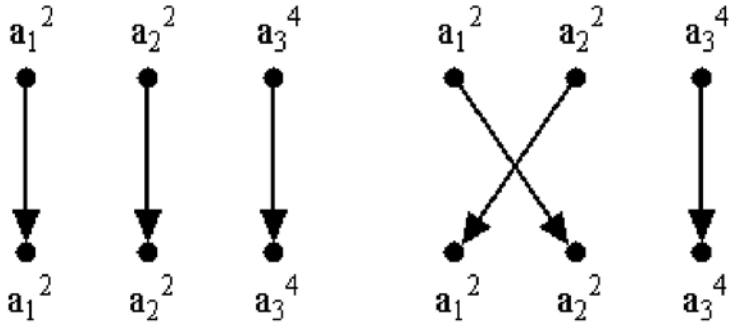

Figura 2.2: Le due MV permutazioni in $\alpha = \{a_1^2, a_2^2, a_3^4\}$.

2.2 Conteggio delle MV iniezioni

Per il calcolo del numero di iniezioni forti tra due MV set α e γ occorre osservare come queste derivino dalle iniezioni tra gli insiemi supporto in modo del tutto simile a come le permutazioni su un insieme α derivano dalle permutazioni sull'insieme supporto. Si ha:

Proposizione 2.3 *Il numero di MV funzioni fortemente iniettive da* α *a* γ *è*

$$(\chi)_{\nu} = \prod_{i\in\mathbb{N}} (s_i^{\chi})_{s_i^{\nu}} .$$

Dimostrazione : Per ogni $i \in \mathbb{N}$, sia A_i il sottoinsieme di A composto da tutti quegli elementi aventi molteplicità i. Analogamente, sia C_i il sottoinsieme di C composto da tutti quegli elementi aventi molteplicità i. Una iniezione forte da α a γ induce un'iniezione $f : A_i \rightarrowtail C_i$, per ogni $i \in \mathbb{N}$. Viceversa, fissando una

iniezione $f : A_i \rightarrowtail C_i$, per ogni $i \in \mathbb{N}$, si identifica una specifica iniezione forte da α a γ. Tenendo conto che $|A_i| = s_i^\nu$ e $|C_i| = s_i^\chi$, ne segue immediatamente il risultato.

□

Il calcolo del numero di MV funzioni fortemente iniettive è, come visto, abbastanza semplice. Un po' più complesso sarà il calcolo delle iniezioni deboli. Per arrivare ad una formula, occorrerà introdurre alcuni nuovi concetti. Primo tra questi, la *relazione di viability* :

Definizione 2.1 *Sia $V_\alpha^\gamma \subseteq A \times C$ la relazione binaria definita da*

$$(a, c) \in V_\alpha^\gamma \quad \Leftrightarrow \quad \gamma(c) \text{ div } \alpha(a) \ .$$

Chiamiamo V_α^γ la relazione di viability *tra α e γ. Se $\tilde{\alpha}$ e $\tilde{\gamma}$ sono MV set tali che $||\tilde{\alpha}|| = \nu$ e $||\tilde{\gamma}|| = \chi$, allora la relazione V_α^γ è isomorfa a $V_{\tilde{\alpha}}^{\tilde{\gamma}}$. Scriviamo quindi V_ν^χ per indicare la relazione di viability tra un ν-set e un χ-set.*

Ricordiamo quindi la definizione di *matching*:

Definizione 2.2 *Il* matching *di un grafo è un sottoinsieme M di archi indipendenti, ovvero tali che nessuna coppia di essi condivida lo stesso nodo.*

Data una qualsiasi relazione binaria R tra insiemi finiti è possibile definire la funzione $\mathcal{M}(R)$ che conta il numero di *matching completi* di R, ovvero, se $R = X \times Y$, il numero di matching di tutto X in un un sottoinsieme di Y.
Siamo così pronti a contare i numero di iniezioni deboli tra due MV set:

Proposizione 2.4 *Il numero di MV funzioni debolmente iniettive tra α e γ è pari al numero di matching completi della relazione di viability V_ν^χ. In simboli,*

$$((\chi))_\nu \ = \ \mathcal{M}(V_\nu^\chi) \ .$$

Dimostrazione : Osservando la definizione di V_ν^χ si può facilmente dedurre che un matching completo di V_ν^χ induce una iniezione debole da α a γ. Sia ora $f : \alpha \to \gamma$ una iniezione debole. Consideriamo la relazione $F \subseteq A \times C$ data dal grafo di $f : A \to C$. Essendo $f : A \to C$ una funzione iniettiva estendibile ad una MV funzione da α a γ, F è un matching (necessariamente completo, in quanto f è una funzione) V_ν^χ.

□

Un esempio ci sarà utile a chiarire quanto detto.

Esempio 2.3 *Siano dati gli MV set* $\alpha = \{a_1^2, a_2^4, a_3^5\}$ *e* $\gamma = \{c_1^2, c_2^2, c_3^5\}$.

- *Ricaviamo dalla definizione la relazione di viability:*

$$V_\alpha^\gamma = \{(a_1, c_1), (a_1, c_2), (a_2, c_1), (a_2, c_2), (a_3, c_3)\}.$$

- *I matching completi della relazione* V_α^γ *sono:*
  $M_1 = \{(a_1, c_1), (a_2, c_2), (a_3, c_3)\}$
  $M_2 = \{(a_1, c_2), (a_2, c_1), (a_3, c_3)\}$.

- *Il numero di funzioni debolmente iniettive da* α *a* γ *è pari al numero di matching completi della relazione* V_α^γ, *quindi* $\mathcal{M}(V_\alpha^\gamma) = 2$.

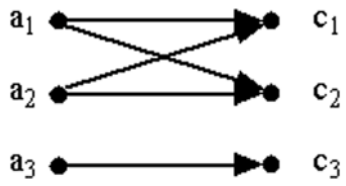

Figura 2.3: Il grafo (bipartito) corrispondente alla relazione V_α^γ .

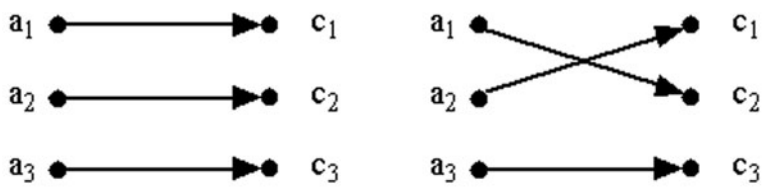

Figura 2.4: I due matching completi della relazione V_α^γ .

Conseguenza di quanto detto è che il problema del "contare il numero di iniezioni deboli tra due MV set" ha la stessa complessità del conteggio del numero di matching completi in un grafo bipartito. Le relazioni di viability sono infatti istanze di grafi semplici[3] bipartiti su insiemi disgiunti A e C, aventi vertici in $A \cup C$ e lati diretti da A a C. In particolare possiamo dimostrare che preso un qualunque grafo di questo tipo esistono due MV set aventi come relazione di viability quella rappresentata dal grafo:

Proposizione 2.5 *Siano $A \cap C = \emptyset$, $R \subseteq A \times C$ (e sia $G = (A \cup C, R)$ un grafo semplice bipartito aventi lati tutti diretti da A a C). Esistono allora gli MV set α e γ tali che $V_\nu^\chi = R$. Di fatto, possono essere scelti dei numeri primi come molteplicità degli elementi di γ e dei numeri* square-free[4] *come molteplicità degli elementi di α.*

Dimostrazione : Sia $p_i \in \mathbb{P} \doteq \{2, 3, 5, 7, \ldots\}$ l'i-esimo numero primo e sia $C = \{c_1, \ldots, c_x\}$. Definiamo $\gamma : C \to \mathbb{P}$ come $\gamma(c_i) = p_i$. Per ogni $a \in A$, poniamo:

$$\alpha(a) = \prod_{c \in \{\, c_j \mid (a,c_j) \in R\}} \gamma(c) \; .$$

[3]Un grafo è semplice se e solo se non contiene autoanelli né lati paralleli.

[4]Chiamiamo *square-free* quei numeri interi i cui fattori primi non compaiano più di una volta.

Supponiamo che $(a_i, c_j) \in R$. Abbiamo allora, per costruzione, che $\gamma(c_j) = p_j$ div $\alpha(a_i)$, quindi $(a_i, c_j) \in V_v^\chi$. Supponiamo poi che $(a_i, c_j) \notin R$. In questo caso p_j non compare tra i fattori primi di $\alpha(a_j)$, per cui p_j non divide $\alpha(a_i)$, e $(a_i, c_j) \notin V_v^\chi$. Abbiamo allora che $V_v^\chi = R$. Per costruzione, $\alpha(a_i)$ è square-free.

□

Esempio 2.4 *Sia data la relazione*

$R = \{(a_1, c_1), (a_2, c_2), (a_2, c_3), (a_3, c_2), (a_3, c_3), (a_4, c_4)\}$,

rappresentata dal grafo semplice bipartito in figura 2.5(a).

- *Costruiamo, con il procedimento descritto nella precedente dimostrazione, i due MV set α e γ. Nel nostro caso è sufficiente porre $|\gamma| = 4$, quindi γ sarà formato da elementi aventi come molteplicità i primi quattro numeri primi. Inoltre: $\alpha(a_1) = \gamma(c_1)$, $\alpha(a_2) = \alpha(a_3) = \gamma(c_2)\cdot\gamma(c_3)$, $\alpha(a_4) = \gamma(c_4)$. Si ha dunque: $\gamma = \{c_1^2, c_2^3, c_3^5, c_4^7\}$, $\alpha = \{a_1^2, a_2^{15}, a_3^{15}, a_4^7\}$.*

- *La relazione di viability tra α e γ è, come desiderato, $V_v^\chi = R$.*

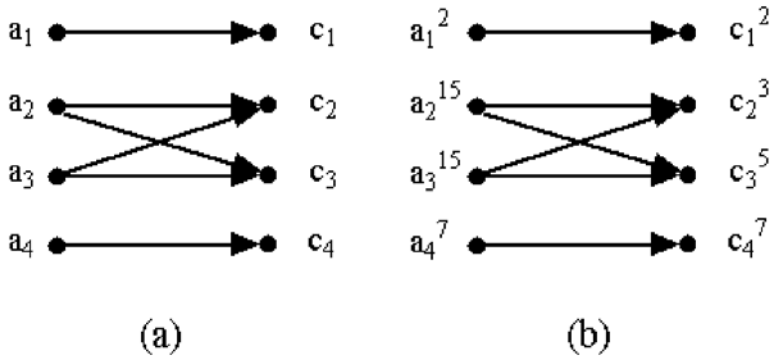

Figura 2.5: Un grafo bipartito (a) e la relazione tra MV set associata (b).

