

Simboli matematici in T_EX e L^AT_EX

Enrico Gregorio

Sommario

Un'introduzione ai comandi primitivi di T_EX per la composizione di formule matematiche e ai corrispondenti comandi di L^AT_EX con esempi d'uso e suggerimenti per definire nuovi simboli in modo appropriato per sfruttare le spaziature automatiche di T_EX.

Abstract

An introduction to the primitive commands of T_EX for the typesetting of mathematical formulas and to the corresponding L^AT_EX commands, with examples and suggestions for defining new symbols in a suitable way in order to exploit the automatic spacing provided by T_EX.

1 Introduzione

Uno dei punti di forza di T_EX nei confronti di altri sistemi di tipografia elettronica è senza dubbio la capacità di comporre formule matematiche senza troppi interventi 'manuali'. Le tradizionali regole di composizione della matematica, che sono molto simili in tutti i paesi occidentali, sono già note al motore tipografico di T_EX e quindi chi lo adopera non è obbligato a conoscerle.

Il classico esempio è dato dalle due formule

$$\begin{aligned}a + b \\ a = b\end{aligned}$$

perché, a un'osservazione attenta, appare chiaro come la spaziatura fra i simboli sia diversa: leggermente più larga nella seconda rispetto alla prima. Infatti T_EX sa che il segno di uguaglianza è un simbolo di relazione, mentre quello di addizione è un simbolo di operazione e la tradizione tipografica, che ha origine nel significato dei simboli, vuole così. Eppure le due formule possono essere scritte semplicemente come `$a+b$` e `$a=b$` rispettivamente; delle spaziature si fa carico T_EX e normalmente non sbaglia.

Un altro classico esempio è quello della formula

$$\log(xy) = \log x + \log y$$

nella quale si può notare uno spazio sottile fra il simbolo della funzione logaritmo e la variabile, assente nel primo membro fra 'log' e la parentesi aperta. Se invece consideriamo

$$e^{x+y} = e^x e^y$$

possiamo notare che nell'operazione a esponente la spaziatura è assente: così dice la pratica tipografica, perché spaziare quella somma introdurrebbe un'ambiguità semantica che è meglio evitare. Possiamo provare a comporre la stessa formula spaziando l'esponente

$$e^{x+y} = e^x e^y$$

e l'effetto è certamente meno attraente.

Scopo di questo articolo non è di insegnare a scrivere correttamente le formule, si veda per questo l'ottimo 'Consigli su come non maltrattare le formule matematiche' (GUIGGIANI e MORI, 2008) se si desidera imparare bene quest'arte. L'articolo è invece orientato ad approfondire gli aspetti T_EXnici della faccenda: ciò dovrebbe essere utile a chi vuole ottenere effetti particolari o capire meglio il funzionamento del sistema senza cadere nei trabocchetti che qualche volta causano grattacapi e affrante richieste di aiuto.

Sebbene l'articolo riguardi L^AT_EX, sarà necessario e utile esaminare a fondo anche i comandi primitivi che regolano la composizione delle formule.

2 Codici matematici

I caratteri che si scrivono in ambiente matematico hanno per T_EX un significato diverso da quello che possiedono nel normale modo di composizione del testo. Ogni carattere ha associato un *codice matematico*, un numero, che viene convenientemente espresso in rappresentazione esadecimale; un codice matematico è un intero compreso fra "0 e "7FFF. Useremo la convenzione del T_EXbook nel quale i numeri preceduti dal simbolo " e scritti in carattere dattilografico sono in base 16. Le cifre sono, come si usa, 0 1 2 3 4 5 6 7 8 9 A B C D E F; la traduzione di "7FFF in rappresentazione decimale è 32 767, cioè $2^{15} - 1$: un codice matematico è un numero a 15 bit. C'è una piccola eccezione: un codice matematico può anche essere "8000 = 32 768, ci torneremo; per il momento ci limiteremo ai codici normali.

Ogni formato (il nucleo di L^AT_EX, per esempio) è responsabile di assegnare ai caratteri un codice matematico; solo alcune convenzioni sono impostate fin dall'inizio, ma modificabili se lo si desidera. Per esempio, il codice matematico di **a** è "7161 e si riconosce nelle ultime due cifre il codice ASCII del carattere **a** (esadecimale). Analogamente, il codice matematico di **1** è "7031. Inizialmente il codice matematico di ciascuna lettera (maiuscola

o minuscola) che abbia codice ASCII x è

$$"7100 + x$$

mentre quello delle cifre (da 0 a 9) è

$$"7000 + y$$

dove y è il codice ASCII della cifra. Tutti gli altri caratteri hanno, all'inizio, codice matematico nullo. Scopriremo fra poco che le prime due cifre del codice matematico sono proprio quelle responsabili di far comporre a T_EX le lettere in corsivo (quello speciale usato nelle formule) e le cifre in tondo. Le ultime due, naturalmente, si riferiscono al carattere da scegliere.

Si assegna un codice matematico a un carattere con comandi del tipo

```
\mathcode'a="7161
\mathcode'1="7031
```

perché un numero esadecimale può essere inserito, quando la sintassi di T_EX richiede un numero, precedendo le cifre con il carattere ".¹

Facciamo una breve digressione. T_EX mantiene un vettore con 256 componenti, il vettore dei codici matematici (ne ha parecchi altri). L'istruzione

```
\mathcode⟨numero₁⟩ = ⟨numero₂⟩
```

dice a T_EX di impostare la componente del vettore dei codici matematici di posto $\langle numero_1 \rangle$ al valore $\langle numero_2 \rangle$. Ovviamente $\langle numero_1 \rangle$ deve essere a 7 bit, mentre $\langle numero_2 \rangle$ deve essere a 15 bit. Gli spazi attorno al simbolo di uguaglianza, che indica l'assegnazione, sono opzionali (sarebbe opzionale anche il segno =, ma lo scriveremo sempre).

Una convenzione utile per scrivere $\langle numero_1 \rangle$ è di usare un carattere preceduto da ' che rappresenta il codice ASCII di quel carattere. In caso di dubbio se il carattere sia speciale e possa non essere compreso così com'è, è possibile precederlo con la barra rovescia: in questo contesto, '\a e 'a sono equivalenti.

Proviamo a vedere che succede scrivendo la formula

```
\[ \mathcode'a="7162 \mathcode'1="7131
a+1=x
\]
```

che produce

$$b + 1 = x$$

L'effetto è alquanto misterioso, ma non troppo: al posto di a appare b , perché il carattere b ha codice ASCII "62; il numero 1 appare come numero minuscolo, perché il font della famiglia Computer Modern usato per i caratteri matematici, che viene scelto quando la seconda cifra del codice matematico è "1, ha al posto delle cifre i numeri minuscoli

1. Si veda alla fine dell'articolo un'importante avvertenza.

o 1 2 3 4 5 6 7 8 9. L'effetto di quelle bizzarre assegnazioni non si propaga, per fortuna, dal momento che le formule costituiscono un gruppo e quindi l'assegnazione di quei codici viene annullata alla fine della formula. Infatti ora possiamo scrivere $\$a+1=x\$$ e ottenere $a + 1 = x$.

Ma torniamo alla teoria generale: un codice matematico è un numero esadecimale a quattro cifre. La prima cifra può essere da "0 a "7, come abbiamo visto, la seconda una cifra qualunque da "0 a "F; le ultime due rappresentano il codice ASCII del carattere da stampare, prendendolo dal font che viene scelto da T_EX esaminando le prime due cifre.

La prima cifra assegna al carattere un *tipo* secondo questa tabella:

0	ordinario	4	apertura
1	operatore	5	chiusura
2	operazione	6	punteggiatura
3	relazione	7	speciale

Questo è il trucco con il quale T_EX decide come trattare i simboli e le spaziature corrispondenti: per esempio, il codice matematico di < è "313C che lo qualifica come simbolo di relazione; quello di , è "613B che qualifica la virgola come segno di punteggiatura in modo che le formule come $\$(x,y)\$$ siano stampate correttamente: (x,y) .

Normalmente il tipo 1 non viene mai assegnato come codice matematico di un carattere: serve per simboli come quello di integrale o di sommatoria, vedremo più avanti perché esiste. Tutti i formati assegnano codici ai caratteri 'stampabili' diversi dalle lettere e dalle cifre² nel modo seguente:

```
\mathcode'. = "013A \mathcode'/ = "013D
\mathcode'| = "026A \mathcode'* = "2203
\mathcode'+ = "202B \mathcode'- = "2200
\mathcode': = "303A \mathcode'< = "313C
\mathcode' = "303D \mathcode'> = "313E
\mathcode'(' = "4028 \mathcode'[' = "405B
\mathcode') = "5029 \mathcode'] = "505D
\mathcode'! = "5021 \mathcode'? = "503F
\mathcode', = "613B \mathcode'; = "603B
\mathcode'' = "8000
```

Queste assegnazioni sono un po' sorprendenti, in certi casi. Il punto viene considerato un simbolo ordinario perché è usato come separatore decimale (nell'uso inglese). La barra è un simbolo ordinario invece che di operazione, perché non si usa spaziarla rispetto a dividendo e divisore; lo stesso vale per la barra dritta che non andrebbe usata per indicare il modulo, preferendo i comandi \lvert e \rvert forniti da amsmath. Asterisco, più e meno sono simboli di operazione; notate qualche stranezza sull'assegnazione del codice matematico al carattere - (trattino)?

2. Per la precisione, il codice matematico è rilevante solo per i caratteri con codice di categoria 11 o 12.

Un'altra piccola sorpresa è il tipo di punto esclamativo e interrogativo: sono simboli di chiusura, come le parentesi: il fatto è che questo fornisce la corretta spaziatura per il fattoriale ($n!$); il punto interrogativo non viene usato in matematica, gli viene assegnato il tipo 5 per analogia con $!$. Ultima osservazione: il carattere 'due punti' è un simbolo di relazione, perché il suo uso più frequente è questo. Vedremo come è possibile trasformarlo in un segno di punteggiatura o in simbolo di operazione.

Un discorso a parte merita il codice assegnato all'apostrofo (o singolo apice): è "8000 ed è questo che permette alla scrittura $\$f'(x)\$$ di essere interpretata come $f'(x)$.

Il tipo 7 è simile al tipo 0 per quanto riguarda la scelta delle spaziature, cioè produce un simbolo considerato ordinario; ma un carattere di tipo 0 sarà *sempre* preso dal font scelto guardando la seconda cifra del codice matematico, per uno di tipo 7 T_EX guarda invece il valore del registro numerico interno `\fam`, secondo una regola che scopriremo più in là.

3 Inserire simboli diversi

I caratteri del codice ASCII certamente non sono sufficienti a coprire il vastissimo spettro dei simboli matematici di uso frequente: solo i simboli di relazione sono alcune decine, poi ci sono quelli di operazione, gli operatori come l'integrale, le lettere greche e tanti altri. L'idea è allora di adoperare numeri a 15 bit per esprimerli; il comando apposito è `\mathchar`. Con

`\mathchar"tfxy`

si istruisce T_EX a stampare il carattere di posto "xy con tipo "t e preso dal font corrispondente alla famiglia "f. Se però il tipo è "7, T_EX opererà in modo analogo ai caratteri inseriti direttamente per quanto riguarda la scelta del font. Naturalmente sarebbe assurdo dover scrivere

`\[ \mathchar"1350_{i=1}^n a_i ]`

per la formula

$$\sum_{i=1}^n a_i$$

e infatti tutti sanno che il comando con cui richiamare il simbolo $\sum$ è `\sum`. Si potrebbe dunque pensare che sia presente nel formato in uso la definizione

`\def\sum{\mathchar"1350 }`

ma T_EX possiede un'istruzione più compatta:

`\mathchardef\sum="1350`

che è agli effetti pratici equivalente a quella con `\def`.³

3. C'è una differenza T_EXnica; il comando definito con `\def` è espandibile, quello definito con `\mathchardef` no.

Vediamo qui finalmente un simbolo di tipo 1, che obbedisce a regole molto particolari: infatti viene stampato verticalmente centrato rispetto all'asse delle formule, una linea immaginaria che è leggermente più alta rispetto alla linea di base e, normalmente, attraversa il simbolo di sottrazione e le linee di frazione, come in $-\frac{a}{b}$.

Ciascun formato (Plain T_EX o L^AT_EX, per esempio) ha una lunga lista di definizioni di simboli matematici come quella precedente. Per poterle scrivere è necessario consultare le tabelle dei font matematici; per fortuna in L^AT_EX sono state definite alcune tabelle standard che permettono la compatibilità fra famiglie di font diverse, ammesso che gli sviluppatori le seguano. Per la verità, L^AT_EX ha un suo sistema per dare quelle definizioni, che però alla fine si riduce a istruzioni elementari di quella forma; le vedremo in una prossima sezione.

4 Famiglie matematiche

Abbiamo dunque visto il significato della prima cifra di un codice matematico, sia per il vettore `\mathcode` che per `\mathchar` o `\mathchardef`. Per capire il significato della seconda, occorre introdurre il concetto di *famiglia*.

T_EX gestisce un massimo di 16 famiglie di font matematici, corrispondenti proprio alla seconda cifra di un codice matematico. Per convenzione, nella famiglia 0 c'è sempre il font con cui si scrive il testo normale; nella famiglia 1 il font per il corsivo matematico, cioè per le lettere da usare comunemente nelle formule; nella famiglia 2 c'è un font per i simboli più frequenti; la famiglia 3 contiene il font per i simboli 'grandi', come parentesi ingrandibili, sommatoria, integrale e altri.

Ci ricordiamo che il codice matematico dei caratteri 0-9 ha infatti "0 come seconda cifra. Questo significa che, normalmente, i caratteri da stampare verranno presi dal font usuale per il testo. *Normalmente*, perché se al momento di comporre questo carattere il parametro `\fam` ha un valore compreso tra 0 e 15, il carattere sarà preso dal font della famiglia `\fam`.

Facciamo un esempio che dovrebbe illustrare il concetto, sperando di non confondere le idee:

`\$ \fam=2 a=z \$`

produce la formula $\mathfrak{a} = \mathfrak{z}$. Infatti, siccome il codice matematico di **a** e **z** è rispettivamente "7161 e "717A, il loro tipo è 7, quindi T_EX sceglie il font appartenente alla famiglia 2 per comporli. Naturalmente nessuno fa così!

Le famiglie 0, 1, 2 e 3 sono sempre predefinite, perché forniscono il minimo indispensabile per comporre le formule. In Plain T_EX vengono definite anche le famiglie con nomi simbolici `\itfam`, `\slfam`, `\bfam` e `\ttfam`; scrivendo `\fam=\itfam` i caratteri con tipo 7 saranno composti nel font corsivo

usato nel testo, perché alla famiglia corrispondono le assegnazioni

```
\textfont\itfam=\tenit
\scriptfont\itfam=\sevenit
\scriptscriptfont\itfam=\fiveit
```

Il loro significato è: se occorre comporre un carattere nella famiglia `\itfam` (questo, lo ricordiamo, è semplicemente un certo numero), T_EX userà il font `\tenit` se gli occorre la grandezza normale, mentre userà `\sevenit` per gli esponenti e i pedici di primo livello, ricorrendo a `\fiveit` per quelli di livello superiore. Il significato di questi comandi di scelta di font dovrebbe essere chiaro.

I simboli matematici (caratteri normali o comandi definiti con `\mathchardef`) di tipo diverso da 7 non vengono influenzati dal valore di `\fam`. Il valore del parametro rispetta la struttura dei gruppi, quindi

```
#{\fam=\bffam a}=a$
```

produce la formula $\mathbf{a} = a$. In Plain però non si scrive così, perché il comando `\bf` è definito con

```
\def\bf{\fam\bffam\tenbf}
```

Fuori dal modo matematico il valore di `\fam` è irrilevante; in modo matematico un comando di scelta di font come `\tenbf` è ignorato.

All'inizio di ogni formula (dopo il `$` o `$$`), il valore di `\fam` è sempre impostato a `-1`, cosicché occorre cambiarne esplicitamente il valore se si desidera che i simboli di tipo 7 siano composti con il font corrispondente alla famiglia scelta. Dunque si può scrivere la formula precedente con `#{\bf a}=a$`. O anche con

```
#{\bf a}=\mit a$
```

perché `\mit` è il comando per scegliere la famiglia 1, cioè quella del normale corsivo matematico. Siccome il codice matematico di `=` è `"303D`, questo simbolo non è influenzato dal valore di `\fam`.

Con L^AT_EX la sintassi per ottenere la stessa cosa è molto diversa; un tempo si poteva in effetti scrivere come in Plain, ma i comandi introdotti con il passaggio alla versione L^AT_EX 2_ε sono più chiari e aiutano a minimizzare gli errori.⁴ È chiaro però che, alla fine, i comandi L^AT_EX eseguono sempre comandi primitivi e scrivere `#{\mathbf{a}}=a$` porta a eseguire qualcosa di molto simile a `#{\fam=<numero> a}=a$`. La situazione è più complicata, perché L^AT_EX deve tener conto del corpo corrente e quindi non ha un'assegnazione fissa dei font appartenenti a una famiglia.

Facciamo un riassunto di quanto visto. Un codice matematico può essere

4. Si possono, a dire il vero, usare comandi come `\bf`, ma è meglio evitarlo; versioni future di L^AT_EX potrebbero disabilitare questi arcaismi.

- assegnato dal vettore `\mathcode`;
- richiamato tramite un comando definito con `\mathchardef`;
- richiamato direttamente come argomento di `\mathchar`.

Un codice matematico è un numero che ha una rappresentazione esadecimale del tipo `"tfx`, dove `"t` è il tipo, `"f` la famiglia di riferimento e `"xy` indica il carattere da scegliere nel font che T_EX impiegherà usando la regola seguente:

1. se il tipo `"t` è minore di 7, il font sarà quello appartenente alla famiglia `"f`;
2. se il tipo `"t` è 7 e il valore attuale di `\fam` non è fra 0 e 15 (compresi), si procede come nel passo precedente;
3. se il tipo `"t` è 7 e il valore attuale di `\fam` è compreso fra 0 e 15, il font è quello appartenente alla famiglia `\fam`.

5 Atomi

Non è finita qui: il tipo determina anche le spaziature rispetto agli altri elementi della formula che T_EX sta componendo. Un codice matematico produce un *atomo* che può essere

0. un simbolo ordinario (tipo 0 o tipo 7);
1. un operatore (tipo 1);
2. un simbolo di operazione (tipo 2);
3. un simbolo di relazione (tipo 3);
4. un delimitatore aperto (tipo 4);
5. un delimitatore chiuso (tipo 5);
6. un segno di punteggiatura (tipo 6).

Per gli atomi, cioè i costituenti elementari delle formule, il T_EXbook usa i nomi **Ord**, **Op**, **Bin**, **Rel**, **Open**, **Close**, **Punct**. La lista non è completa, perché T_EX conosce altri sei tipi di atomo: **Inner**, **Over**, **Under**, **Acc**, **Rad** e **Vcent**. Tuttavia questi sei tipi di atomi vengono alla fine considerati come se fossero **Ord**, eccetto quelli **Inner**.

Esistono comandi (primitivi) che possono trasformare un atomo di qualsiasi tipo o anche una sottoformula in un altro; i nomi dovrebbero spiegarsi da soli:

```
\mathord \mathop \mathbin \mathrel
\mathopen \mathclose \mathpunct \mathinner
```

Leggermente diversi sono i comandi `\overline` e `\underline` che producono atomi di tipo **Over** e **Under** rispettivamente. In L^AT_EX esiste anche `\mathalpha` che corrisponde ai codici matematici di tipo 7, ma va usato solo in particolari casi che esamineremo più avanti.

6 Spaziature in modo matematico

T_EX decide la spaziatura fra due atomi a seconda della loro natura, usando la tabella seguente: in orizzontale è l'atomo di sinistra, cioè il primo, in verticale l'atomo di destra, cioè il secondo. Il significato di 0 è che lo spazio fra i due atomi è nullo, di 1 che lo spazio è sottile, di 2 che lo spazio è medio e di 3 che lo spazio è ampio. Quando il numero è fra parentesi, lo spazio non è aggiunto se gli atomi sono in apici o pedici. L'intestazione delle righe e delle colonne è con i numeri corrispondenti al tipo di atomo, 'I' indica gli atomi di tipo **Inner**.

	0	1	2	3	4	5	6	I
0	0	1	(2)	(3)	0	0	0	(1)
1	1	1	*	(3)	0	0	0	(1)
2	(2)	(2)	*	*	(2)	*	*	(2)
3	(3)	(3)	*	0	(3)	0	0	(3)
4	0	0	*	0	0	0	0	0
5	0	1	(2)	(3)	0	0	0	(1)
6	(1)	(1)	*	(1)	(1)	(1)	(1)	(1)
I	(1)	1	(2)	(3)	(1)	0	(1)	(1)

Torneremo fra poco sul significato dell'asterisco. I parametri di T_EX relativi a queste spaziature si chiamano `\thinmuskip`, `\medmuskip` e `\thickmuskip` che vanno impostati con sintassi del tipo

```
\thinmuskip=3mu
\medmuskip=4mu plus 2mu minus 4mu
\thickmuskip=5mu plus 5mu
```

cioè le dimensioni vanno espresse in *unità matematiche*, in modo analogo alle lunghezze elastiche (*rubber lengths* nel manuale di L^AT_EX): un'ampiezza naturale, un grado di espandibilità e un grado di contraibilità, questi due opzionali. Vale l'uguaglianza $18\mu = 1em$, dove $1em$ è preso dal font della famiglia 2 garantendo così spaziature adatte sia per le parti normali che per apici e pedici di primo e secondo livello. I valori normalmente usati sono proprio quelli indicati, e consigliamo di non modificarli. Come si vede, lo spazio sottile non è espandibile né comprimibile; quello usato attorno ai simboli di operazione è leggermente espandibile e può annullarsi all'occorrenza; quello attorno ai simboli di relazione è solo espandibile. La scelta ha a che fare con le usuali regole di precedenza nelle espressioni matematiche: i simboli di relazione 'legano di meno' rispetto ai simboli di operazione. La differenza fra le ampiezze naturali di questi spazi non è grande, ma sufficiente a distinguere visivamente le funzioni dei simboli nelle formule.