Un ulteriore esempio mostrerà uno specifico caso di conteggio di MV iniezioni.

Esempio 2.5 *Siano dati i due MV set* $\alpha = \{a_1^2, a_2^3\}$ *e* $\gamma = \{c_1^1, c_2^2, c_3^3\}$.

- *Utilizzando la formula descritta, ricaviamo il numero di MV funzioni fortemente iniettive tra* α *e* γ *. Si ottiene:* $(\chi)_\nu = \prod_{i\in\mathbb{N}}(s_i^\chi)_{s_i^\nu} = 1$.

- *Ricaviamo ora, dalla definizione data, la relazione di viability:* $V_\nu^\chi = \{(a_1, c_1), (a_1, c_2), (a_2, c_1), (a_2, c_3)\}$.

- *I matching completi della relazione* V_ν^χ *sono:* $M_1 = \{(a_1, c_1), (a_2, c_3)\}$, $M_2 = \{(a_1, c_2), (a_2, c_3)\}$, $M_3 = \{(a_1, c_2), (a_2, c_1)\}$.

- *Il numero di funzioni debolmente iniettive da* α *a* γ *è pari al numero di matching completi della relazione* V_ν^χ*, quindi* $\mathcal{M}(V_\nu^\chi) = 3$.

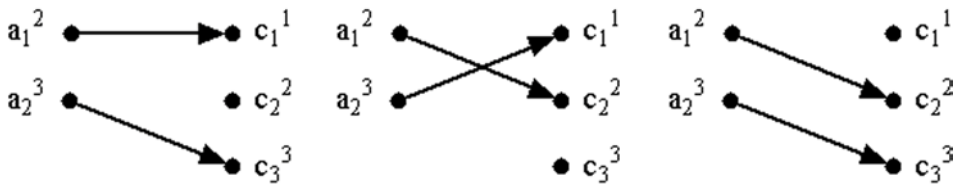

Figura 2.6: Le tre MV iniezioni deboli da $\alpha = \{a_1^2, a_2^3\}$ a $\gamma = \{c_1^1, c_2^2, c_3^3\}$.

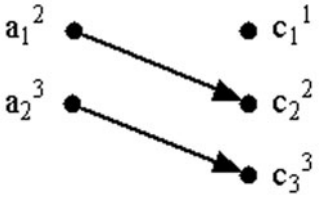

Figura 2.7: L'unica MV iniezioni forte da $\alpha = \{a_1^2, a_2^3\}$ a $\gamma = \{c_1^1, c_2^2, c_3^3\}$.

2.3 MV partizioni

Le partizioni venivano utilizzate, nel caso classico, per il conteggio delle funzioni suriettive. Allo stesso modo utilizzeremo le MV partizioni nella categoria degli MV set.

Introduciamo prima un nuovo concetto, il *prodotto cartesiano* di due MV set, che non servirà per il computo delle MV funzioni suriettive, ma che verrà utilizzato nei capitoli seguenti e sarà presente nel pacchetto *Mathematica*® sviluppato.

Definizione 2.3 *Il prodotto cartesiano di due MV set α e γ, indicato con $\alpha \times \gamma$, è, a meno di isomorfismi, l' MV set $\alpha \times \gamma : A \times C \to \mathbb{N}$ tale che $\alpha \times \gamma((a,c)) = mcm\{\alpha(a), \gamma(c)\}$, per ogni $(a,c) \in A \times C$.*

Esempio 2.6 *Siano dati gli MV set $\alpha = \{a_1^2, a_2^3\}$ e $\gamma = \{c_1^1, c_2^2\}$.*

- *Il prodotto cartesiano di α e γ è:*
  $\alpha \times \gamma = \{(a_1, c_1)^2, (a_1, c_2)^2, (a_2, c_1)^3, (a_2, c_2)^6\}$.

Possiamo tornare alle MV partizioni. Nel caso classico, uno dei modi di introdurre le partizioni di un insieme è guardare la partizione di un insieme A come un insieme di *fibre*, i blocchi della partizione, indotte su A dalla suriezione $f : A \twoheadrightarrow C$. Faremo in modo che il rapporto tra MV partizioni e MV suriezioni sia analogo.

Sia quindi $f : \alpha \to \gamma$ una MV funzione debolmente suriettiva. $f : A \to C$, funzione tra gli insiemi supporto, induce una partizione su A. Una fibra $B \subseteq A$ sarà composta da tutti quegli elementi di A che la MV funzione f mappa in un unico elemento $c_B \in C$. Questo elemento avrà molteplicità $\gamma(c_B)$ e noi diciamo che la fibra B ha molteplicità $\gamma(c_B)$. Inoltre, diciamo che ogni elemento $a \in B$ ha una molteplicità indotta dalla MV funzione f che sarà pari a quell'unico numero naturale m_a tale che $m_a \gamma(c_B) = \alpha(a)$.

Definizione 2.4 *Una* MV partizione debole *di α è un MV set π in cui gli elementi dell'insieme supporto $\beta_1, \beta_2, ..., \beta_k$, $k \in \mathbb{N}$, sono MV set, gli MV blocchi di π, che soddisfano le seguenti condizioni:*

1. *Se B_i = Dom β_i, $i \in \{1, \ldots, k\}$, allora $\{B_1, \ldots, B_k\}$ è una partizione di A = Dom α in k blocchi non vuoti.*

2. *Per ogni $a \in \beta_i$, $\beta_i(a)\pi(\beta_i) = \alpha(a)$.*

Una partizione in *k* blocchi è spesso chiamata *k*-partizione. Nella categoria degli MV set k viene rimpiazzato da un MV numero naturale $\kappa = \|\pi\|$.

Definizione 2.5 *Sia π una MV partizione debole di α e sia $\kappa \in \mathbb{MVN}$. Diciamo che π è una κ*-partizione debole*, o una MV partizione debole di α* in κ MV blocchi*, se e solo se $\|\pi\| = \kappa$.*

In altre parole, κ descrive il numero e le molteplicità degli MV blocchi di una κ-partizione debole.

Esempio 2.7 *Sia dato $\alpha = \{a_1^2, a_2^3, a_3^4, a_4^5, a_5^6\}$.*

- $\pi_1 = \left\{\{a_4^5\}^1, \{a_1^1, a_3^2\}^2, \{a_2^1, a_5^2\}^3\right\}$ *è una* $\left\langle 1^1, 2^1, 3^1\right\rangle$ *- partizione debole di* α .

- $\pi_2 = \left\{\{a_2^3, a_4^5\}^1, \{a_1^1, a_3^2, a_5^3\}^2\right\}$ *è una* $\left\langle 1^1, 2^1\right\rangle$ *- partizione debole di* α .

- $\pi_3 = \left\{\{a_2^3\}^1, \{a_4^5\}^1, \{a_1^1, a_3^2, a_5^3\}^2\right\}$ *è una* $\left\langle 1^2, 2^1\right\rangle$ *- partizione debole di* α .

- $\pi_4 = \left\{\{a_1^2, a_2^3, a_3^4, a_4^5, a_5^6\}^1\right\}$ *è una* $\left\langle 1^1\right\rangle$ *- partizione debole di* α .

Come nella teoria classica veniva introdotto il numero di Stirling di seconda specie, che conta il numero di partizioni di un insieme, introduciamo ora il suo analogo.

Definizione 2.6 *Chiamiamo* MV numero di Stirling debole di seconda specie , *e lo indichiamo con*

$$\left\{\!\left\{ \begin{matrix} \nu \\ \kappa \end{matrix} \right\}\!\right\},$$

il numero di κ-partizioni deboli di un ν-set.

Passiamo ora al caso delle suriezioni forti, andando a definire la MV partizione indotta da una MV funzione fortemente suriettiva.

Definizione 2.7 *Una MV partizione debole* $\Pi = \{\beta_1^{t_1}, \ldots, \beta_k^{t_k}\}$ *di un MV set* α *è una* MV partizione forte *se e solo se, per* $i \in \{1, \ldots, k\}$ *, vale:*

$$t_i = \text{MCD}\ \{\ \alpha(a) \mid a \in B_i\ \}\ .$$

Indicheremo una MV partizione forte con la lettera maiuscola Π , i suoi blocchi con β_i e gli insiemi supporto dei blocchi con B_i. Diremo che Π è una κ-*partizione forte*, o una MV partizione forte di α in $\kappa \in \mathbb{MVN}$ MV blocchi, se e solo se $\|\Pi\| = \kappa$.

Esempio 2.8 *Sia dato* $\alpha = \{a_1^1, a_2^3, a_3^4, a_4^8\}$.

- $\Pi_1 = \left\{\{a_1^1, a_2^3\}^1, \{a_3^1, a_4^2\}^4\right\}$ *è una* $\left\langle 1^1, 4^1 \right\rangle$ *- partizione forte di* α *, perché* $\text{MCD}\ \{1, 3\} = 1$ *e* $\text{MCD}\ \{4, 8\} = 4$.

- $\Pi_2 = \left\{\{a_1^1, a_2^3, a_4^8\}^1, \{a_3^1\}^4\right\}$ *è una* $\left\langle 1^1, 4^1 \right\rangle$ *- partizione forte di* α *, perché* $\text{MCD}\ \{1, 3, 8\} = 1$.

Come già fatto per le MV partizioni deboli, anche in questo caso possiamo introdurre la seguente:

Definizione 2.8 *Chiamiamo* MV numero di Stirling forte di seconda specie *, e lo indichiamo con*

$$\left\{ \begin{matrix} \nu \\ \kappa \end{matrix} \right\},$$

il numero di κ-partizioni forti di un ν-set.

Esempio 2.9 *Sia dato* $\alpha = \{a_1^2, a_2^3, a_3^4\}$.

- *L'insieme supporto di* α *è* $C = \{a_1, a_2, a_3\}$. *Le sue partizioni sono in numero pari a* $S(3,1) + S(3,2) + S(3,3) = 1 + 3 + 1 = 5$. *In particolare, le* 5 *partizioni di C sono:*

  $\{\{a_1, a_2, a_3\}\}$, $\{\{a_1, a_2\}, \{a_3\}\}$, $\{\{a_1, a_3\}, \{a_2\}\}$, $\{\{a_2, a_3\}, \{a_1\}\}$,

  $\{\{a_1\}, \{a_2\}, \{a_3\}\}$.