I comandi per usare queste spaziature in unità matematiche sono `\mskip`, analogo di `\hskip`, e `\mkern`, analogo di `\kern`.

Nella tabella il simbolo * significa che la combinazione è impossibile; infatti T_EX cambia d'autorità un atomo **Bin** in **Ord** in certe situazioni, perché

un atomo **Bin** deve essere preceduto e seguito da atomi compatibili con un simbolo di operazione binaria. Per esempio è impossibile che un atomo **Op** sia seguito da un simbolo di operazione: `\log-a` viene risolto come $\log -a$ cambiando il tipo del simbolo di opposto (normalmente un simbolo di operazione) in un atomo **Ord**; in termini di atomi la formula diventa **Op Ord Ord**. Lo stesso avviene nel caso di due simboli di operazione adiacenti: `a++b` diventa $a++b$, il secondo simbolo di operazione diventa un atomo **Ord**. Entrambi gli esempi sono di errato uso dei simboli. Un caso che capita è un atomo **Open** seguito da un atomo **Bin**: `1+(-1)` è una formula accettabile che diventa

Ord Bin Open Ord Ord Close

come si desidera, cioè $1 + (-1)$. Un caso in cui questo può creare problemi è quello del valore assoluto; il codice matematico di `|` è "026A, cioè si tratta di un simbolo ordinario; scrivere `|-1|` produce quindi il risultato scorretto $|-1|$. Soluzione: scrivere `\{-1\}` oppure `\lvert-1\rvert`, con i comandi di `amsmath`, che danno $|-1|$.

Nel caso si desiderasse usare il simbolo `++` come simbolo unico di operazione si può scrivere

```
$a\mathbin{++}b=\sqrt{a^2+b^2}$
```

ottenendo $a++b = \sqrt{a^2+b^2}$, ne parleremo diffusamente più avanti.

Si noti che fra due atomi di tipo **Rel** non viene inserito alcuno spazio: è questo che permette di scrivere $a := b$ senza spaziature sbagliate. Questa proprietà è usata in altre situazioni, per esempio per ottenere le relazioni negate: in Plain troviamo le definizioni

```
\mathchardef\not="3236
\def\neq{\not=}
```

che si basano sul fatto che il carattere di posto "36 nel font della famiglia 2 ha ampiezza nulla e ciò che viene stampato sta a destra del punto in cui T_EX si trova. Così il carattere 'sembra' sovrapposto al simbolo di uguaglianza.⁵

7 Modificare il tipo di un atomo

Supponiamo di voler usare la lettera *R* come simbolo di relazione, come capita spesso in formule come $a R b$. Potremmo consultare la tabella per vedere quale sia la spaziatura corretta e scrivere

```
$a\mskip\thickmuskip R
\mskip\thickmuskip b$
```

oppure `a\R;b`, visto che `\;` è equivalente a `\mskip\thickmuskip`. Ma c'è un modo molto più efficiente e sicuro:

5. Lo si può controllare con il comando `\the\fontcharwd\textfont2 "36` che produce infatti 0.0pt

`\mathrel{R}b`

che ha anche il pregio di essere più chiaro sintatticamente. Allo stesso modo qualsiasi oggetto può diventare un atomo **Op** tramite il comando `\mathop` il quale però ha una particolarità che richiede una certa attenzione: *se l'argomento di `\mathop` risulta essere un solo carattere, questo sarà centrato verticalmente rispetto all'asse delle formule*. Vediamo infatti il risultato di `\mathop{A}A` che produce $A A$. Eventuali comandi di scelta di font matematici sono eseguiti, ma non cambiano il comportamento: lo si vede con la formula `\mathop{\fam=0 A}A` che produce $A A$. Qui si nota anche come le graffe attorno all'argomento di `\mathop` costituiscono un gruppo, infatti l'assegnazione del valore a `\fam` non influenza il secondo carattere. La situazione con $\text{\LaTeX}$ è identica: la formula `\mathop{\mathrm{A}}A` produce $A A$. Il trucco per evitare questo comportamento è di inserire qualcosa di innocuo che renda la sottoformula non composta da un solo carattere: `\mathop{\kern0pt\fam=0 A}A` produce la formula 'corretta' $A A$.

Caricando il pacchetto `amsmath` (più precisamente il modulo `amsopn`), si ha a disposizione con $\text{\LaTeX}$ il comando `\operatorname` che solleva l'utente dal problema: con

`\[ \operatorname{A}A \]`

si ottiene la formula $A A$. Torneremo più avanti sugli atomi **Op**.

La specificazione di apici e pedici di un atomo non ne cambia la natura: per esempio le spaziature in `$(a+_1b)+_2c$` sono corrette, come si vede componendo la formula, $(a+_1b)+_2c$. Non è lo stesso se si usa un accento: una costruzione `\hat{+}` infatti produce un atomo **Acc** che, secondo le regole, è trattato come se fosse **Ord** per quanto riguarda le spaziature. Perciò per avere $a \hat{+} b$ come simbolo di operazione occorre

`\[ a \mathbin{\hat{+}} b \]`

che infatti dà le spaziature corrette

$$a \hat{+} b$$

che non si avrebbero con `\hat{+}` che dà $a \hat{+} b$. Ovviamente un utente saggio definirebbe un comando personale che in $\text{\LaTeX}$ sarebbe

`\newcommand*{\hplus}{\mathbin{\hat{+}}}`

e scriverebbe `\hplus`.

Ogni atomo può essere trasformato in **Ord** con il comando `\mathord`. Per esempio, volendo definire la 'sottrazione pitagorica'

$$a + - + b = \sqrt{a^2 - b^2}$$

non si può scrivere

`\[ a \mathbin{+ - +} b = \sqrt{a^2 - b^2} \]`

che darebbe

$$a + - + b = \sqrt{a^2 - b^2}$$

Infatti, secondo le regole sugli atomi **Bin** i due segni di addizione vengono trasformati in **Ord** e quello di sottrazione rimane **Bin**, con conseguenza disastrosa. Tuttavia non è necessario ricorrere alla mostruosità

`\mathbin{\mathord{+}\mathord{-}\mathord{+}}`

perché basterebbe

`\mathbin{+\mathord{-}+}`

sempre per le stesse regole sugli atomi **Bin**. In realtà c'è un modo ancora più semplice: una sottoformula, cioè una parte della formula scritta tra graffe, è sempre considerata come un **Ord** ai fini delle spaziature rispetto agli oggetti attorno. Perciò

`\mathbin{+{-}+}`

è sufficiente: la successione **Bin Ord Bin** viene trasformata in **Ord Ord Ord**.

Questo trucco permette di usare la virgola come separatore decimale in modo non troppo complicato: scrivere `$2,5$` produrrebbe 2,5, mentre basta `$2{,}5$` per avere 2,5. Purtroppo la virgola ha usi diversi che rendono impossibile rendere automatica la spaziatura. Almeno un pacchetto cerca di ovviare all'inconveniente, richiedendo però che la virgola usata come segno di interpunzione sia seguita da uno spazio. Più semplice, forse, è definire

`\def\comma{\mathord{,}}`

se si usa poco la virgola come separatore decimale; oppure

`\mathcode' ,="013B`
`\def\comma{\mathpunct{,}}`

se si usa molto la virgola come separatore decimale e poco come segno di punteggiatura (in modo matematico).

8 Atomi di tipo Op

Segue dalle regole già viste che un codice matematico del tipo `"1fxy` produce un carattere che viene centrato rispetto all'asse delle formule. Non solo: ciò che è specificato come apice e pedice va, se si è in una formula in `\displaystyle`, sopra e sotto il carattere. La stessa cosa accade con un atomo generato tramite `\mathop`. Per esempio,

`\[ \mathop{\mathrm{log}}_{10} 2`
`\approx 0.30103 \]`

verrebbe reso scorrettamente con

$$\log_2 \approx 0.30103$$

Esistono tre comandi primitivi che si occupano di questo: `\displaylimits`, `\limits` e `\nolimits`. Se non si specifica alcuno di questi dopo l'atomo **Op**, il comportamento è quello determinato dal primo: apici e pedici sopra e sotto solo in `\displaystyle`, altrimenti in posizione normale. Il secondo forza in ogni caso, anche non in `\displaystyle`, apici e pedici sopra e sotto; il terzo invece forza apici e pedici in posizione normale in qualsiasi caso. Dunque la definizione corretta dell'operatore 'logaritmo' sarebbe

```
\newcommand*{\log}{%
  \mathop{\mathrm{log}}\nolimits}
```

che è (circa) la definizione data dal nucleo di LATEX. La definizione senza `\nolimits` andrebbe bene per operatori come `min` e `max`. La definizione del simbolo di integrale è indiretta: in Plain è

```
\mathchardef\intop="1352
\def\int{\intop\nolimits}
```

e in LATEX è molto simile. Questo perché in generale i limiti di integrazione vanno posti a fianco del simbolo. Nel caso si volesse mettere i limiti sopra e sotto, basta specificare `\limits`: se ci sono più comandi del tipo in discussione subito dopo l'atomo **Op**, vale l'ultimo. Perciò, per esempio, volendo scrivere gli estremi di integrazione sopra e sotto (probabilmente perché si tratta di un integrale di linea), si potrà scrivere

```
\[
\int\limits_{\cammino}f(z)\,dz =
\sum_{i=1}^n\int_{\gamma_i}f(z)\,dz
\]
```

(dando a `\cammino` un'opportuna definizione) per ottenere

$$\int_{\gamma_1 \sqcup \gamma_2 \sqcup \dots \sqcup \gamma_n} f(z) dz = \sum_{i=1}^n \int_{\gamma_i} f(z) dz$$

Si noti che nell'integrale di sinistra si è usato `\limits`, in quello di destra no: è questione di gusti. Come appare evidente, l'indicazione `\limits` annulla l'effetto di `\nolimits` nella definizione di `\int`.⁶

Gli atomi di tipo **Op** possono anche essere 'abusati' per la loro proprietà di porre i limiti sopra e sotto. Ne è un esempio la definizione in Plain di `\buildrel`

6. Il lettore attento noterà che la funzione integranda è più vicina al segno di integrale; è stato infatti inserito nella formula un comando `\mspace{-24mu}` che la rende più gradevole. Si lascino sempre all'ultimo questi aggiustamenti.

```
\def\buildrel#1\over#2{\mathrel{%
  \mathop{\kern0pt#2}\limits^{#1}}}
```

La sintassi non è familiare per chi è abituato a LATEX, dove esiste il comando `\stackrel` che fa, più o meno la stessa cosa:

```
\def\stackrel#1#2{%
  \mathrel{\mathop{#2}\limits^{#1}}}
```

Per ottenere un simbolo di relazione nuovo, una x sormontata da un punto esclamativo, in Plain e LATEX si scriverebbe, rispettivamente,

```
\buildrel !\over x      \stackrel{!}{x}
```

e il risultato sarebbe (a sinistra quello in Plain, a destra quello in LATEX)

$$A \overset{!}{x} B \qquad A \stackrel{!}{x} B$$

Il lettore attento noterà una lieve differenza: a causa di `\kern0pt` nella definizione di Plain, la x non è centrata rispetto all'asse delle formule, come infatti deve essere. Dunque è meglio evitare l'uso di `\stackrel`, preferendo invece `\overset`: ecco le due formule di prima, ma a destra si è adoperato quest'ultimo comando, messo a disposizione da `amsmath`

$$A \overset{!}{x} B \qquad A \overset{!}{x} B$$

Come si vede, sono uguali (e corrette): il codice è, per la formula a sinistra quello già descritto con `\buildrel`, per l'altra si è usato `\mathrel{\overset{!}{x}}`.