- *Alle partizioni trovate possono essere associate MV partizioni: i blocchi delle partizioni classiche saranno i domini dei blocchi delle MV partizioni. Elenchiamo dunque le partizioni su* α *, specificando tra parentesi se si tratta di partizioni deboli o forti:*

$\left\{\{a_1^2, a_2^3, a_3^4\}^1\right\}$ *(FORTE)*	$\left\{\{a_1^2\}^1, \{a_2^3\}^1, \{a_3^2\}^2\right\}$ *(debole)*
$\left\{\{a_1^2, a_2^3\}^1, \{a_3^4\}^1\right\}$ *(debole)*	$\left\{\{a_1^2\}^1, \{a_2^3\}^1, \{a_3^1\}^4\right\}$ *(debole)*
$\left\{\{a_1^2, a_2^3\}^1, \{a_3^2\}^2\right\}$ *(debole)*	$\left\{\{a_1^2\}^1, \{a_2^1\}^3, \{a_3^4\}^1\right\}$ *(debole)*
$\left\{\{a_1^2, a_2^3\}^1, \{a_3^1\}^4\right\}$ *(FORTE)*	$\left\{\{a_1^2\}^1, \{a_2^1\}^3, \{a_3^2\}^2\right\}$ *(debole)*
$\left\{\{a_1^2, a_3^4\}^1, \{a_2^3\}^1\right\}$ *(debole)*	$\left\{\{a_1^2\}^1, \{a_2^1\}^3, \{a_3^1\}^4\right\}$ *(debole)*
$\left\{\{a_1^2, a_3^4\}^1, \{a_2^1\}^3\right\}$ *(debole)*	$\left\{\{a_1^1\}^2, \{a_2^3\}^1, \{a_3^4\}^1\right\}$ *(debole)*
$\left\{\{a_1^1, a_3^2\}^2, \{a_2^3\}^1\right\}$ *(debole)*	$\left\{\{a_1^1\}^2, \{a_2^3\}^1, \{a_3^2\}^2\right\}$ *(debole)*
$\left\{\{a_1^1, a_3^2\}^2, \{a_2^1\}^3\right\}$ *(FORTE)*	$\left\{\{a_1^1\}^2, \{a_2^3\}^1, \{a_3^1\}^4\right\}$ *(debole)*
$\left\{\{a_2^3, a_3^4\}^1, \{a_1^2\}^1\right\}$ *(debole)*	$\left\{\{a_1^1\}^2, \{a_2^1\}^3, \{a_3^4\}^1\right\}$ *(debole)*
$\left\{\{a_2^3, a_3^4\}^1, \{a_1^1\}^2\right\}$ *(FORTE)*	$\left\{\{a_1^1\}^2, \{a_2^1\}^3, \{a_3^2\}^2\right\}$ *(debole)*
$\left\{\{a_1^2\}^1, \{a_2^3\}^1, \{a_3^4\}^1\right\}$ *(debole)*	$\left\{\{a_1^1\}^2, \{a_2^1\}^3, \{a_3^1\}^4\right\}$ *(FORTE)*

- *In totale abbiamo trovato* 22 *MV partizioni deboli di* α *. Tra queste, le MV partizioni forti sono* 5 .

2.4 Conteggio delle MV suriezioni

Le definizioni introdotte nel precedente paragrafo e, in particolare, l'introduzione degli MV numeri di Stirling di seconda specie, ci permetteranno ora di ottenere delle formule per il conteggio di MV funzioni suriettive. Per arrivare al risultato desiderato, vediamo una nuova definizione e due lemmi che ci saranno di supporto.

Definizione 2.9 *Siano date due MV funzioni debolmente suriettive* f_1 *e* f_2*, da* α *a* γ *.* f_1 *e* f_2 *si dicono* equivalenti sulle fibre *se e solo se esiste una permutazione* $q : \gamma \to \gamma$ *tale che* $q \circ f_1 = f_2$ *.*

È chiaro che l'*equivalenza sulle fibre* è una relazione di equivalenza sull'insieme delle suriezioni deboli $f : \alpha \to \gamma$. Indichiamo con $[f]$ la classe di equivalenza di f.

Lemma 2.1 *Sia* $F = \{ [f] \mid f : \alpha \to \gamma$ *f* è una suriezione debole $\}$ *e sia* $P = \{ \pi \mid \pi$ è una MV partizione debole di $\alpha \}$ *. La mappa che porta* $[f]$ *nella MV partizione indotta da f è una biezione tra* F *e* P *.*

Dimostrazione : È sufficiente provare che se $[f_1] = [f_2]$ allora f_1 e f_2 inducono la stessa MV partizione di α . Osserviamo che f_1 e f_2 inducono la stessa partizione (booleana) sull'insieme supporto di α . Inoltre, una MV permutazione preserva le molteplicità, quindi f_1 e f_2 inducono le stesse molteplicità sulla partizione di A .

□

Lemma 2.2 *Sia* $F = \{ [f] \mid f : \alpha \to \gamma$ *f* è una suriezione forte $\}$ *e sia* $P = \{ \Pi \mid \Pi$ è una MV partizione forte di $\alpha \}$ *. La mappa che porta* $[f]$ *nella MV partizione indotta da f è una biezione tra* F *e* P *.*

Dimostrazione : La dimostrazione è analoga a quella affrontata per il lemma 2.1.

□

Possiamo finalmente "contare" le MV suriezioni. Otteniamo infatti, come immediata conseguenza dei lemmi precedenti, le seguenti proposizioni:

Proposizione 2.6 *Il numero di MV funzioni debolmente suriettive da α a γ è*

$$\left\{\left\{ \begin{matrix} \nu \\ \kappa \end{matrix} \right\}\right\} \chi !$$

Proposizione 2.7 *Il numero di MV funzioni fortemente suriettive da α a γ è*

$$\left\{ \begin{matrix} \nu \\ \kappa \end{matrix} \right\} \chi !$$

Esempio 2.10 *Consideriamo gli MV set* $\alpha = \{a_1^2, a_2^3, a_3^4\}$ *e* $\gamma = \{c_1^1, c_2^2\}$. *Si tenga presente quanto visto nell'esempio 2.9* .

- *La cardinalità di* γ *è* $\chi = \langle 1^1, 2^1 \rangle$.
- *Osserviamo le MV partizioni di* α : $\left\{\{a_1^2, a_2^3\}^1, \{a_3^2\}^2\right\}$, $\left\{\{a_1^1, a_3^2\}^2, \{a_2^3\}^1\right\}$, $\left\{\{a_2^3, a_3^4\}^1, \{a_1^1\}^2\right\}$ *sono* $\langle 1^1, 2^1 \rangle$ *- partizioni, tra queste* $\left\{\{a_2^3, a_3^4\}^1, \{a_1^1\}^2\right\}$ *è una* $\langle 1^1, 2^1 \rangle$ *- partizioni forte* .
- *Grazie alle due proposizioni appena viste e osservando che esiste una sola permutazione su γ deduciamo che ci sono 3 MV suriezioni deboli e una sola MV suriezione forte da α a γ .*

Esempio 2.11 *Consideriamo lo stesso MV set* $\alpha = \{a_1^2, a_2^3, a_3^4\}$ *e sia questa volta* $\gamma = \{c_1^1, c_2^1, c_3^2\}$.

- *La cardinalità di* γ *è* $\chi = \langle 1^2, 2^1 \rangle$.

- $\left\{\{a_1^2\}^1, \{a_2^3\}^1, \{a_3^2\}^2\right\}$ *e* $\left\{\{a_1^1\}^2, \{a_2^3\}^1, \{a_3^4\}^1\right\}$ *sono le uniche* $\langle 1^2, 2^1\rangle$ *- partizioni (deboli) di* α .

- *Ci sono 2 MV permutazioni su* γ. *Il numero di MV suriezioni deboli da* α *a* γ *è* $\left\{\left\{ {\gamma \atop \kappa} \right\}\right\} \chi ! = 2 \cdot 2 = 4$. *Non esistono invece MV suriezioni forti.*

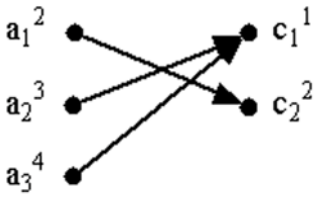

Figura 2.8: L'unica MV suriezione forte da $\alpha = \{a_1^2, a_2^3, a_3^4\}$ a $\gamma = \{c_1^1, c_2^2\}$.

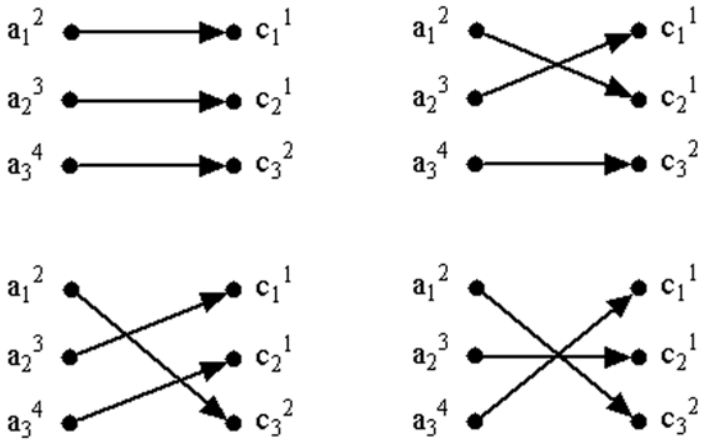

Figura 2.9: Le 4 MV suriezioni deboli da $\alpha = \{a_1^2, a_2^3, a_3^4\}$ a $\gamma = \{c_1^1, c_2^1, c_3^2\}$.

Capitolo 3

UN PACCHETTO PER LO STUDIO DI MV SET

"Spesso si sente dire che la matematica consiste principalmente nel dimostrare teoremi. È forse il principale lavoro di uno scrittore quello di scrivere frasi?"

(Gian-Carlo Rota)

Mathematica è un sistema per... "fare matematica".
Scopo di questo capitolo è progettare, all'interno di questo sistema, un ambiente per lo studio degli MV set. Procederemo, dopo una breve panoramica su *Mathematica*, allo sviluppo delle funzioni che andranno a comporre il pacchetto *mvset.m* e estenderanno le possibilità del sistema di partenza permettendo agli utenti di effettuare le operazioni tra MV set analizzate nei primi capitoli. Le descrizioni delle funzioni saranno accompagnate da utili esempi che mostreranno come utilizzarle e, in alcune casi, descriveranno i passi compiuti nella fase di progetto.
Il codice dell'intero pacchetto è disponibile in appendice A. Sono poi disponibili, in appendice B, svariati esempi di utilizzo delle funzioni, che permetteranno, se osservati in parallelo a queste pagine, di meglio comprendere i paragrafi che seguono.

3.1 Programmare in Mathematica

Mathematica è dotato di un linguaggio di programmazione ad alto livello, scritto in una variante del linguaggio *C*, che permette di eseguire calcoli numerici e simbolici e include una vasta libreria di funzioni che coprono innumerevoli

ambiti della matematica. Il linguaggio è interpretato e permette di utilizzare *Mathematica* sia come una potente calcolatrice (si scrive una funzione e si attende il risultato) sia come ambiente di sviluppo; nonostante ciò è possibile, grazie ad un compilatore interno, compilare le funzioni utilizzate in maniera intensiva al fine di aumentare la velocità di calcolo. Molte altre sono le possibilità offerte da questo sistema, per esempio la capacità di generare codice *MathML* o TEX e di trattare funzioni scritte in *C* o *FORTRAN*.