Vediamo come funziona `\stackrel`: viene creato un atomo **Rel** a partire da un atomo **Op** costituito dal secondo argomento sopra il quale, usando `\limits` viene imposto il primo argomento. Nella definizione di `\buildrel` l'atomo **Op** non è mai costituito da un solo carattere, per via di `\kern0pt`. È importante ricordarsi che `\stackrel` crea un atomo **Rel**: se si desidera impiegarlo per costruire un simbolo ordinario occorre mettere l'intera costruzione tra graffe o come argomento di `\mathord`, altro motivo per non usarlo.

Il comando `\overset` di `amsmath` invece cerca di creare un atomo del tipo corretto basandosi sulla natura del secondo argomento: riconosce **Bin** e **Rel**, trattando tutto il resto come **Ord**; può quindi essere necessaria la menzione esplicita del tipo di atomo desiderato tramite uno dei comandi appositi come `\mathrel` e simili. La definizione di `\overset` è assai istruttiva e ingegnosa (dovuta a Mike Spivak):

```
\def\binrel@#1{\begingroup
  \setbox0=\hbox{\thinmuskip 0mu
    \medmuskip -1mu \thickmuskip 1mu
    \setbox2=\hbox{##1$}\kern-\wd2
    $}\#1$}%
  \edef\@tempa{\endgroup
```

```
\let\noexpand\binrel@@
\ifdim\wd0<0pt \mathbin
\else\ifdim\wd0>0pt \mathrel
\else \relax\fi\fi}%
\@tempa}
\let\binrel@@\relax
\newcommand{\overset}[2]{\binrel@{#2}%
\binrel@@{\mathop{\kern\z@#2}%
\limits^{#1}}}
```

Si tratta di una definizione del tutto simile a quella di `\buildrel`, solo che la menzione esplicita di `\mathrel` è sostituita da `\binrel@@` che riceve un valore dall'esecuzione di `\binrel@`; questo valore può essere `\mathbin` o `\mathrel` se il secondo argomento è rispettivamente un atomo `\Bin` o `\Rel`, è `\relax` altrimenti, presentando a T_EX, in questo caso, una sottoformula tra graffe che è perciò trattata come simbolo ordinario. Il lavoro di `\binrel@` consiste nell'impostare i valori dei parametri di spaziatura matematica in modo piuttosto curioso:

```
\thinmuskip = 0mu
\medmuskip = -1mu
\thickmuskip = 1mu
```

A questo punto si compone una *box* temporanea: prima, con

```
\setbox0=\hbox{\thinmuskip 0mu
\medmuskip -1mu \thickmuskip 1mu
\setbox2=\hbox{$\#1$}\kern-\wd2
$\#1$}
```

si misura l'ampiezza dell'argomento (che è il secondo di `\overset`) e si 'va a sinistra' di quell'ampiezza. Poi viene composta la formula `$_{\#1}$` che mette il secondo argomento fra due atomi `Ord` vuoti. Se l'argomento è un `Bin`, la spaziatura sarà negativa, quindi la formula risulterà più 'stretta' dell'ampiezza naturale e la *box* temporanea avrà ampiezza negativa. Se l'argomento è un `Rel`, la spaziatura sarà positiva, quindi l'ampiezza della *box* temporanea sarà positiva. Negli altri casi, non verrà aggiunto alcuno spazio e perciò l'ampiezza della *box* temporanea sarà nulla. Il resto della macro serve per dare il valore desiderato a `\binrel@@`, secondo i vari casi. Non è questo il luogo per andare nel dettaglio di come funzioni la definizione con `\edef` e quella particolare struttura di `\begingroup` e `\endgroup`.

Va in ogni caso osservato che un gruppo tra `\begingroup` e `\endgroup` in una formula *non* costituisce un atomo di tipo `Ord` e, se per caso il gruppo non contiene in definitiva alcun atomo, viene addirittura ignorato per quanto riguarda le spaziature matematiche. Il caso in esame è proprio questo: il gruppo viene aperto in modo da usare una nuova istanza locale dei registri `\box0` e `\box2` che sono usati spesso nelle macro per la composizione di formule matematiche.⁷ Si veda

7. La definizione è stata semplificata per eliminare dettagli T_EXnici irrilevanti.

la differenza tra `$a\begingroup+\endgroup b$` e `$a{+}b$`:

$$a + b \quad a + b$$

dove balza agli occhi che il gruppo non cambia la natura dell'atomo `Bin` nel primo caso, a differenza che nel secondo. In genere è consigliabile usare `\begingroup` e `\endgroup` per isolare assegnazioni locali in una formula, quando in altre situazioni il più semplice `{...}` è sufficiente.

Un altro aspetto degli atomi `Op` è stato sfruttato da Claudio Beccari (BECCARI, 2009, p. 144) nell'interessante definizione di una macro per scrivere automaticamente la 'd' del differenziale negli integrali o nelle forme differenziali. La tradizione matematica richiede che questa sia separata da ciò che precede con uno spazio sottile, ma con certe cautele:

$$\int \frac{1}{x} dx = \int \frac{dx}{x}$$

Nel primo integrale lo spazio sottile è necessario, nel secondo è vietato. Vorremmo anche spaziature automatiche in espressioni come

$$\omega = xy \, dx + dy$$

Il trucco è di inserire prima della 'd' un atomo `Op vuoto`! La tabella delle spaziature dice che questo atomo sarà preceduto dalla spaziatura giusta a seconda dell'atomo che precede (nessuna spaziatura se il differenziale compare all'inizio di una formula o sottoformula); poiché l'atomo seguente è `Ord` (la 'd'), sarà seguito da uno spazio sottile che annulliamo con il comando `\!`:

```
\def\diff{\mathop{\}\!d}
```

Chi volesse la 'd' in carattere tondo può modificare la definizione in modo ovvio. Le formule precedenti sono state scritte

```
\int\frac{1}{x}\diff x=
\int\frac{\diff x}{x}
\omega=xy\diff x + \diff y
```

(si ricordi che `\frac` è un comando L^AT_EX e non Plain). Scrivere `\diff{x}` è del tutto equivalente, ma non lo è `\diff{(x+y)}`, perché la parte tra graffe sarebbe trattata come una sottoformula e le spaziature attorno al + non partecipano agli aggiustamenti per la giustificazione. Si potrebbe definire `\diff` con un argomento, non sembra ragionevole, tanto più che spesso occorre $d^2 f$ e una sintassi come `\diff[2]{x}` è pesante e poco intuitiva rispetto a `\diff^2{x}`. Si veda BECCARI (1997) per una definizione dello stesso autore, molto più complicata.

Il trucco funziona perché T_EX inserisce ugualmente lo spazio richiesto dalle regole anche se l'atomo `Op` (vuoto) è seguito da un comando `\!` che significa `\mskip-\thinmuskip`: un comando di questo tipo non è un atomo.

9 Delimitatori

Gli atomi di tipo 4 e 5 sono quelli destinati a funzionare come le usuali parentesi tonde. Ogni atomo può essere trasformato in uno di questi con i comandi `\mathopen` e `\mathclose`. La definizione di `\lvert` di `amsmath` sarebbe essenzialmente equivalente a

```
\def\lvert{\mathopen|}
```

se non fosse che ci occorre una definizione diversa in modo che il comando funzioni anche dopo `\left`. Un caso in cui si possono usare esplicitamente `\mathopen` e `\mathclose` è la notazione (brutta) degli intervalli aperti nella forma $]a, b[$, che è stata scritta

```
\mathopen]a,b\mathclose[
```

Più interessante è scoprire come funziona la scelta dei simboli che compaiono dopo `\left` e `\right` e di come possano ingrandirsi a piacimento.

T_EX ha un altro vettore con 256 componenti simile a `\mathcode` che si chiama `\delcode`: il vettore dei *codici di delimitazione*. Ogni componente di questo vettore è un numero a 24 bit, quindi espresso con sei cifre esadecimali. La maggior parte di questi codici è -1 , quelli non negativi sono

```
\delcode' ( = "028300
\delcode' ) = "029301
\delcode' . = "0
\delcode' / = "02F30E
\delcode' < = "26830A
\delcode' > = "26930B
\delcode' [ = "05B302
\delcode' ] = "05D303
\delcode' | = "26A30C
```

Un codice di delimitazione negativo significa che il carattere non può essere usato dopo `\left` o `\right`; codice nullo significa che il carattere indica un delimitatore vuoto. Quando è non nullo il codice è della forma $"fxyguv$ e va letto come due numeri a 12 bit affiancati, molto simili ai codici matematici. La cifra $"f$ indica la famiglia da cui prendere il carattere di posto $"xy$, se la grandezza del delimitatore richiesto è 'normale'; se il delimitatore deve essere ingrandito, verrà preso il carattere dalla famiglia $"g$ di posto $"uv$. Per esempio,

```
$\mathchar"028 \mathchar"300$
```

produce $(($. Il font usato per la famiglia 3 ha le informazioni necessarie per costruire i delimitatori più grandi a richiesta, partendo, nel caso della parentesi aperta, da quello di posto $"00$.⁸

Il delimitatore vuoto è utile in situazioni nelle quali si vuole delimitare una formula solo da una

8. La formula precedente con le due parentesi non fa esattamente quello che viene mostrato: per motivi T_EXnici i caratteri nella famiglia 3 hanno posizioni strane rispetto alla linea di base.

parte: per scrivere un sistema di equazioni, per esempio. In questo caso si specificherà

```
\left\lbrace \langle equazioni \rangle \right.
```

Ma com'è possibile scrivere `\lbrace` dopo `\left`, visto che `\lbrace` non è un carattere? Se chiediamo a T_EX di mostrare la definizione di questo comando otterremo come risposta che è una macro:

```
macro:->\delimiter "4266308
```

La verità è che `\left` e `\right` devono essere seguiti o da un carattere con codice di delimitazione non negativo oppure da una macro la cui espansione cominci con `\delimiter`. Si potrebbe anche usare direttamente

```
\left\delimiter"4266308 \langle numero a 27 bit \rangle
```

ma nessuno lo fa mai, perché il numero da inserire è, come sempre in questi casi, arcano. Il comando `\delimiter` ha come argomento un numero a 27 bit che possiamo dunque esprimere nella forma $"tfxyguv$. La parte $"tfxy$ serve a T_EX nel caso non si sia dopo un comando `\left` o `\right` quando viene interpretata come un codice matematico; se invece siamo dopo uno dei comandi `\left` o `\right`, T_EX usa la parte $"fxyguv$ esattamente come se fosse stato specificato un codice di delimitazione.