Mathematica è composto essenzialmente da due parti: il *kernel* e il *front end*. Il *front end* è l'interfaccia verso l'utente e viene chiamato *notebook*. Il *kernel* è il motore di calcolo e comunica con il *front end* attraverso *mathlink*. È inoltre possibile sfruttare il *kernel* come motore di calcolo all'interno di altri linguaggi, grazie ad opportune librerie di funzioni.

Mathematica supporta diversi paradigmi di programmazione ed è possibile scegliere caso per caso se è più conveniente adottare tecniche di programmazione procedurale o orientata agli oggetti o, ancora, di programmazione funzionale o rule-based. Anche se è molto diffusa la programmazione procedurale, raramente è l'approccio migliore in un ambiente simbolico come questo e spesso tecniche di programmazione funzionale e rule-based portano a risolvere i problemi in maniera più semplice ed efficiente. Un breve cenno a queste tecniche diventa quindi d'obbligo:

La programmazione funzionale è uno stile di programmazione che consiste in funzioni applicate a argomenti e funzioni. Sorprendentemente questa tecnica permette di scrivere programmi molto complessi; inoltre, ci permette di trattare efficientemente liste e strutture dati in genere, manipolando questi oggetti interamente piuttosto che, come si usa nella programmazione procedurale, prendere ogni elemento e trattarlo separatamente.

Un programma rule-based consiste in una collezione di definizioni di funzioni dello stesso tipo, in cui ogni definizione è valida per un dato valore del parametro in ingresso. La programmazione rule-based può essere il modo più semplice per affrontare alcuni problemi, perché l'unica cosa che occorre fare è scrivere quella collezione di regole che permette al sistema di scegliere che funzione applicare nei vari casi. Questa tecnica applicata in casi abbastanza problematici (numerose condizioni da analizzare sui parametri ricevuti) evita di scrivere programmi "spaghetti".

3.2 Il linguaggio di Mathematica

Descriviamo brevemente com'è e come si comporta il linguaggio utilizzato per sviluppare il pacchetto *mvset.m*. Partiamo dalle basi: un'espressione *Mathematica* è una stringa di simboli nella forma

$$\square\,[\,\square, \square, \square, \ldots\,]$$

dove con il simbolo $\square$ indichiamo un segnaposto che può essere sostituito da simboli atomici (numeri, simboli e stringhe) o altre espressioni. Un'espressione può quindi essere annidata. Questa è la rappresentazione interna di ogni cosa in *Mathematica*, e questo rende la programmazione semplice e potente. Il comando `FullForm` ci permetterà di ricavare la rappresentazione in termini di espressione, ovvero la rappresentazione interna, di ogni istruzione. Ad esempio, scrivendo `FullForm[a+b]` si otterrà in risposta `Plus[a,b]`. Un programma non è quindi altro che una espressione e tutto quello che *Mathematica* fa è osservare le regole contenute nel programma e valutare la dovuta espressione.

In *Mathematica* è possibile utilizzare variabili. Assegnando un valore a una variabile è possibile scegliere se questa prenderà da subito e per tutta l'esecuzione il valore calcolato dell'espressione assegnata oppure se questo valore

verrà calcolato ad ogni utilizzo della variabile. È inoltre possibile l'utilizzo di liste e tabelle multidimensionali, l'utilizzo di regole che permettano la sostituzione di valori a simboli contenuti nelle espressioni, la riscruttura di funzioni esistenti e la programmazione di nuove funzioni. Un metodo per scrivere funzioni complesse è l'utilizzo di *moduli* che permettano di conservare le variabili localmente, senza modificare il valore di eventuali variabili omonime esterne. Ancora, è possibile utilizzare la ricorsione e tecniche di programmazione dinamica, e sono presenti funzioni `Do` , `For` e `While` per l'esecuzione di cicli.
Come si evince dalla rapidità di questa descrizione, non è nostra intenzione descrivere nel dettaglio il linguaggio, ma vogliamo comunque introdurre le particolari funzioni, utilizzate frequentemente in seguito, che permettono l'approccio alla risoluzione di un problema con una tecnica di programmazione funzionale.

Map è una delle più importanti funzione utilizzate nella programmazione funzionale e permette di mappare una funzione sugli argomenti di un'altra funzione. Ad esempio, l'espressione `Map[ Log,{ a, b, c}]` , dove `{a,b,c}` è una lista, restituirà `{Log[a],Log[b],Log[c]}`, mentre l'espressione `Map[Log,Plus[a,b]]` restituirà `Plus[Log[a],Log[b]]` ovvero `Log[a]+Log[b]` . `Map[g,f[x,y]]` può anche essere scritto, abbreviando, come `g/@f[x,y]`.

Apply sostituisce l'intestazione di una espressione con una nuova intestazione. Per esempio, `[Apply[List,a+b+c]]` restituirà la lista `{a,b,c}`. Anche per questa funzione esiste una forma abbreviata; nel nostro caso avremmo potuto scrivere `List @@ (a+b+c)`.

Thread è una funzione particolarmente utile quando si vogliono fare sostituzioni multiple in una sola volta. L'espressione `Thread[{a,b}->{1,2}]`

ci permetterà infatti di ottenere `{a->1,b->2}` . Più in generale, la funzione `Thread[f[{x,y},{u,v}]` darà come risultato `{f[x,u],f[y,v]}`.

Outer fa essenzialmente le stesse cose di `Map`, ma è applicata a funzioni di più variabili. In particolare, `Outer[Plus,{a,b,c},{x,y}]` restituisce `{{a + x, a + y}, {b + x, b + y}, {c + x, c + y}}`.

3.3 Rappresentazione di MV set

Possiamo iniziare a descrivere passo passo le fasi dello sviluppo del pacchetto *mvset.m*. La prima decisione da prendere riguarda la rappresentazione di un MV set. Ricordiamo che un MV set è solitamente rappresentato come:

$$\alpha = \{a_1^{m_1}, \ldots, a_n^{m_n}\}$$

dove $A = \{a_1, \ldots, a_n\}$ è l'insieme supporto e $\alpha(a_i) = m_i$, per $i \in \{1, \ldots, n\}$. Occorre osservare che per il calcolo delle funzioni studiate è sufficiente conoscere la molteplicità m_i degli elementi dell'MV set, ovvero non è necessario conoscere come è esattamente un MV set, ma basta conoscerlo *a meno di isomorfismi*. Gli MV set $\alpha = \{a_1^2, a_2^3\}$ e $\gamma = \{c_1^2, c_2^3\}$ saranno per le nostre funzioni la stessa cosa. A questo punto, ci saranno due diversi modi per definire, a meno di isomorfismi, un MV set: potremo elencare le molteplicità dei suoi elementi oppure descrivere la distribuzione di frequenza delle molteplicità. Nel primo caso l'MV set sarà rappresentato, in *Mathematica*, da una lista di molteplicità e verrà assegnato con

$$\alpha = \{m_1, m_2, \ldots, m_n\}.$$

Nel secondo caso la sua rappresentazione sarà come lista di distribuzioni di frequenza, dove al primo posto ci sarà il numero degli elementi aventi molteplicità 1, al secondo il numero di quelli aventi molteplicità 2 e così via. Si potrà quindi

assegnare un MV set a una variabile α con il comando

$$\alpha = \{s_1, s_2, \ldots\}.$$

Questo è il punto di partenza: ogni funzione che sarà programmata lavorerà su liste di questo tipo. Possiamo notare che la prima rappresentazione permette di risalire direttamente alla cardinalità dell'insieme supporto di un MV set `a`, uguale alla lunghezza della lista e calcolabile, in *Mathematica*, con la funzione `Length[a]`. La seconda rappresentazione è invece espressione dell'MV cardinalità dell'MV set.
È utile, nei due casi visti, sviluppare funzioni per testare se una lista è la rappresentazione corretta di un MV set. Per quanto riguarda la prima forma di rappresentazione si dovrà verificare se tutti gli elementi della lista sono interi e positivi, nella seconda forma sono invece accettabili come elementi anche valori nulli. Le funzioni per effettuare questo controllo saranno:

```
MVTestQ[L_]:=And[Head[L]===List,Apply[And,Flatten
            [{Map[IntegerQ,L],Map[Positive, L]}]]]
MVTestFQ[L_]:=And[Head[L]===List,Apply[And,Flatten
          [{Map[IntegerQ,L],Map[NonNegative,L]}]]]
```

Analizziamole brevemente. Diciamo innanzitutto che il nome assegnato a queste funzioni segue uno schema tipico nella programmazione in *Mathematica*: il nome definisce il comportamento della funzione, le parole che lo compongono sono scritte per esteso, unite tra loro e iniziate da una lettera maiuscola. Noi inizieremo con *MV* tutti i nomi di funzioni che si occupano di MV set e termineremo per *F* i nomi di quelle funzioni che agiscono su distribuzioni di frequenza. Prestiamo attenzione alla prima funzione. L'utilizzo di *map* non fa altro che applicare i test *IntegerQ* e *Positive* ad ogni elemento della lista ricevuta in ingresso. La funzione *Flatten* "appiattisce" poi quello che riceve, in

pratica trasformando una lista di liste in una lista semplice formata da tutti gli elementi coinvolti. Il risultato che si ottiene è una lista di valori di verità che saranno tutti uguali a *true* solo se tutti gli elementi sono interi (test effettuato da *IntegerQ*) e positivi (*Positive*). Sostituendo alla funzione *List* la funzione *And* si ricava una congiunzione logica dei valori di verità, che, insieme al controllo del fatto che l'ingresso sia una lista, fornisce in uscita il valore *true* solo se il test è verificato, ovvero se l'input rappresenta un MV set. Analoga è la seconda funzione.
Si vuole,a questo punto, mettere a disposizione dell'utente un metodo per passare da una rappresentazione ad un'altra. Per fare questo, sono state sviluppate le seguenti funzioni:

```
MVFrequency[a_?MVTestQ]:=If[a==={},{0},
          Table[Count[a,i],{i,Max[a]}]]
MVMultiplicity[af_?MVTestFQ]:=Flatten[Table
     [Table[i,{j,af[[i]]}],{i,Length[af]}]]
```