Vediamo il caso di `\uparrow`, che è definito con

```
\def\uparrow{\delimiter"3222378 }
```

Se il comando non appare dopo `\left` o `\right`, l'effetto è equivalente a dire `\mathchar"3222`, quindi verrebbe creato un atomo **Rel**. Altrimenti T_EX costruisce un delimitatore (di apertura o di chiusura) secondo la regola vista per i codici di delimitazione: carattere $"22$ del font della famiglia 2 se è richiesta una misura normale, carattere $"78$ della famiglia 3 se deve essere ingrandito: lo vediamo con

```
$\mathchar"222 \mathchar"378$
```

che produce $\uparrow\uparrow$. La seconda freccia è apparentemente più piccola, per via delle regole interne di T_EX sulla costruzione di caratteri estendibili. Si tratta di regole piuttosto complesse che non è il caso di esaminare: basta sapere che funzionano, purché chi definisce il font sappia ciò che fa. Compito di T_EX è misurare la formula da delimitare (se è richiesto usando `\left` e `\right`) e scegliere la misura corretta della 'parentesi' in base anche ai valori dei parametri `\delimiterfactor` e `\delimitershortfall`. Un altro parametro che può interessare è `\nulldelimiterspace`, che è l'ampiezza di una *box* vuota inserita quando viene specificato un codice di delimitazione nullo.

Una parola sul comando `\middle` che è a disposizione quando si usino le estensioni ε -T_EX, in

particolare se si compila con L^AT_EX in una distribuzione aggiornata. Il comando ha le stesse proprietà e richieste di `\left` e `\right`: richiede un delimitatore allo stesso modo e lo ingrandisce con le stesse regole, dovendo però apparire tra quei due comandi. Il delimitatore costruito verrà considerato come un atomo **Ord**; è lasciato all'utente inserire le giuste spaziature. Potremmo per esempio definire

```
\def\bra#1{\left<#1\;\right|}
\def\ket#1{\left|\;\#1\right>}
\def\braket#1#2{%
  \left<#1\;\middle|\;\#2\right>}
```

per la notazione ‘bra-ket’ di Dirac usata in fisica teorica. Si noti come sia necessario specificare le spaziature attorno (prima o dopo nel caso di `\bra` e `\ket`) alla riga verticale. Un esempio:

```
\bra{\frac{x}{2}}\quad\ket{\frac{y}{3}}
\quad\braket{\frac{x}{2}}{\frac{y}{3}}
```

darebbe

$$\left\langle \frac{x}{2} \middle| \right| \frac{y}{3} \right\rangle \quad \left\langle \frac{x}{2} \middle| \frac{y}{3} \right\rangle$$

Il pacchetto `braket` si occupa di questo (risolvendo le spaziature in modo diverso, però).

Ci si può aspettare che i comandi delle serie `\big...` e `\Big...` siano definiti in termini di `\left` e `\right`: la congettura è corretta. Vediamone uno, `\bigl`, che è definito in Plain tramite

```
\def\bigl{\mathopen\big}
\def\big#1{\{\hbox{$\left#1$}
  \vbox to8.5pt{\}\right.$}}}
```

(la definizione è semplificata). Quando scriviamo `\bigl(`, T_EX sostituisce `\mathopen\big`, quindi produce un atomo **Open** che consiste di una parentesi aperta adatta a delimitare una formula che sia alta 8.5 pt. Il valore è adatto a caratteri di corpo 10, e occorre modificarlo se il corpo usato è diverso. In L^AT_EX la definizione è la stessa e infatti questi comandi non funzionano nel modo che ci si aspetterebbe. Il pacchetto `amsmath` infatti li modifica in modo che le misure siano adatte al corpo corrente. Per esempio,

```
$\bigl(b+(c+d)\bigr)$,
{\Large$\bigl(b+(c+d)\bigr)$}
```

produce

$$a(b + (c + d)), a\big(b + (c + d)\big)$$

ma, senza `amsmath`, si otterrebbe

$$a(b + (c + d)), a\big(b + (c + d)\big)$$

che è, evidentemente, sbagliato.

10 Atomi di tipo Inner

Un atomo di tipo **Inner** è una sottoformula delimitata con `\left` e `\right`. Si guardi con attenzione la formula inserita con

```
\[ x\Bigl(\frac{a}{b}\Bigr) +
x\left(\frac{a}{b}\right) ]
```

che produce

$$x\left(\frac{a}{b}\right) + x\left(\frac{a}{b}\right)$$

In effetti abbiamo barato, aumentando la dimensione dello spazio sottile che, secondo la tabella, viene inserito tra un atomo **Ord** e un atomo **Inner**, cioè il valore del parametro `\thinmuskip`. Se riproduciamo la formula senza questo aumento

$$x\left(\frac{a}{b}\right) + x\left(\frac{a}{b}\right)$$

la differenza si può notare ugualmente. Per un'espressione fra parentesi così grande la cosa può anche andare bene, ma è certamente scorretta nella formula `$a\left(b+c\right)$`; a sinistra della riga metteremo la formula scorretta, a destra quella scritta correttamente `$a(b+c)$`:

$$a(b+c) \quad a(b+c).$$

Alcuni front-end per T_EX e L^AT_EX inseriscono automaticamente `\left` e `\right` prima delle parentesi: meglio disabilitare questa funzione. Oppure è possibile, per quanto visto prima, mettere l'espressione delimitata con `\left` e `\right` tra graffe che la trasformeranno da **Inner** a **Ord**, ciò che non è sempre desiderabile.

11 Stili matematici

Il titolo di questa sezione non tragga in inganno: non si discuterà di come scrivere la matematica, ma di un argomento T_EXnico. Infatti T_EX conosce ben otto stili matematici, che usa nelle varie situazioni: due per le formule in mostra, due per le formule in testo, due per apici e pedici di primo livello e gli ultimi due per apici e pedici di secondo livello. Il T_EXbook li indica con *D*, *D'*, *T*, *T'*, *S*, *S'*, *SS* e *SS'*. La differenza tra gli stili *X* e *X'* riguarda essenzialmente il posizionamento degli esponenti e non la tratteremo.

A ogni coppia di stili corrisponde un comando di scelta di quello stile: `\displaystyle`, `\textstyle`, `\scriptstyle` e `\scriptscriptstyle`. Si noti che questi comandi sono *dichiarazioni*, il loro effetto è valido in tutto il gruppo in cui si trovano: l'intera formula o una sottoformula tra graffe o la coppia `\left-\right`. Questi comandi *non* obbediscono ai gruppi tra `\begingroup` e `\endgroup`.

La differenza fondamentale tra stile *D* e *T* riguarda la scelta della grandezza di certi simboli, per esempio `\sum`. Questo simbolo viene preso dal font della famiglia 3 che possiede informazioni in

modo da sceglierne uno più grande se lo stile di formula in vigore è D invece che T :

```
\[{\textstyle\sum}\sum\]
```

sceglie due simboli diversi, cioè

$$\Sigma \sum$$

Infatti nella sottoformula lo stile in vigore è T , mentre fuori dalle graffe lo stile in vigore è D , dal momento che siamo in una formula in mostra. L'utente non deve fare nulla, di solito: la decisione sulla grandezza del simbolo usato spetta a TEX. Il motivo della differenza è che nelle formule in corpo un simbolo grande turberebbe la spaziatura verticale delle righe, problema che non si pone con le formule fuori corpo.

Normalmente l'assegnazione dello stile è automatica; sarà D quando TEX deve cominciare a comporre una formula fuori corpo, diventerà T durante l'elaborazione del numeratore di una frazione che appaia in stile D o D' (e sarà T' per il denominatore); sarà invece S per il numeratore (e S' per il denominatore) di una frazione che appaia in stile T . Se un radicale compare in stile D , il radicando viene composto in stile D' . Un esponente di un atomo in stile D , D' , T o T' viene composto in stile S , un pedice invece in stile S' . Ma, come abbiamo visto, è possibile forzare la scelta di uno stile con le dichiarazioni apposite. Per esempio, la definizione della macro `\dfrac` di `amsmath` è più o meno equivalente a

```
\def\dfrac#1#2{%
  {\displaystyle\frac{#1}{#2}}}
```

(si noti la coppia di graffe in più per confinare l'effetto della dichiarazione).

È spesso impossibile conoscere in anticipo in che stile verrà composta una formula o un certo simbolo, oppure è necessario definire questo simbolo in modo diverso nei vari stili. Per questo TEX ha la primitiva `\mathchoice` che prende quattro argomenti: le liste di token da eseguire nei quattro stili principali. Se scriviamo

```
\[ \def\x{\mathchoice{0}{1}{2}{3}}
\x\frac{x^{\x^{\x}}}{a} \]
```

otteniamo la strana formula

$$0 \frac{1^3}{a}$$

che mostra appunto le scelte fatte da TEX nei vari stili.

Supponiamo che non si gradiscano troppo i simboli del tipo `\bigoplus` che vengono usati in stile D , perché troppo grandi. Un modo possibile di ovviare al problema è di usare `\mathchoice`:

```
\newcommand*\boplus{\mathop{%
  \mathchoice{\textstyle\bigoplus}
    {\bigoplus}{\bigoplus}{\bigoplus}}}
```

In questo modo LATEX userà anche in stile D lo stesso simbolo che usa in stile T . Si noti che l'effetto non si propaga, perché l'argomento di `\mathop` tra graffe costituisce un gruppo. Ovviamente, cambiare uno dei simboli richiede di cambiare anche tutti quelli simili; vogliamo una macro che faccia il lavoro con il minimo sforzo. Eccola qui, adoperando la macro `\csletcs` del pacchetto `etoolbox`:

```
\newcommand*\reducebigsymbol[1]{%
  \csletcs{@@#1}{#1}%
  \@namedef{#1}{\mathop{\mathchoice
    {\textstyle\@nameuse{@@#1}}
    {\@nameuse{@@#1}}
    {\@nameuse{@@#1}}
    {\@nameuse{@@#1}}}}}
```

Naturalmente va data in un ambiente in cui `@` sia una lettera. Dopo questo, avremo che

```
\reducebigsymbol{\bigoplus}
\reducebigsymbol{\bigotimes}
\reducebigsymbol{\bigvee}
\reducebigsymbol{\bigwedge}
```

ridefinirà `\bigoplus` e gli altri indicati. Nel caso si dovesse tornare alle definizioni originali, basterà commentare le righe con `\reducebigsymbol`. Si tratta di una versione astratta della definizione precedente, con la differenza che questa salva il significato originale di `\bigoplus` (per esempio) nel comando `\@@bigoplus` e ridefinisce il comando stesso. Dunque il documento potrà essere scritto come sempre.