Quando una funzione verrà richiamata, *Mathematica* procederà a verificare l'input per mezzo delle funzioni viste precedentemente e solo se il test risulta soddisfatto verrà valutata l'espressione seguente. I test `?MVTestQ` e `?MVTestFQ` verranno utilizzati in ogni funzione che sarà sviluppata: non sarà quindi possibile eseguire le funzioni del pacchetto su un qualunque ingresso, ma solo quando l'argomento ricevuto dalla funzione è un MV set.
Analizziamo le funzioni `MVFrequency` e `MVMultiplicity` . La prima riceve in ingresso un MV set rappresentato come lista di molteplicità e crea una lista avente nella posizione *i*-esima in numero di elementi, contati da `Count`, aventi molteplicità *i*. Per farlo, è utilizzata la funzione `Table[expr, {i, imax}]` che genera la lista dei valori di `expr` per *i* che va da 1 a *imax*. La seconda riceve invece una lista di distribuzioni di frequenza e, preso ogni valore s_i nella

posizione i, non fa altro che scrivere nuove liste ricopiando per s_i volte il valore di i. Queste liste saranno gli elementi di una lista che la funzione `Flatten` provvederà a rendere semplice (monodimensionale).
Possiamo effettuare un ulteriore controllo su MV set: il test di uniformità. Un MV set si dice infatti *u-uniforme* se tutti i suoi elementi hanno la stessa molteplicità. Dato un tipo di dato MV set, questo sarà u-uniforme se la lista che lo rappresenta contiene tutti elementi uguali, oppure, nel caso sia utilizzata la distribuzione di frequenza, se solo un valore della lista è diverso da 0.
Le funzioni che eseguono questa verifica sono:

```
MVIsUniform[a_?MVTestQ]:=Apply[Equal,a]
MVIsUniformF[af_?MVTestFQ]:=
        (Length[Select[af,#!=0 &]]==1||
         Length[Select[af,#!=0 &]]==0)
```

Siamo pronti per entrare nell'ambiente *Mathematica* e iniziare a testare queste prime funzioni. Apriamo quindi un *notebook* vuoto. Una volta aperto un foglio di lavoro, per accedere alle funzioni del pacchetto occorre importarlo con il comando

```
<<mvset.m
```

A questo punto si può procedere all'utilizzo di tutte le funzioni descritte in questo capitolo. Ricordiamo che il nome delle funzioni deve essere scritto per esteso e che il parametro passato alle funzioni deve essere un MV set: in caso contrario *Mathematica* non darà nessun messaggio di errore, perché, non conoscendo nessuna funzione avente quel nome e quei parametri di ingresso, considererà l'espressione appena scritta come una nuova espressione. Per ogni funzione è disponibile un *help* in linea, che descrive brevemente le caratteristiche della funzione, i parametri in ingresso e il valore ritornato. Per richiamare

l'*help* sono disponibili due comandi: `?NomeFunzione`, che si limita a descrivere la funzione e `??NomeFunzione`, che aggiunge la definizione della funzione ed è molto utile per osservare il comportamento di funzioni dinamiche.
Importiamo allora, prima di tutto, il pacchetto *mvset.m*:

```
In:=  << mvset.m
```

Mathematica non dà nessuna risposta: vuol dire che tutto è andato bene. Assegnamo quindi un MV set a una variabile, che chiameremo `a` [1] :

```
In:=  a = {1,1,2,3}
Out=  {1,1,2,3}
```

Possiamo ora verificare se `a` è realmente un MV set. Procederemo poi assegnando alla variabile `af` la trasformazione di `a` in un MV set rappresentato come distribuzione di frequenza, per poi testare il contenuto di `af`. Infine, proveremo le funzioni per verificare l'uniformità. Quello che si ottiene è:

```
In:=  af = MVFrequency[a]
Out=  {2,1,1}
In:=  MVMultiplicity[af]
Out=  {1,1,2,3}
In:=  MVTestQ[a]
Out=  True
In:=  MVTestFQ[af]
Out=  True
In:=  MVTestQ[{1,0,2}]
Out=  False
In:=  MVIsUniform[a]
```

[1] Nell'utilizzare *Mathematica* chiameremo, per comodità, gli MV set con lettere minuscole, anziché con le solite lettere greche α e γ .

```
Out=  False
In:=  MVIsUniform[{4,4,4}]
Out=  True
In:=  ?MVTestQ
Out=  MVTestQ[L] restituisce True se L è una lista di
interi positivi.
In:=  ??MVIsUniform
Out=  MVIsUniform[α] restituisce True se α è un MV set
u-uniforme, False altrimenti.
Out=  MVIsUniform[α_?MVTestQ]:=Equal@@α
```

3.4 Prodotto cartesiano tra MV set

Affrontiamo come primo problema quello del calcolo del prodotto cartesiano tra due MV set. Ricordiamo che il prodotto cartesiano di due MV set α e γ, indicato con $\alpha \times \gamma$, è l' MV set $\alpha \times \gamma : A \times C \to \mathbb{N}$ tale che:
$\alpha \times \gamma((a,c)) = mcm\{\alpha(a), \gamma(c)\}$, per ogni $(a,c) \in A \times C$.
La creazione di questo nuovo MV set avverrà mediante la funzione `Table`, che inserirà in una nuova lista i valori dei minimi comuni multipli tra ogni elemento di α e ogni elemento di γ. La funzione prodotta è :

```
MVCartesianProduct[a_?MVTestQ,c_?MVTestQ]:=
   Flatten[Table[LCM[a[[i]], c[[j]]],
   {i, Length[a]},{j,Length[c]}]]
```

Apriamo allora il nostro *notebook* e andiamo a testare questa nuova funzione. Osserveremo, in questo caso, anche i risultati intermedi, per seguire così le fasi dello sviluppo della funzione. Questi passi saranno ripetuti in seguito solo nei casi in cui questo possa aiutare a capire meglio il significato delle funzioni

utilizzate. Precisiamo inoltre che il simbolo % viene sostituito, in *Mathematica*, con l'output dell'ultimo comando eseguito.

```
In:= ?MVCartesianProduct
Out= MVCartesianProduct[α,γ] restituisce l'MV Set α×γ,
prodotto Cartesiano di α e γ.
In:= a={1,2,2}
Out= {1,2,2}
In:= c={2,3}
Out= {2,3}
In:= LCM[a[[1]],c[[1]]]
Out= 2
In:= Table[LCM[a[[i]],c[[j]]],{i,Length[a]},
{j,Length[c]}]
Out= {{2,3},{2,6},{2,6}}
In:= Flatten[%]
Out= {2,3,2,6,2,6}
In:= MVCartesianProduct[a,c]
Out= {2,3,2,6,2,6}
```

3.5 Conteggio di MV funzioni e permutazioni

Entriamo nella parte centrale dello sviluppo del pacchetto, affrontando prima di tutto il problema del conteggio di MV funzioni. Ricordiamo le formule ricavate:

$$\chi^{\nu} = \prod_{a\in\alpha} \sum_{c\in\gamma} [\gamma(c) \text{ div } \alpha(a)]$$

$$\chi^{\nu} = \prod_{i\in\mathbb{N}} \left(S_i^{\chi}\right)^{s_i^{\nu}} .$$

La prima formula sarà utilizzabile quando in ingresso si ha la lista delle molteplicità degli elementi, la seconda quando si utilizza la distribuzione di frequen-

za.

Veniamo al primo caso. Il controllo del valore del predicato $[\gamma(c)$ div $\alpha(a)]$ può essere effettuato con la funzione Mod, verificando se il resto della divisione intera è nullo. È sufficiente quindi porre questa espressione all'interno di funzioni per il calcolo di sommatoria e produttoria, correttamente indicizzate. Possiamo quindi scrivere:

```
MVFunctions [a_?MVTestQ,c_?MVTestQ] :=
   Product[Sum[If[Mod[a[[i]],c[[j]]]==0,1,0],
   {j,Length[c]}],{i,Length[a]}]
```

Nel secondo caso occorre calcolare il valore del cumulante *i*-esimo S_i, pari a $\sum_{j \text{ div } i} s_j^{\lambda}$, dove gli s_j sono elementi della lista ricevuta in ingresso. È semplice, a questo punto, scrivere la funzione per il conteggio di MV funzioni, che sarà:

```
MVFunctionsF[af_?MVTestFQ,bf_?MVTestFQ]:=
   Product[If[af[[i]]==0,1,(Sum[
   If[Mod[i,j]==0,bf[[j]],0],{j,Length[bf]}])
   ^af[[i]]],{i,Length[af]}]
```

Andiamo a confrontare, con l'ausilio di *Mathematica*, i risultati prodotti da queste due funzioni :

```
In:=  a = {2, 3, 4, 5, 5, 6}
Out=  {2, 3, 4, 5, 5, 6}
In:=  c = {1, 2, 3, 4}
Out=  {1, 2, 3, 4}
In:=  af = MVFrequency[a]
Out=  {0, 1, 1, 1, 2, 1}
In:=  cf = MVFrequency[c]
Out=  {1, 1, 1, 1 }
```

```
In:= MVFunctions[a, c]
Out= 36
In:= MVFunctionsF[af, cf]
Out= 36
```

Possiamo introdurre un'ulteriore funzione, per il calcolo delle MV permutazioni su un MV set. Sfruttando le distribuzioni di frequenza, possiamo calcolare il valore di $\nu! = \prod_{i\in\mathbb{N}} s_i^{\nu}!$ per mezzo della funzione:

```
MVPermutationsF[af_?MVTestFQ] :=
   Product [ af[[i]]! , {i, Length[af]}]
```

Nel caso l'ingresso fosse fornito come lista di molteplicità ci limitiamo a trasformarlo nella seconda forma. Per il computo di MV permutazioni si utilizzerà in questo caso la funzione:

```
MVPermutations[a_?MVTestQ] :=
   MVPermutationsF [MVFrequency[a]]
```

Con riferimento all'ultimo esempio visto, andiamo allora a calcolare il numero di MV permutazioni sull'MV set a:

```
In:= MVPermutations[a]
Out= 2
```

3.6 Conteggio di MV iniezioni forti

Ricordiamo la formula per il calcolo del numero di MV funzioni fortemente iniettive tra due MV set:

$$(\chi)_{\nu} = \prod_{i\in\mathbb{N}} (s_i^{\chi})_{s_i^{\nu}} \ .$$

Quello che ci serve è una funzione per il calcolo del fattoriale decrescente $(x)_n = x(x-1)\cdots(x-n+1)$. Ne scriveremo una nuova, sfruttando la funzione `Pochhammer`, inclusa tra le librerie di base di *Mathematica*, che calcola il fattoriale crescente $\langle x\rangle_n = x(x+1)\cdots(x+n-1)$. Chiameremo questa nuova funzione `DFactorial`:

```
DFactorial [x_Integer, n_Integer] :=
            Pochhammer[x - n + 1, n]
```

`DFactorial` , come evidente, non fa altro che calcolare un fattoriale crescente, sfruttando il fatto che $(x)_n = \langle x-n+1\rangle_n$.
La funzione sviluppata per il calcolo del numero di MV iniezioni forti, ricevuti in ingresso due MV set come distribuzione di frequenza delle molteplicità, è:

```
MVStronglyInjectiveFunctionsF
  [af_?MVTestFQ,bf_?MVTestFQ]:=Module[{nbf},If
  [(Length[bf]<Length[af])&&(af[[Length[af]]]!=0),
  0,nbf=PadRight[bf,Length[af]];Product[DFactorial
  [nbf[[i]],af[[i]]],{i, Length[af]}]]]
```

Proviamo ad analizzare questa funzione. Osserviamo, innanzitutto, che è stato utilizzato un modulo. `Module[{x, y, ... }, expr]` permette di definire delle variabili `x` e `y` che all'interno delle espressioni `expr` verranno trattate come locali. L'utilizzo della funzione `PadRight[list, n]` permette inoltre di portare una lista alla lunghezza `n` accorciandola oppure riempendola di zeri alla destra. In pratica, la funzione `MVStronglyInjectiveFunctionsF`, dopo aver effettuato un controllo sulla possibilità di esistenza di iniezioni forti (se l'MV set `af` possiede elementi con molteplicità maggiore della molteplicità massima in `bf` non possono esistere MV iniezioni forti tra i due MV set, perché non sarebbe possibile preservare le molteplicità), produce a partire da una lista `bf` la

nuova lista nbf della stessa lunghezza di af . La produttoria viene poi calcolata utilizzando nbf anziché bf, in modo da permettere il calcolo dei fattoriali decrescenti per ogni valore in af.
Andiamo a studiare in un foglio di lavoro il funzionamento della funzione. Iniziamo assegnando un valore a due MV set:

```
In:= af = MVFrequency[{1, 2, 3}]
Out= {1, 1, 1}
In:= bf = MVFrequency[{1, 2, 2, 3, 4, 4, 5}]
Out= {1, 2, 1, 2, 1}
```

Procediamo verificando il funzionamento delle istruzioni interne al modulo, ovvero il test e l'istruzione PadRight :

```
In:= Length[bf]<Length[af]&&af[[Length[af]]]!=0
Out= False
In:= PadRight[bf, Length[af]]
Out= {1, 2, 1}
```

Infine, invochiamo la funzione sviluppata:

```
In:= MVStronglyInjectiveFunctionsF[af, bf]
Out= 2
```

Non ci resta che sviluppare la funzione MVStronglyInjectiveFunctions : utilizzeremo la funzione che riceve in input MV set rappresentati come distribuzioni di frequenza delle molteplicità, premurandoci di trasformare i nostri argomenti nel formato corretto.

```
MVStronglyInjectiveFunctions
    [a_?MVTestQ,c_?MVTestQ]:=
    MVStronglyInjectiveFunctionsF
    [MVFrequency[a],MVFrequency[c]]
```

3.7 Conteggio di MV iniezioni deboli

Come visto nei precedenti capitoli, il calcolo del numero di funzioni debolmente iniettive tra due MV set non è semplice come i casi affrontati sino ad ora. Il nodo della questione è il calcolo del permanente, necessario per arrivare al risultato finale. Occorre quindi sviluppare due funzioni intermedie prima di passare alle iniezioni deboli: una si preoccuperà di produrre una matrice rappresentante la relazione di viability tra due MV set, la seconda avrà validità del tutto generale (non dipenderà cioè dalla categoria in cui stiamo lavorando) e calcolerà il permanente di una qualsiasi matrice $n \times m$.
Partiamo dalla prima; chiameremo questa funzione `ViabilityMatrix` :

```
ViabilityMatrix[a_?MVTestQ,c_?MVTestQ]:=
   Array[If[Mod[a[[#1]],c[[#2]]]==0,1,0]
   &,{Length[a],Length[c]}]

ViabilityMatrixF[af_?MVTestFQ,cf_?MVTestFQ]:=
          ViabilityMatrix[MVMultiplicity[af],
                          MVMultiplicity[cf]]
```

Quello che facciamo è produrre una matrice di dimensione pari a $n \times x$ (dove, dati gli MV set a e c, parametri della funzione, $n = |a|$ e $x = |c|$) per mezzo della funzione `Array`. Riempiamo poi questa matrice ponendo ogni suo elemento $V_{i,j}$ pari a $[a[i] \text{ div } c[j]]$.
Guardiamo come lavora la funzione prodotta:

```
In:=  a = {4, 6, 8}
Out=  {4, 6, 8}
In:=  c = {2, 3, 4, 6, 6, 8}
Out=  {2, 3, 4, 6, 6, 8}
In:=  v = ViabilityMatrix[a, c]
```

```
Out=  {{1,0,1,0,0,0}, {1,1,0,1,1,0}, {1,0,1,0,0,1}}
In:=  MatrixForm[%]
Out=
```

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Il calcolo del permanente della matrice risultante ci darà il numero di matching completi della relazione di viability, ovvero il numero di MV iniezioni deboli tra i due MV set.
Veniamo quindi al calcolo del permanente. Diversi sono gli algoritmi reperibili per effettuare questo calcolo, molti dei quali si occupano di dare una soluzione a questo problema per approssimazione. A noi serve un algoritmo esatto e che lavori bene nel nostro caso, in cui la matrice coinvolta è composta da soli 0 e 1. Abbiamo quindi preferito scrivere una funzione nuova:

```
Permanent[{}] := 0;
Permanent[{{sequence___}}]:=Plus[sequence];
Permanent[v_?MatrixQ]:=
   Module[{pivot,temp},temp=Map[Count[#,0]&,v];
   pivot=Min[Flatten[Position[temp,Max[temp]]]];
   (Permanent[v]=Sum[If[v[[pivot]][[i]]==0,0,
   v[[pivot]][[i]]*Permanent[Map[Drop[#,{i}]&,
   Drop[v,{pivot}]]]],Length[v[[1]]]})];
```

Il nostro algoritmo lavora in pratica scegliendo una riga della matrice, prendendo un elemento non nullo di questa e calcolando il prodotto tra questo e il permanente del minore ottenuto dalla matrice eliminando la riga e la colonna in cui è presente l'elemento considerato. La funzione è quindi ricorsiva e per ogni elemento della riga scelta richiamerà se stessa sui minori fino a giungere ad un

vettore riga (composto da un solo elemento nel caso l'ingresso sia una matrice quadrata) per poi calcolare la somma degli elementi del vettore e, infine, procedere a ritroso ricalcolando i prodotti. Cerchiamo di analizzarla in dettaglio, a partire dalla terza espressione, quella che riceve come parametro in ingresso una matrice (la verifica è effettuata per mezzo della funzione `MatrixQ`) .
All'interno del modulo sono utilizzate le variabili `temp` e `pivot`. Alla prima variabile viene assegnato, per mezzo dell'istruzione `Map[Count[#,0]&,v]`, un vettore avente nella posizione *i* il numero di zeri presenti nella riga *i*-esima della matrice in ingresso. La *funzione anonima* `Count[#,0]&` si comporta esattamente come una normale funzione in cui il parametro verrà sostituito al simbolo `#`. La funzione `Map` mapperà questa funzione sulla matrice `v` producendo come risultato un vettore `{Count[v[1],0],...}`. Alla variabile pivot verrà poi assegnata la posizione della riga avente il maggior numero di zeri, calcolata su `temp` tramite l'espressione `Min[Flatten[Position[temp,Max[temp]]]]`. L'utilizzo di una riga pivot velocizza notevolmente l'algoritmo, perché nel passo successivo la ricorsione avverrà solo per i valori non nulli della riga, mentre ai valori nulli verrà immediatamente assegnato valore 0. L'istruzione successiva effettuerà la ricorsione: il valore del permanente della matrice `v` in ingresso sarà uguale alla somma dei valori non nulli della riga pivot moltiplicati per i permanenti dei rispettivi minori.
Ripercorriamo quanto detto in un *notebook Mathematica*, osservando i risultati via via prodotti dalle funzioni all'interno del modulo. Consideriamo i due MV set dell'ultimo caso visto e calcoliamo, prima di tutto, il valore delle variabili usate internamente al modulo:

```
In:=  temp = Count[#, 0] & /@ v
Out=  {4, 2, 3}
In:=  Position[temp, Max[temp]]
Out=  {{1}}
```

```
In:= Flatten[%]
Out= {1}
In:= Min[%]
Out= 1
In:= pivot = Min[Flatten[Position[temp, Max[temp]]]]
Out= 1
```

Osserviamo quindi come viene selezionato il minore della matrice:

```
In:= MatrixForm[v]
Out=
```

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

```
In:= MatrixForm[Drop[v, {pivot}]]
Out=
```

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

```
In:= Drop[#1, {1}] & /@ %
Out=
```

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

All'interno del modulo viene richiamata la funzione `Permanent` su questo minore e il valore restituito è moltiplicato per il valore `v[[pivot]][[1]]` . Questo avviene per ogni `i` tale che `v[[pivot]][[i]]!=0` . I valori ottenuti dai prodotti vengono quindi sommati. Nel caso che stiamo considerando si ottiene:

```
In:= Permanent[v]
Out= 13
```

Va notato l'utilizzo di meccanismi dinamici in questa funzione. Il valore del permanente non è infatti semplicemente calcolato e restituito, ma, insieme ai valori dei permanenti di tutti i minori, viene memorizzato come definizione di una nuova funzione. In pratica ad ogni esecuzione della funzione `Permanent` verranno accumulate nuove informazioni. Successivamente, quando *Mathematica* incontrerà la chiamata della funzione su matrici o minori già noti, restituirà immediatamente il risultato conosciuto.

L'*help* di *Mathematica* ci può essere d'aiuto nell'osservare la dinamicità della funzione. Ricordiamo infatti che il comando `??NomeFunzione` restituisce, oltre alla descrizione della funzione, la sua definizione. Apriamo allora un foglio di lavoro e vediamo cosa succede alla funzione `Permanent` dopo un suo utilizzo.

```
In:= ?? Permanent
Out=
Permanent[M] restituisce il permanente della matrice M.
Permanent[{}] := 0
Permanent[{{sequence__}}] := +sequence
Permanent[v_?MatrixQ] := Module[...]
In:= v
Out= {{1,0,1,0,0,0},{1,1,0,1,1,0},{1,0,1,0,0,1}}
In:= Permanent[v]
Out= 13
In:= ?? Permanent
Out=
Permanent[M] restituisce il permanente della matrice M.
Permanent[{}] := 0
Permanent[{{1, 0, 1, 1, 0}, {0, 1, 0, 0, 1}}] = 6
Permanent[{{1,1,1,1,0},{1,0,0,0,1}}]=7
```

```
Permanent[{{1,0,1,0,0,0},{1,1,0,1,1,0},{1,0,1,0,0,1}}]=13
Permanent[{{sequence__}}] := +sequence
Permanent[v_?MatrixQ] := Module[...]
```

Non ci resta altro che spiegare le due definizioni di `Permanent` che abbiamo inizialmente tralasciato. La prima assegna valore 0 al permanente di un vettore nullo, la seconda restituisce la somma degli elementi di un vettore monodimensionale. La programmazione della funzione è quindi *rule-based*: tre diverse definizioni sono date per la stessa funzione, e il differente valore dell'argomento ricevuto determina quale espressione verrà effettivamente eseguita.