Più maneggevole è la macro `\mathpalette` che ha due argomenti; il primo dei due deve a sua volta essere una macro che prende due argomenti, il primo dei quali sia una delle dichiarazioni di stile. Per esempio,

```
\mathpalette\ab
```

si espande a

```
\mathchoice{\a\displaystyle\b}
{\a\textstyle\b}
{\a\scriptstyle\b}
{\a\scriptscriptstyle\b}
```

ed è compito della macro `\a` sapere che fare dei due argomenti. Tipicamente `\a` sarà definita con

```
\def\ab#1#2{{#1...#2...}}
```

cioè comporrà una sottoformula nello stile specificato (il primo argomento sarà sempre una delle dichiarazioni di stile) facendo qualcosa con il secondo argomento. Si possono trovare parecchi esempi in rete.

12 LATEX

Uno dei problemi fondamentali di LATEX è di fornire definizioni sufficientemente generiche che siano

adatte a diverse famiglie di caratteri e codifiche. Quando viene compilato il formato con $\text{\LaTeX}$ vengono letti quattro file: `latex.ltx` che contiene le definizioni del nucleo di $\text{\LaTeX}$, `fonttext.ltx`, `fontmath.ltx` e `preload.ltx`. Ci interessa in particolare il secondo che si occupa delle definizioni normali dei simboli matematici e dei font da usare. Pacchetti caricati dall'utente nel proprio documento potranno modificare queste definizioni (per esempio `fourier` o `mathpazo`).

Non studieremo tutto il file `fontmath.ltx`, ma solo le parti fondamentali. Come Plain definisce le famiglie matematiche 'obbligatorie' (da 0 a 3), anche $\text{\LaTeX}$ lo deve fare, ma in modo indiretto in modo che le attribuzioni di font alle famiglie dipendano dal corpo in uso. Perciò, invece di assegnare

```
\textfont0=\tenrm
\scriptfont0=\sevenrm
\scriptscriptfont0=\fiverm
```

come fa Plain, in $\text{\LaTeX}$ si scrive

```
\DeclareSymbolFont{operators}
  {OT1}{cmr}{m}{n}
```

Infatti la famiglia 0 riceve il nome simbolico `operators`; analogamente

```
\DeclareSymbolFont{letters}
  {OML}{cmm}{m}{it}
\DeclareSymbolFont{symbols}
  {OMS}{cmsy}{m}{n}
\DeclareSymbolFont{largesymbols}
  {OMX}{cmex}{m}{n}
```

definiscono i font da usare per le famiglie 1, 2 e 3. Ricordiamo che in $\text{\LaTeX}$ ogni font è caratterizzato da quattro attributi: codifica, famiglia, peso e forma. Normalmente a ogni combinazione di codifica e famiglia corrisponde un file di descrizione (per esempio `ot1cmr.fd` o `omlcmm.fd`) dove vengono stabilite le corrispondenze delle combinazioni dei vari attributi con i 'veri' font disponibili. Queste famiglie vanno dichiarate in quest'ordine preciso, perché $\text{\TeX}$ ha bisogno che nelle famiglie da 0 a 3 ci siano font con certe caratteristiche.

Le codifiche OT1, OML, OMS e OMX sono esattamente quelle descritte nel $\text{\TeX}$ book per i font `cmr10`, `cmmi10`, `cmsy10` e `cmex10` e tutti i font matematici disponibili usano essenzialmente queste tabelle.

I comandi come `\mathrm` vengono definiti con

```
\DeclareSymbolFontAlphabet
  {\mathrm}{operators}
```

e questo fa in modo che una chiamata come `\mathrm{A}` sia sostituita da una sostanzialmente equivalente a

```
{\fam=\symoperators A}
```

perché il nome simbolico `operators` corrisponde alla costante `\symoperators`; analogamente per le altre famiglie:

Nome	Famiglia
<code>operators</code>	<code>\symoperators = 0</code>
<code>letters</code>	<code>\symletters = 1</code>
<code>symbols</code>	<code>\symsymbols = 2</code>
<code>largesymbols</code>	<code>\symlargesymbols = 3</code>
<code>AMSA</code>	<code>\symAMSA = 4</code>
<code>AMSb</code>	<code>\symAMSb = 5</code>

Le ultime due righe denotano famiglie definite dal pacchetto `amsfonts` (caricato automaticamente da `amssymb`); i valori mostrati non sono fissi, eccetto i primi quattro, perché altri pacchetti caricati prima potrebbero modificarli e l'uso di nomi simbolici evita di doverli conoscere.

Il file `fontmath.ltx` prosegue definendo i font matematici per la versione `bold`:

```
\SetSymbolFont{operators}{bold}
  {OT1}{cmr}{bx}{n}
\SetSymbolFont{letters}{bold}
  {OML}{cmm}{b}{it}
\SetSymbolFont{symbols}{bold}
  {OMS}{cmsy}{b}{n}
```

La sintassi è simile, ma il comando ha un argomento in più e può essere usato solo se sono già state definite la *versione* e la famiglia. Le versioni predefinite sono `normal` e `bold`, questa viene scelta con la dichiarazione

```
\mathversion{bold}
```

Di fatto l'introduzione dei nuovi comandi di $\text{\LaTeX}$ 2_ε evita quasi sempre il ricorso alle contorsioni necessarie per ottenere formule in nero: si carichi il pacchetto `bm`, se servono simboli in nero. Una nuova versione è definita con il comando

```
\DeclareMathVersion{nome}
```

che non ha altro effetto che di abilitare l'uso del comando `\SetSymbolFont` per questa versione. Se una particolare famiglia non ha associato un font per la versione scelta, viene adoperato quello associato alla versione `normal`, cioè quello dichiarato con `\DeclareSymbolFont`.

Successivamente `fontmath.ltx` definisce

```
\DeclareMathAlphabet{\mathbf}
  {OT1}{cmr}{bx}{n}
\DeclareMathAlphabet{\mathsf}
  {OT1}{cmsy}{m}{n}
```

e simili comandi per `\mathit` e `\mathtt`. Occorre moderazione nell'uso di questi 'alfabeti matematici', perché la prima apparizione di `\mathbf`, o di uno degli altri comandi dello stesso tipo, ordina l'allocazione di una nuova famiglia. Il numero di famiglie disponibili è, come sappiamo, sedici di cui quattro già impegnate per le esigenze minime. Per questo $\text{\LaTeX}$ non dice

```
\DeclareMathAlphabet{\mathrm}
{OT1}{cmr}{m}{n}
```

altrimenti anche l'uso di `\mathrm` allocherebbe una nuova famiglia: inutile, visto che la famiglia apposita è già disponibile, e dannoso perché getterebbe via una risorsa preziosa. Perciò il comando diverso

```
\DeclareSymbolFontAlphabet
{\mathrm}{operators}
```

che abbiamo già visto.

Il comando `\DeclareMathAlphabet` definisce i font per la versione `normal`; se si volesse definire il font corrispondente a `\mathsf` per la versione `bold` si potrebbe usare

```
\SetMathAlphabet{\mathsf}{bold}
{OT1}{cms}{bx}{n}
```

Il comando è del tutto analogo a `\SetSymbolFont`.

I comandi successivi in `fontmath.ltx` si occupano della corrispondenza del corpo con gli stili matematici. Per esempio, la riga

```
\DeclareMathSizes{10}{10}{7}{5}
```

dice che quando il corpo corrente è 10 pt, i simboli in stile *D* e *T* saranno di corpo 10, quelli in stile *S* di corpo 7 e quelli in stile *SS* in corpo 5.

L^AT_EX ricalcola le associazioni dei font alle famiglie ogni volta che deve comporre una formula, basandosi sul corpo corrente. È una procedura che fa perdere un po' di tempo, ma non è possibile fare altrimenti, se si vuole che le formule si adattino al corpo in uso. Il comando interno per questa associazione è `\check@mathfonts`, che usa una delle dichiarazioni precedenti per dare un significato ai comandi `\tf@size`, `\sf@size` e `\ssf@size`. Se non ne trova di adatte, perché si usa un corpo non standard, esegue i calcoli necessari secondo uno schema ragionevole. Per esempio, a corpo 80 eseguirebbe

```
\def\tf@size{80}
\def\sf@size{56}
\def\ssf@size{40}
```

cioè il corpo in stile *S* è 8/10 del corpo principale, quello in stile *SS* è la metà. Soprattutto con corpi piccoli è consigliabile usare quelli standard, in modo che il calcolo non porti a font illeggibili. Per esempio L^AT_EX definisce

```
\DeclareMathSizes{5}{5}{5}{5}
\DeclareMathSizes{6}{6}{5}{5}
```

proprio per evitare che venga usato il corpo 4 per gli esponenti e pedici di primo livello e addirittura 2.5 per quelli di secondo livello. Ma, naturalmente, scrivere a corpo 5 è da evitare comunque.

La sintassi usata da `fontmath.ltx` per dare valori al vettore `\mathcode` è assai particolare:

```
\DeclareMathSymbol{a}{\mathalpha}
{letters}{'a}
```

...

```
\DeclareMathSymbol{Z}{\mathalpha}
{letters}{'Z}
```

```
\DeclareMathSymbol{0}{\mathalpha}
{operators}{'0}
```

...

```
\DeclareMathSymbol{9}{\mathalpha}
{operators}{'9}
```

```
\DeclareMathSymbol{!}{\mathclose}
{operators}{'21}
```

```
\DeclareMathSymbol{*}{\mathbin}
{symbols}{'03}
```

...

```
\DeclareMathSymbol{.}{\mathord}
{letters}{'3A}
```

```
\DeclareMathSymbol{:}{\mathrel}
{operators}{'3A}
```

...

Si tratta essenzialmente di un'operazione equivalente (per la prima riga) a

```
\mathcode'a="7\symletters'a
```

dove `\symletters` e `'a` vanno pensati nella loro rappresentazione esadecimale. Di ogni carattere sono specificati nell'ordine il tipo, la famiglia da usare e il codice ASCII da impiegare. Si tratta solo di un modo più esplicito di impostare i codici rispetto a quello compatto ma oscuro direttamente con `\mathcode`.

Per il vettore `\delcode` e per i comandi definiti in Plain con `\delimiter` la sintassi è simile:

```
\DeclareMathDelimiter{(){\mathopen}
{operators}{'28}{largesymbols}{'00}
\DeclareMathDelimiter{\uparrow}{\mathrel}
{symbols}{'22}{largesymbols}{'78}
```

Nel caso il primo argomento sia un carattere, questo equivale a impostare il valore di `\delcode` e il secondo argomento è ignorato; se il primo argomento è un comando, questo è definito mediante `\delimiter`. Perciò il primo esempio equivale a⁹

```
\delcode' (=
"\symoperators28\symlargesymbols00
```

mentre il secondo a

```
\def\uparrow{\delimiter
'3\symsymbols22\symlargesymbols78}
```

Per verificarlo, scriviamo

```
\the\delcode' (
\meaning\uparrow
```

9. Qui si suppone sempre che `\sym...` sia tradotto nella sua rappresentazione esadecimale, non è possibile scrivere così quelle definizioni e infatti la definizione in L^AT_EX non è questa, ma quella riportata prima.

per avere il significato (questo articolo è composto per L^AT_EX e quindi il risultato dipende dalle definizioni riportate):

```
164608
macro:->\delimiter "322378
```

che è corretto, essendo $164608 = 28300$.