Possiamo passare al calcolo del numero di MV iniezioni deboli tra due MV set che, arrivati a questo punto, è francamente banale. È infatti sufficiente richiamare la funzione `ViabilityMatrix` sui due MV set e calcolare il permanente della matrice ottenuta:

```
MVWeaklyInjectiveFunctions
      [a_?MVTestQ,c_?MVTestQ] :=
      Permanent[ViabilityMatrix[a,c]]

MVWeaklyInjectiveFunctionsF
      [af_?MVTestQ,cf_?MVTestQ] :=
      Permanent[ViabilityMatrixF[af,cf]]
```

Utilizzando gli MV set `a` e `c` visti prima, possiamo verificare come il calcolo della funzione `MVWeaklyInjectiveFunctions` su di essi dia lo stesso risultato ottenuto con il calcolo del permanente:

```
In:=  a = {4, 6, 8}
Out=  {4, 6, 8}
In:=  c = {2, 3, 4, 6, 6, 8}
Out=  {2, 3, 4, 6, 6, 8}
```

```
In:= MVWeaklyInjectiveFunctions[a,c]
Out= 13
```

Riguardo alla funzione `Permanent`, e di conseguenza anche alle funzioni per il conteggio di MV iniezioni deboli che la utilizzano, occorre osservare che può essere necessario, per produrre risultati, incrementare il valore del parametro di sistema `$RecursionLimit` . Per farlo, è sufficiente il semplice assegnamento: `$RecursionLimit = nuovo_valore`.

3.8 Calcolo del permanente

Nel precedente paragrafo abbiamo visto come viene effettuato il calcolo del numero di funzioni debolmente iniettive tra due MV set e abbiamo detto che la parte centrale di questo calcolo risiede nella funzione `Permanent`.
Vogliamo ora staccarci per un attimo dalla descrizione del pacchetto per soffermarci sulla funzione per il calcolo del permanente. Proporremo una diversa soluzione per risolvere il medesimo problema e cercheremo di fare un confronto tra questa funzione e quella inserita nel pacchetto.
La soluzione alternativa ci viene da *Mathworld* ed è proposta da Eric W. Weisstein[2]. Andiamo a presentarla e ad osservare il suo comportamento all'interno di un *notebook*:

```
In:= AltPermanent[m_?MatrixQ]:=
With[{v=Array[x,Length[m]]},
Coefficient[Times @@ (m.  v), Times @@ v]]
In:= try={{1,1,0,0},{1,1,1,0},{1,1,0,0},{1,1,0,1}}
Out= {{1,1,0,0},{1,1,1,0},{1,1,0,0},{1,1,0,1}}
In:= MatrixForm[try]
```

[2]Eric W. Weisstein. "Permanent". From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Permanent.html ©1999-2004 Wolfram Research, Inc.

Out=

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

In:= **v = Array[x, Length[try]]**

Out= {x[1],x[2],x[3],x[4]}

In:= **MatrixForm[v]**

Out=

$$\begin{pmatrix} x[1] \\ x[2] \\ x[3] \\ x[4] \end{pmatrix}$$

In:= **MatrixForm[try. v]**

Out=

$$\begin{pmatrix} x[1] + x[2] \\ x[1] + x[2] + x[3] \\ x[1] + x[2] \\ x[1] + x[2] + x[4] \end{pmatrix}$$

In:= **Times @@ (try. v)**

Out= (x[1]+x[2])^2(x[1]+x[2]+x[3])(x[1]+x[2]+x[4])

In:= **Expand[%]**

Out= x[1]^4+4x[1]^{3}x[2]+6x[1]^{2}x[2]^2+4x[1]x[2]^3+x[2]^4+
x[1]^{3}x[3]+3x[1]^{2}x[2]x[3]+3x[1]x[2]^{2}x[3]+x[2]^{3}x[3]+
x[1]^{3}x[4]+3x[1]^{2}x[2]x[4]+3x[1]x[2]^{2}x[4]+x[2]^{3}x[4]+
x[1]^{2}x[3]x[4]+2x[1]x[2]x[3]x[4]+x[2]^{2}x[3]x[4]

In:= **Times @@ v**

Out= x[1] x[2] x[3] x[4]

```
In:= Coefficient[Times @@ (try.  v), Times @@ v]
Out= 2
In:= AltPermanent[try]
Out= 2
In:= Permanent[try]
Out= 2
```

Descriviamo, in breve, il comportamento della funzione di Weisstein. Innanzitutto si prende un vettore colonna $(x[1], x[2], \ldots, x[n])$, dove n è la dimensione della matrice. Si calcola poi il prodotto (riga per colonna) tra la matrice di partenza e questo vettore. Le componenti del vettore risultante vengono quindi moltiplicate tra loro ottenendo un polinomio nelle variabili $x[1], \ldots, x[n]$: il coefficiente del termine $x[1]x[2]\cdots x[n]$ del polinomio, ovvero del termine in cui compaiono tutte le variabili, sarà il permanente della matrice data.
La funzione `AltPermanent`, come la quasi totalità degli algoritmi disponibili, lavora solamente su matrici quadrate, mentre la nostra funzione considera il permanente di matrici rettangolari. Nel nostro caso serviva infatti, data una relazione $R = X \times Y$, calcolare il numero di matching di tutto X in un sottoinsieme di Y e proprio per questo motivo abbiamo esteso la definizione di permanente. A questo punto, per effettuare dei confronti tra le funzioni viste, potremmo modificare la definizione di Weisstein in:

```
PermanentNew[m_?MatrixQ] :=
  With[{v=Array[x,Length[m[[1]]]]},
  Apply[Plus,Coefficient[Times@@(m. v),
  Map[Apply[Times,#]&,KSubsets[v,Length[m]]]]]]
```

Preferiamo però, su queste pagine, evitare l'utilizzo di questa variante, e, più semplicemente, limitare le funzionalità della nostra `Permanent` al caso di matrici quadrate.

Possiamo iniziare il nostro confronto. La “competizione” sarà puramente empirica e verrà sfruttata la funzione *Mathematica* `Timing`, che restituisce una lista del tipo `{t,r}`, dove `t` è il tempo di esecuzione della funzione ed `r` è il risultato. Proponiamo qui solo un paio di esempi estrapolati dai test effettuati: il primo utilizza una matrice `m1` di dimensione 22×22, generata a partire dalla funzione `ViabilityMatrix`, il secondo una matrice `m2` di pari dimensioni più povera di zeri. I due test sono stati effettuati in maniera indipendente, evitando cioè di aiutare *Mathematica* facendogli eseguire una funzione di cui avrebbe conosciuto, almeno in parte, il risultato. Alla fine di ogni test abbiamo ricalcolato il permanente delle matrici con la funzione `Permanent` del nostro pacchetto, per poter apprezzare le qualità dei meccanismi dinamici.

```
In:= MatrixForm[ViabilityMatrix[m1={2,2,4,1,12,15,12,
8,8,2,2,4,3,12,16,14,15,17,20,25,30,29},{1,2,2,1,3,5,3,
2,1,1,7,17,5,12,8,8,1,6,10,25,1,2}]]
```

Out=

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0
\end{pmatrix}$$

In:= {**Length[m1], Length[m1[[1]]]**}

Out= {22, 22}

In:= **Timing[Permanent[m1]]**

Out= {4.757 Second, 2220825600}

In:= **Timing[AltPermanent[m1]]**

Out= {8.372 Second, 2220825600}

```
In:= Timing[Permanent[m1]]
Out= {0.  Second, 2220825600}
In:= MatrixForm[m2=ViabilityMatrix[{2,2,4,1,12,15,12,
8,8,2,2,4,3,12,16,14,15,17,20,25,30,29},{1,1,1,1,1,5,3,
2,1,1,7,17,5,12,8,8,1,6,10,25,1,2}]]
Out= ...
In:= {Length[m2], Length[m2[[1]]]}
Out= {22, 22}
In:= Timing[Permanent[m2]]
Out= {2.293 Second, 26649907200}
In:= Timing[AltPermanent[m2]]
Out= {11.657 Second, 26649907200}
In:= Timing[Permanent[m2]]
Out= {0.  Second, 26649907200}
```

3.9 Conteggio di MV partizioni

In questo paragrafo svilupperemo funzioni in grado di calcolare il numero di MV partizioni, deboli e forti, di un MV set α in κ blocchi, dove κ è la MV cardinalità di un MV set γ. Le funzioni riceveranno in ingresso i due MV set α e γ e restituiranno un numero intero. Partiremo, in questo caso, dalle definizioni di MV partizioni e dalle loro proprietà, studiate nel paragrafo 2.3 .
Nel seguito verranno utilizzate le funzioni `Permutations` della libreria standard di *Mathemathica* e `KSetPartitions`, contenuta nel pacchetto *combinatorica*, presente in tutte le distribuzioni di *Mathematica* dalla versione 4 in poi. `KSetPartitions[set, k]` restituisce la lista delle partizioni di un insieme in k blocchi. `Permutations[list]` genera la lista di tutte le possibili permutazioni degli elementi in `list`.

Cominciamo dal calcolo delle MV partizioni deboli. Come già fatto nei

precedenti paragrafi, osserviamo la funzione sviluppata e proviamo ad analizzarla:

```
MVWeakPartitions[a_?MVTestQ,c_?MVTestQ]:=
   Module[{P,pi,x},x=Length[c];
   P=KSetPartitions[a, x];pi=Permutations[c];
   Sum[If[Sum[Apply[Plus,Mod[P[[j]][[k]],
   pi[[i]][[k]]]],{k,x}]==0,1,0],
   {i, Length[pi]},{j,Length[P]}]]
```

`MVWeakPartitions` riceve in ingresso i due MV set `a` e `c`. Viene utilizzato un modulo, contenente le variabili locali `x` , `P` e `pi`. Alla variabile `x` viene assegnata la cardinalità di `c` , la variabile `pi` sarà la lista di tutte le permutazioni degli elementi di `c` e la variabile `P` conterrà tutte le permutazioni degli elementi di `a` in `x` blocchi. Verrà poi considerata la i-esima permutazione presente in `pi` e contato il numero di partizioni in `P` che sono MV partizioni di `a` in `|c|` blocchi: in questi casi tutti gli elementi del k-esimo blocco della partizione di `a` divideranno l'elemento in posizione k nella partizione considerata (la i-esima) di `c`, per ogni k. Si effettuerà cioè un controllo esaustivo su tutti i casi candidati ad essere MV partizioni deboli dell'MV set `a` in `|c|` blocchi, contando tutti i test che hanno avuto esito positivo. Questa soluzione non sembra certo essere la migliore, ma numerose prove che vedevano coinvolti MV set di dimensioni non eccessive hanno dimostrato che la soluzione adottata è pienamente accettabile.