Potremmo usare `\mathchoice` e, implicitamente, `\check@mathfonts` nel caso avessimo bisogno di un simbolo particolare di un font, diciamo $\odot$, come simbolo di un'operazione (associativa, vista la regola che segue):

$$a \odot (b \odot c) = (a \odot b) \odot c.$$

Possiamo definirlo senza sprecare una famiglia. Il simbolo si trova al posto "A9 del font Marvosym, accessibile tramite la famiglia mvs. Il pacchetto `pifont` ci permette di usare simboli arbitrari da ogni font di cui conosciamo la tabella:

```
\usepackage{pifont}
\newcommand*\Smile{
  \mathbin{
    \mbox{\Pisymbol{mvs}{A9}}
  }}

```

tratta il comando esattamente come desideriamo. Tuttavia questo simbolo non cambia grandezza se messo a esponente; possiamo allora sfruttare `\mathchoice`:

```
\usepackage{pifont}
\newcommand*\Smilesym[1]{%
  \mbox{\fontsize{#1}{#1}%
    \Pisymbol{mvs}{A9}}}
\makeatletter
\newcommand*\Smile{%
  \mathbin{\mathchoice
    {\Smilesym\tf@size}
    {\Smilesym\tf@size}
    {\Smilesym\tf@size}
    {\Smilesym\tf@size}}}
\makeatother

```

e abbiamo la possibilità di usare il simbolo anche a esponente:

$$x^{a \odot b} = x^a \odot x^b$$

(è un'operazione molto peculiare, $\odot$).¹⁰

Una definizione più generale potrebbe essere

```
\usepackage{pifont}
\makeatletter
\newcommand\Pimathsymbol[3][\mathord]{%
  #1{\@Pimathsymbol{#2}{#3}}}
\def\@Pimathsymbol#1#2{\mathchoice
  {\@Pimathsymbol{#1}{#2}\tf@size}
  {\@Pimathsymbol{#1}{#2}\tf@size}
  {\@Pimathsymbol{#1}{#2}\tf@size}
  {\@Pimathsymbol{#1}{#2}\sf@size}}
\makeatother

```

10. Dopo l'impostazione di `\fontsize` non è necessario dare `\selectfont` perché questo è implicito in `\Pifont`.

```
{\@Pimathsymbol{#1}{#2}\ssf@size}}
\def\@Pimathsymbol#1#2#3{%
  \mbox{\fontsize{#3}{#3}%
    \Pisymbol{#1}{#2}}}
\makeatother

```

data la quale potremmo dire

```
$a\Pimathsymbol[\mathbin]{mvs}{A9}b$
```

per avere la formula $a \odot b$. Ovviamente il tutto andrà in una definizione del tipo

```
\newcommand*\Smile{%
  \Pimathsymbol[\mathbin]{mvs}{A9}}

```

Unica limitazione è che il font sia disponibile nella codifica U (cioè 'generica'): se il nome della famiglia è `xyz`, occorre che nel sistema sia presente il file `uxyz.fd`, che di solito succede per i font di simboli. Il comando `\Pimathsymbol` ha come argomento opzionale il tipo di atomo che desideriamo (valore normale `\mathord`), come primo argomento obbligatorio la famiglia e come secondo argomento il byte da adoperare, esattamente come per `\Pisymbol`. Altro esempio: un simbolo per indicare una contraddizione potrebbe essere

```
$\Pimathsymbol{mvs}{45}$
```

che darebbe $\neg$. È possibile produrre la tabella di un font di cui si conosca il nome della famiglia compilando con L^AT_EX il seguente file `testfam.tex`:

```
\let\noinit!
\input{nfssfont}
\fontencoding{U}
\fontfamily{mvs}
\selectfont
\table\bye

```

Chi è pratico della linea di comando può sostituire in quel file la chiamata esplicita della famiglia con la riga

```
\fontfamily{\test}
```

Lanciando la compilazione con

```
> latex "\def\test{pzd}\input{testfam}"
```

si ricaverebbe la tabella del font Zapf Dingbats. Si ricordi che i comandi in cui si usa il doppio apice " vanno dati prima di caricare `babel`.

Per chi non vuole fare le cose a mano, c'è il pacchetto `fonttable` di Peter Wilson che permette di realizzare tabelle di font e di impostare il font anche con gli attributi di L^AT_EX: il comando `\xfonttable`, a differenza del comando principale `\fonttable` del pacchetto, ha quattro argomenti; nell'ordine (1) la codifica, (2) il nome della famiglia, (3) il peso, (4) la forma.¹¹ La tabella del Computer

11. Il comando `\xfonttable` è stato aggiunto su suggerimento dell'autore con il codice concepito scrivendo questo articolo.

Modern nero corsivo in codifica T1 si otterrebbe con

```
\xfonttable{T1}{cmr}{bx}{it}
```

mentre quella del font Marvosym che ci interessava per il simbolo © si avrebbe con

```
\xfonttable{U}{mvs}{m}{n}
```

Il vantaggio di usare questo comando è che la tabella riporta anche i valori decimali con cui richiamare il carattere, come si vede nella tabella 1 e questo permette di definire i comandi precedenti con il riferimento al valore decimale, eliminando il problema di `babel`. La tabella non mostra tutti i caratteri del font, per questioni di spazio.

13 Altre tecniche

Certi autori preferiscono usare un simbolo simile a $\times$ per indicare prodotti di famiglie di insiemi, ma occorre che sia nello stile degli operatori grandi:

$$\bigtimes_{i=1}^n A_i$$

Il pacchetto `dlftxbcodetips` di Lars Madsen fornisce una definizione adatta:¹²

```
\newcommand\dlf@b@gtimes[1]{%
  \vcenter{\hbox{%
    #1$\m@th\mkern-2mu
    \times\mkern-2mu$}}
\newcommand\dlf@b@bigtimes{%
  \mathchoice{\dlf@b@gtimes\huge}
    {\dlf@b@gtimes\LARGE}
    {\dlf@b@gtimes}
    {\dlf@b@gtimes\footnotesize}
}
\newcommand\bigtimes{%
  \mathop{\dlf@b@bigtimes}\displaylimits}
```

Come in altre situazioni si impiega `\mathchoice` in modo che il simbolo sia della grandezza giusta nei quattro stili matematici. La particolarità di questa definizione è nell'uso di `\vcenter` che crea un atomo **Vcent** (che è trattato poi come se fosse **Ord**). La sintassi di `\vcenter` è la stessa di `\vbox`, ma questo comando può essere usato solo in modo matematico e la `vbox` costruita viene centrata rispetto all'asse delle formule, come richiesto per gli operatori grandi. Si usa, come si vede, il simbolo `\times` a un font più grande (o più piccolo). Un limite della definizione adottata da Madsen è che non funziona se la formula appare in un contesto nel quale sia valida una dichiarazione come `\small` o `\footnotesize`. La si potrebbe modificare nel modo seguente:

12. La definizione è dovuta all'autore, risultato di una discussione su `comp.text.tex`.

```
\makeatletter
\@ifundefined{@currsizeindex}%
  {\RequirePackage{relsize}%
   \let\@bl@rger\relsize}
  {\def\@bl@rger#1{\larger[#1]}}
\def\bigtimes{
  \mathop{\@bigtimes}\displaylimits}
\def\@bigtimes{
  \mathchoice{\@bigt@mes4}{\@bigt@mes3}
    {\@bigt@mes0}{\@bigt@mes{-1}}}
}
\def\@bigt@mes#1{
  \vcenter{\hbox{\@bl@rger{#1}%
    $\m@th\mkern-2mu\times\mkern-2mu$}}}
\makeatother
```

Si tratta di una definizione molto simile, solo che le variazioni di grandezza sono relative al corpo attuale, sfruttando il comando `\relsize` del pacchetto `relsize`. Solo nel caso in cui sia definito il comando `\@currsizeindex` la definizione è diversa, perché la classe del documento è una di quelle dell'AMS nelle quali è definito un comando `\larger` che ammette come argomento opzionale un 'salto'. In queste classi `\larger[4]` è equivalente a `\relsize{4}`.

Come esercizio si analizzino le definizioni di `\nuparrow` e `\ndownarrow`

```
\RequirePackage{amssymb,graphicx}
\def\@nrotarrow#1#2{%
  \setbox0=\hbox{$\m@th#1\uparrow$}%
  \dimen0=\dp0
  \setbox0=\hbox{%
    \reflectbox{\rotatebox[origin=c]{90}%
      {$\m@th#1\mkern2.22mu #2$}}}%
  \dp0=\dimen0 \box0 \mkern2.3965mu
}
\def\nuparrow{\mathrel{
  \mathpalette\@nrotarrow\nrightarrow}}
\def\ndownarrow{\mathrel{
  \mathpalette\@nrotarrow\nleftarrow}}
```

I simboli prodotti sono le negazioni di `\uparrow` e `\downarrow`:

$$\begin{array}{cc} a \uparrow b & a \downarrow b \\ a \nuparrow b & a \ndownarrow b \end{array}$$

Naturalmente non è possibile usare `\nuparrow` e `\ndownarrow` dopo `\left` e `\right` come invece è ammesso con `\uparrow` e `\downarrow`.

14 Casi speciali

TEX considera il codice matematico solo dei caratteri di categoria 11 o 12. Gli altri vengono trattati al modo solito e anche l'espansione delle macro avviene del tutto normalmente, compresa quella dei caratteri attivi. Fanno eccezione i caratteri di categoria 10 (spazio) che in modo matematico sono

TABELLA 1: Tabella del font Marvosym ottenuta con `\xfonttable{U}{mvs}{m}{n}` dando `\fontrange{"40}{AF}`.