Possiamo dunque verificare, come fatto nel paragrafo precedente, il comportamento delle espressioni interne al modulo.

```
In:=  a={2,2,3,4,6}
Out=  {2, 2, 3, 4, 6}
In:=  c={2,2,3}
```

```
Out=  {2, 2, 3}
In:=  x = Length[c]
Out=  3
In:=  P = KSetPartitions[a, x]
Out=
{{{2},{2},{3,4,6}},{{2},{2,3},{4,6}},{{2},{2,4,6},{3}},
{{2},{2,3,4},{6}},{{2},{2,6},{3,4}},{{2},{2,3,6},{4}},
{{2},{2,4},{3,6}},{{2,2},{3},{4,6}},{{2,3},{2},{4,6}},
{{2,4,6},{2},{3}},{{2,2},{3,4},{6}},{{2,3,4},{2},{6}},
{{2,6},{2},{3,4}},{{2,2},{3,6},{4}},{{2,3,6},{2},{4}},
{{2,4},{2},{3,6}},{{2,2,3},{4},{6}},{{2,4},{2,3},{6}},
{{2,6},{2,3},{4}},{{2,2,4},{3},{6}},{{2,3},{2,4},{6}},
{{2,6},{2,4},{3}},{{2,2,6},{3},{4}},{{2,3},{2,6},{4}},
{{2,4},{2,6},{3}}}
In:=  pi = Permutations[c]
Out=  {{2,2,3},{2,3,2},{3,2,2}}
```

Abbiamo visto cosa viene assegnato alle variabili interne, consideriamo ora un passo del ciclo presente nella sommatoria più esterna, assegnando un valore a `i` e `j`, e vediamo cosa succede:

```
In:=  i = 1
Out=  1
In:=  j = 3
Out=  3
In:=  pi[[i]]
Out=  {2,2,3}
In:=  P[[j]]
Out=  {{2},{2,4,6},{3}}
```

```
In:= Table[Mod[P[[3]][[k]],pi[[1]][[k]]],{k,x}]
Out= {{0}, {0, 0, 0}, {0}}
In:=
Table[Apply[Plus,Mod[P[[3]][[2]],pi[[1]][[2]]]],{k, x}]
Out= {0, 0, 0}
In:=
Sum[Apply[Plus,Mod[P[[3]][[k]],pi[[1]][[k]]]],{k,x}]
Out= 0
In:= If[% == 0, 1, 0]
Out= 1
```

Come si può vedere, l'esempio considerato è uno dei casi che contribuirà alla sommatoria esterna con valore 1. Il risultato dell'esecuzione della funzione sarà il conteggio di tutti questi casi:

```
In:= MVWeakPartitions[a, c]
Out= 10
```

Il metodo utilizzato per contare le MV partizioni forti è analogo a quello visto. Va solamente sostituita, in ogni controllo effettuato, la condizione che verifica che "tutti gli elementi del k-esimo blocco della partizione di a dividano l'elemento in posizione k nella partizione considerata (la i-esima) di c" con la condizione che impone che "il massimo comun divisore degli elementi del k-esimo blocco della partizione di a divida l'elemento in posizione k nella partizione considerata (la i-esima) di c". La funzione prodotta è la seguente:

```
MVStrongPartitions[a_?MVTestQ,c_?MVTestQ]:=
   Module[{P,pi,x},x=Length[c];
   P=KSetPartitions[a, x];pi=Permutations[c];
   Sum[If[Apply[And,Table[Apply[GCD,
```

```
P[[j]][[k]]==pi[[i]][[k]],{k, x}]],1,0],
{i,Length[pi]},{j,Length[P]}]]
```

Osserviamo in un foglio di lavoro *Mathematica* i passi compiuti da questa funzione.

```
In:= P[[3]]
Out= {{2}, {2, 4, 6}, {3}}
In:= Table[Apply[GCD, P[[3]][[k]]], {k, x}]
Out= {2, 2, 3}
In:= pi[[1]]
Out= {2, 2, 3}
In:= Table[Apply[GCD,P[[3]][[k]]]==pi[[1]][[k]],{k, x}]
Out= {True, True, True}
In:= Apply[And, %]
Out= True
In:= If[%, 1, 0]
Out= 1
```

Il risultato che si otterrà è:

```
In:= MVStrongPartitions[a, c]
Out= 7
```

3.10 Conteggio di MV suriezioni

Ricordando che il numero di MV funzioni debolmente suriettive da α a γ è

$$\left\{\left\{ \begin{matrix} \nu \\ \kappa \end{matrix} \right\}\right\} \chi !$$

e che il numero di MV funzioni fortemente suriettive da α a γ è

$$\left\{ \begin{matrix} \nu \\ \kappa \end{matrix} \right\} \chi !$$

si capisce come le funzioni MVWeakPartitions e MVStrongPartitions , unite alla già descritta MVPermutations, siano sufficienti per calcolare in modo semplice il numero di MV suriezioni, deboli e forti, tra due MV set.
Presenteremo quindi le funzioni sviluppate per questo caso senza descriverle, poiché la loro comprensione sarà immediata.

```
MVWeaklySurjectiveFunctions
   [a_?MVTestQ,c_?MVTestQ]:=
   MVWeakPartitions[a,c]*MVPermutations[c]

MVWeaklySurjectiveFunctionsF
   [a_?MVTestFQ, c_?MVTestFQ] :=
   MVWeakPartitions[MVMultiplicity[a],
   MVMultiplicity[c]]*MVPermutationsF[c]

MVStronglySurjectiveFunctions
   [a_?MVTestQ,c_?MVTestQ]:=
   MVStrongPartitions[a,c]*MVPermutations[c]

MVStronglySurjectiveFunctionsF
   [a_?MVTestFQ, c_?MVTestFQ] :=
   MVStrongPartitions[MVMultiplicity[a],
   MVMultiplicity[c]]*MVPermutationsF[c]
```

Un ultimo esempio descriverà l'utilizzo di queste funzioni in *Mathematica*:

```
In:=  a = {2, 2, 3, 4, 6}
Out=  {2, 2, 3, 4, 6}
In:=  c = {2, 2, 3}
Out=  {2, 2, 3}
In:=  MVWeakPartitions[a, c]
```

```
Out=  10

In:=  MVPermutations[c]

Out=  2

In:=  MVWeaklySurjectiveFunctions[a, c]

Out=  20

In:=  af = MVFrequency[a]

Out=  {0, 2, 1, 1, 0, 1}

In:=  cf = MVFrequency[c]

Out=  {0, 2, 1}

In:=  MVWeaklySurjectiveFunctionsF[af, cf]

Out=  20

In:=  MVStronglySurjectiveFunctions[a, c]

Out=  14

In:=  MVStronglySurjectiveFunctionsF[af, cf]

Out=  14
```

Appendice A

IL PACCHETTO MVSET.M

Verrà di seguito riportato, per esteso, il codice del pacchetto ‘mvset.m’. La descrizione del pacchetto, del suo sviluppo, delle funzioni disponibili e delle basi teoriche su cui si fonda sono argomenti trattati all’interno di questa tesi. Nonostante ciò, per favorire una lettura del codice, sono stati mantenuti tutti i commenti inseriti, comprese le brevi introduzioni teoriche alle funzioni. Inoltre, il codice è qui riportato, sempre allo scopo di facilitarne la comprensione, nel formato grafico del foglio di lavoro di *Mathematica*® anziché nel formato testuale del file che verrà realmente utilizzato per l’importazione.

Le funzioni che è possibile calcolare con questo pacchetto sono, in ordine alfabetico:

DFactorial

MVCartesianProduct

MVFrequency

MVFunctions - MVFunctionsF

MVIsUniform - MVIsUniformF

MVMultiplicity

MVPermutations - MVPermutationsF

MVStronglyInjectiveFunctions - MVStronglyInjectiveFunctionsF

MVStronglySurjectiveFunctions - MVStronglySurjectiveFunctionsF

MVStrongPartitions

MVTestQ - MVTestFQ

MVWeaklyInjectiveFunctions - MVWeaklyInjectiveFunctionsF

MVWeaklySurjectiveFunctions - MVWeaklySurjectiveFunctionsF

MVWeakPartitions

Permanent

ViabilityMatrix - ViabilityMatrixF

Ricordiamo che per poter utilizzare il pacchetto all'interno di un notebook *Mathematica*® è necessario inserire nella directory
`%mathematica_home%\AddOns\ExtraPackages`
il file `mvset.m` e importarlo con il comando `<<mvset.m` .

Appendice B

ESEMPI DI UTILIZZO DI MVSET.M

Verranno proposti alcuni esempi di utilizzo del pacchetto *mvset.m*.
Nella prima sezione gli esempi coinvolgeranno i casi proposti nei primi due capitoli e ne affronteranno di nuovi, simili, ma più complessi. In una seconda parte si proverà a sfruttare il pacchetto in modo diverso…

BIBLIOGRAFIA

[1] M. Aigner. *Combinatorial Theoris*. Springer, 1979.

[2] J. Browne. *Programming for Engineers with* Mathematica®. Electronic Edition, Swinburne University of Technology, Melbourne, Australia, 2001.

[3] O. D'Antona. *Introduzione alla matematica discreta*. Apogeo, Milano, 1999.

[4] V. Marra e D. Mundici. Mv-algebras and abelian l-groups: a fruitful interaction. *Ordered algebraic structures*, pages 57–80, 2002.

[5] O. D'Antona e V. Marra. The combinatorics of mv sets. In pubblicazione.

[6] R.P. Stanley. *Generating functions*, volume 17 in 'Studies in Combinatorics'. Mathematical Association of America, 1978. A cura di G.-C. Rota.

[7] S. Wolfram. *The* Mathematica® *Book*. Cambridge University Press, 4th edition, 1999.

STRUMENTI INFORMATICI

1. *Mathematica*® 4, ver. 4.2, © copyright 1988-2002 Wolfram Research

2. MiKTEX , ver. 2.2, © copyright 1996-2003 MiKTeX.org

3. TEXnicCenter, ver. 1b6.01, © copyright 1999-2002 www.toolscenter.org