	'0	'1	'2	'3	'4	'5	'6	'7	
'10x	@ ₆₄	Ⓐ ₆₅	Ⓜ ₆₆	Ⓒ ₆₇	€ ₆₈	⚡ ₆₉	Ⓢ ₇₀	— ₇₁	"4x
'11x	☺ ₇₂	Ⓜ ₇₃	Ⓢ ₇₄	☕ ₇₅	Ⓜ ₇₆	77	□ ₇₈	□ ₇₉	
'12x	— ₈₀	∞ ₈₁	--- ₈₂	Ⓢ ₈₃	☎ ₈₄	⌚ ₈₅	✓ ₈₆	✂ ₈₇	"5x
'13x	✂ ₈₈	☯ ₈₉	☎ ₉₀	/ ₉₁	✂ ₉₂	≡ ₉₃	≡ ₉₄	/ ₉₅	
'14x	Ⓢ ₉₆	☼ ₉₇	Ⓢ ₉₈	€ ₉₉	€ ₁₀₀	€ ₁₀₁	Ⓢ ₁₀₂	— ₁₀₃	"6x
'15x	☼ ₁₀₄	Ⓢ ₁₀₅	☼ ₁₀₆	☼ ₁₀₇	↓ ₁₀₈	☼ ₁₀₉	☼ ₁₁₀	☼ ₁₁₁	
'16x	— ₁₁₂	✂ ₁₁₃	--- ₁₁₄	✂ ₁₁₅	Ⓢ ₁₁₆	Ⓢ ₁₁₇	☼ ₁₁₈	☼ ₁₁₉	"7x
'17x	Ⓢ ₁₂₀	Ⓢ ₁₂₁	Ⓢ ₁₂₂	○ ₁₂₃	♂ ₁₂₄	♀ ₁₂₅	♀ ₁₂₆	♂ ₁₂₇	
'20x	♂ ₁₂₈	♀ ₁₂₉	□ ₁₃₀	♂ ₁₃₁	♀ ₁₃₂	♀ ₁₃₃	† ₁₃₄	♂ ₁₃₅	"8x
'21x	† ₁₃₆	☼ ₁₃₇	☼ ₁₃₈	☼ ₁₃₉	♥ ₁₄₀	@ ₁₄₁	Ⓢ ₁₄₂	□ ₁₄₃	
'22x	☼ ₁₄₄	● ₁₄₅	● ₁₄₆	■ ₁₄₇	■ ₁₄₈	● ₁₄₉	— ₁₅₀	□ ₁₅₁	"9x
'23x	□ ₁₅₂	Ⓢ ₁₅₃	Ⓢ ₁₅₄	○ ₁₅₅	Ⓢ ₁₅₆	Ⓢ ₁₅₇	Ⓢ ₁₅₈	Ⓢ ₁₅₉	
'24x	Ⓢ ₁₆₀	β ₁₆₁	Ⓢ ₁₆₂	Ⓢ ₁₆₃	€ ₁₆₄	Ⓢ ₁₆₅	\$ ₁₆₆	☼ ₁₆₇	"Ax
'25x	☼ ₁₆₈	☼ ₁₆₉	Ⓢ ₁₇₀	Ⓢ ₁₇₁	Ⓢ ₁₇₂	Ⓢ ₁₇₃	☼ ₁₇₄	☼ ₁₇₅	
	"8	"9	"A	"B	"C	"D	"E	"F	

del tutto ignorati; ne segue che anche un carattere di fine riga (categoria 5) convertito in spazio sarà ignorato e quindi una formula può cominciare in una riga del documento `.tex` e finire in una successiva. Tuttavia non è possibile lasciare righe vuote in un ambiente matematico, perché una riga vuota equivale a un comando `\par` che in modo matematico è illegale.

Un carattere di categoria 11 o 12 può avere codice matematico "8000 che lo rende molto speciale: viene infatti trattato come se fosse un carattere attivo (cioè una macro) e `TeX` ne deve avere una definizione proprio come carattere attivo. L'esempio classico è quello dell'apostrofo:

```
\mathcode'\='8000
{\catcode'\='active
 \gdef'\bgroup\prim@s}}
```

Non ci interessa qui analizzare la definizione, quanto di segnalare questa possibilità. Una definizione ingenua potrebbe essere

```
{\catcode'\='active
 \gdef'\prime}}
```

che però non sarebbe affatto efficiente; ciò che conta è che l'apostrofo in modo matematico si comporta come una macro.

Esistono altri caratteri con codice matematico "8000:

```
\mathcode'\='8000 % \space
\mathcode'\='8000 % ^\prime
\mathcode'\_='8000 % \_
{\catcode'\_='active
 \global\let\_=_}}
```

Queste sono le definizioni in Plain, in `LATEX` sono equivalenti. In realtà, a parte l'apostrofo, servono solo a trattare casi particolari: certe macro potrebbero rendere lo spazio e il carattere `'_` di categoria 11 o 12, con queste definizioni lo spazio verrebbe trattato come uno spazio (il carattere spazio di categoria 13 è definito come `\space`, quindi alla fine sarebbe ignorato). Invece `'_` attivo stampa il carattere stesso: è chiaro, se lo rendiamo 'stampabile', è perché non lo vogliamo più come indicatore dei pedici.

Uno dei vantaggi di questo tipo di 'caratteri matematicamente attivi' è che la loro espansione avviene solo durante la composizione tipografica, non quindi durante la scrittura di file ausiliari e quindi sono automaticamente robusti. Tuttavia non è facile immaginare applicazioni significative di questa possibilità, se non quella di riservare il carattere `'?` per gli allineamenti in colonna di numeri:

```
\mathcode'?'="8000
\begin{group}\lccode'\_?' \lccode'0='0
 \lowercase{\endgroup\def~{\phantom{0}}}
```

La tabella seguente sfrutta questa definizione da mettere nel preambolo: è codice un po' oscuro, ma la magia ha le sue regole.

Unità	Valore (sp)
punto	65 536
millimetro	186 467
punto Didot	70 124
pollice	4 736 286

È in generale molto complicato allineare in colonna numeri sotto un'intestazione più larga dei numeri stessi, se vogliamo che l'intestazione sia centrata. La soluzione consiste nel definire la colonna con allineamento al centro e rendere tutti i numeri della stessa ampiezza. Nel preambolo si è data la definizione richiesta del punto interrogativo come attivo in modo matematico e la tabella è inserita con

```
\begin{tabular}{l>{$}c<{$}}
\toprule
\multicolumn{1}{c}{Unità} &
$Valore (sp)$\\
\midrule
punto & ?\,?65\,536\\
millimetro & ?\,186\,467\\
punto Didot & ?\,?70\,124\\
pollice & 4\,736\,286\\
\bottomrule
\end{tabular}
```

Si è usato il pacchetto `array` per specificare che la seconda colonna è in modo matematico, i caratteri \$ servono, nell'intestazione, ad annullare l'effetto di quelli inseriti a inizio e fine della cella. Il punto interrogativo inserisce uno spazio equivalente a una cifra; naturalmente occorre usare un font in cui tutte le cifre abbiano la stessa larghezza, ma questa regola vale in generale per tabelle con dati numerici. Il prezzo da pagare è l'impossibilità di lasciare i dati così come sono; la rinuncia al carattere ? in modo matematico non sembra una grave perdita.

Non è l'unico modo, ma ha il pregio di essere automatico. La stessa tabella si può ottenere senza usare il trucco di prima, eseguendo alcune misure possibili solo quando si conoscano già i dati della tabella: qui si è calcolata la differenza tra l'ampiezza dell'intestazione e quella del numero maggiore; divisa per due dice di quanto spostare l'intestazione verso destra e altrettanto spazio va inserito in più tra prima e seconda colonna che, però, avrà un allineamento a destra.

Unità	Valore (sp)
Punto	65 536
Millimetro	186 467
Punto Didot	70 124
Pollice	4 736 286

Il codice, nella parte iniziale, è

```
\setlength{\dimen0}{
(\widthof{Valore (sp)}-
\widthof{4\,736\,286})/2
}
\begin{tabular}{lr!{\hspace{\dimen0}}}{
\toprule
...

```

che usa le operazioni messe a disposizione dal pacchetto `calc`.

Un secondo uso dei caratteri attivi in modo matematico è per poter usare la virgola come delimitatore della parte decimale di un numero. Come abbiamo visto, il codice matematico della virgola corrisponde a un segno di punteggiatura, perché l'uso maggiore è, almeno nei paesi di lingua inglese, per formule del tipo (x, y) . In Italia si usa invece la virgola per dividere la parte intera da quella decimale; il trucco più semplice per ingannare TEX è di scrivere `1{,}05`, ma questo può essere noioso e porta a errori di composizione se ci si dimentica delle graffe. Una possibile soluzione è di usare il pacchetto `icomma` di Walter Schmidt il cui contenuto è, con modifiche inessenziali,

```
\AtBeginDocument{%
\mathchardef\mathcomma\mathcode'\,
\mathcode'\,="8000 }
{\catcode'\,=\active
\gdef{\futurelet\@let@token\sm@rtcomma}}
\def\sm@rtcomma{%
\ifx\@let@token\@sptoken\else
\mathord\fi\mathcomma}
```

Non entriamo nel dettaglio di `\futurelet`; il funzionamento è in sostanza questo: quando TEX incontra una virgola in modo matematico, osserva anche il *token* che la segue (il carattere o comando successivo, per intenderci); se questo *token* non è uno spazio, ciò che verrà eseguito è `\mathord\mathcomma`, mentre se il token è uno spazio (`\@sptoken` è gergo LATEX per uno spazio) verrà eseguito solo `\mathcomma`. Questo comando è definito usando il codice matematico della virgola, lo si fa a inizio del documento per evitare problemi nel caso si usino pacchetti che assegnano alla virgola un codice matematico insolito.

Il pacchetto dunque permette di scrivere `$1,05$` e di ottenere 1,05; se si vuole una virgola come segno di punteggiatura, occorre lasciare uno spazio (o un fine riga), come in `$(a, b)$` per avere (a, b) . Un piccolo difetto di questo approccio è che richiede una grande disciplina nella scrittura del documento `.tex`, ma è certamente conveniente se si hanno parecchi numeri decimali da scrivere in modo matematico.

Avvertenza

Si faccia attenzione quando si usa `babel`, perché quasi tutti i suoi moduli per le lingue rendono attivo il carattere ", producendo misteriosi errori quando si vogliono passare a TEX numeri in formato esadecimale. In queste note abbiamo finto di scrivere "1234, dove in effetti abbiamo scritto `\string"1234`, che evita il problema. Un modo diverso di evitarlo è di scrivere il codice tra

```
\shorthandoff{"}
...
\shorthandon{"}
```

oppure usare un gruppo come in

```
{\shorthandoff{"}...}
```

Nel preambolo del documento il problema non c'è, perché " è attivato solo nell'ambiente `document`, purché non si sia dato il comando

```
\selectlanguage{italian}
```

o altro simile, che però non andrebbe mai dato nel preambolo. L'uso delle tabelle di font prodotte con i comandi di `fonttable` permette di conoscere il codice decimale dei caratteri da usare. Dunque il doppio apice può dare problemi solo se si dovesse usare nel documento un comando esplicito come `\mathchar"613B`, cosa del tutto sconsigliabile: meglio definire una macro nel preambolo.

Riferimenti bibliografici

BECCARI, C. (1997). «Typesetting mathematics for science and technology according to ISO 31/XI». *TUGboat*, **18** (1), pp. 39–48.

BECCARI, C. (a cura di) (2009). *Introduzione all'arte della composizione tipografica*. GfT. URL <http://www.guit.sssup.it>.

GUIGGIANI, M. e MORI, L. F. (2008). «Consigli su come *non* maltrattare le formule matematiche». *ArsT_EXnica*, **6**, pp. 5–14.

▷ Enrico Gregorio
Dipartimento di Informatica
Università di Verona
Enrico dot Gregorio at univr
dot it