Il concorso della scacchiera

Claudio Beccari, Roberto Giacomelli, Maurizio Molinaro

Sommario

Questo articolo descrive tre soluzioni al problema della scacchiera. Ciascuna è prodotta da uno dei tre autori e le soluzioni sono molto diverse l'una dall'altra. È molto interessante notare il diverso approccio che ogni autore ha usato per affrontare il problema.

Abstract

This article describes three solutions to the problem of designing a chess board. Each solution is created by a different author and it is very interesting to note the different approach they had in facing the problem.

1 La sfida

Nell'ottobre 2017 lo staff del GUIT ha lanciato una specie di concorso fra gli utenti di TEX. Esso consisteva nel presentare una soluzione al seguente semplice problema:

> Creare il software necessario per disegnare una scacchiera con un numero specificato di righe e colonne che abbia una casella nera nel quadretto in basso a sinistra e che l'intera scacchiera sia poi riempita alternativamente di celle bianche e nere alternate.
>
> Si può usare qualunque dei tre programmi di composizione basati su LATEX: pdfLATEX, XeLATEX e LuaLATEX; si possono usare tutti i pacchetti del sistema TEX tranne quelli che già disegnano cose simili e ovviamente non si possono "copiare" le macro di questi pacchetti.

In teoria questo gioco doveva costituire un concorso e le varie soluzioni avrebbero dovuto essere sottoposte al giudizio del Consiglio Scientifico che avrebbe decretato la migliore di queste soluzioni.

Questo guanto di sfida non è stato raccolto da un numero sufficiente di utenti di TEX: la sfida avrebbe avuto il giudizio del Consiglio Scientifico solo se i concorrenti fossero stati almeno tre.

Tuttavia alcuni membri dello staff del GUIT si sono cimentati a loro volta, se non altro per controllare che il problema non fosse tanto difficile da scoraggiare i possibili concorrenti. Le loro soluzioni li hanno convinti che il problema era sufficientemente semplice e hanno lanciato il guanto.

Forse il problema era troppo facile? Forse. Tuttavia, senza mettere a confronto di merito le varie soluzioni, qui se ne descrivono tre:

- una semplice soluzione di Maurizio Molinaro, molto lineare, basata sul pacchetto TikZ;
- una soluzione di Claudio Beccari, molto estesa nelle prestazioni, ma basata solo sulle funzionalità grafiche dell'ambiente picture descritto da Lamport (1994) nella versione del 1994 del mark-up LATEX;
- una soluzione di Roberto Giacomelli, noto ai lettori di ArsTEXnica per la capacità di usare creativamente LuaLATEX e il linguaggio Lua.

Il bando della sfida non specificava se i colori della scacchiera dovessero essere limitati al bianco e al nero; infatti i tre autori hanno impostato le loro soluzioni in modo da consentire l'uso anche di altri colori.

Il bando non specificava nemmeno che i numeri delle righe e delle colonne dovessero essere uguali tanto da costruire una scacchiera di $N \times N$, invece che di $N \times M$ righe e colonne. O meglio, lasciava intendere, anche mediante l'uso della parola "scacchiera", che dovesse trattarsi di un oggetto quadrato. Infatti la soluzione quadrata è stata presunta da tutti e tre gli autori, ma Beccari ha previsto un'opzione per una scacchiera rettangolare e Giacomelli ha previsto che i numeri di righe e colonne vengano sempre specificati entrambi anche se sono uguali.

Il bando non escludeva l'uso di nessun pacchetto, tranne, ovviamente, quelli già esistenti anche se non adattati al numero arbitrario di righe e colonne, come per esempio Fischer (2014); o meglio: questo pacchetto è previsto per il gioco degli scacchi, ma è in grado di rappresentare anche zone parziali rettangolari della scacchiera.

Tutte e tre le soluzioni sono originali; il lettore può esaminarle con attenzione e magari scoprire metodi di programmazione che potrebbero servirgli anche in altre circostanze.

In questo articolo i comandi e/o gli ambienti descritti saranno sempre chiamati col nome "scacchiera". In realtà i pacchettini che contengono i tre codici sono stati leggermente modificati per distinguerli l'uno dall'altro; si ritiene infatti che i lettore interessato possa tranquillamente capire la differenza d'uso che ne viene fatto nei paragrafi che seguono.

```
\ProvidesPackage{ScacchieraMM}[2018-02-05 1.2 Comando per generare una scacchiera]

\usepackage{framed}
\usepackage{ifthen}
\usepackage{tikz}

% Definizione degli indici per i cicli
\newcounter{k1}
\newcounter{k2}

\newcommand{\scacchieraMM}[2]{%
\ifthenelse{#1 < 2 \or #1 > 20}
{\textit{La dimensione richiesta è #1,\\
 e invece dev'essere compresa tra 2 e 20!}}
{\ifthenelse{\isodd{#1}}%
% d dispari
{\setcounter{k1}{\numexpr (#1-1)/2}
 \setcounter{k2}{\numexpr (#1-3)/2}}
% d pari
{\setcounter{k1}{\numexpr (#1-2)/2}
 \setcounter{k2}{\numexpr (#1-2)/2}}

\begin{tikzpicture}[chiara/.style={white}, scura/.style={black!20!white}, scale=#2]
% caselle chiare
\fill[chiara] rectangle (#1,#1);
% primo ciclo
\foreach \m in {0, ..., \value{k1}}
\foreach \n in {0, ..., \value{k1}}
 \fill[scura] (2*\n,2*\m) rectangle +(1,1);
% secondo ciclo
\foreach \m in {0, ..., \value{k2}}
\foreach \n in {0, ..., \value{k2}}
 \fill[scura] (2*\n+1,2*\m+1) rectangle +(1,1);
% contorno
\draw (0,0) rectangle (#1,#1);
\end{tikzpicture}}
}
```

Codice 1: Codice della soluzione di Molinaro

2 La soluzione di Molinaro

Molinaro è l'unico utente di LaTeX, non facente parte dello staff del GUIT, che ha raccolto il guanto di sfida.

Questa è una soluzione estremamente lineare basata sul pacchetto TikZ; Molinaro definisce il comando `\scacchiera` che accetta in entrata il ⟨*numero*⟩ di righe e la ⟨*dimensione*⟩ del lato di ogni cella. Si assume che la scacchiera sia quadrata ed entrambi i comandi obbligatori, quindi la sintassi è la seguente:

```
\scacchieraMM{⟨numero⟩}{⟨dimensione⟩}
```

Se si vogliono le caselle di colore diverso dal nero, bisogna ridefinire le opzioni `chiara` e `scura` presenti nel comando di apertura dell'ambiente tikzpicture; d'altra parte non sarebbe difficile definire due colori, diciamo `colorechiaro` e `colorescuro`, e in quelle opzioni del comando di apertura definire le suddette opzioni in termini di questi colori.

Il codice 1 comincia con la gestione dei limiti accettabili per il numero delle caselle delle righe e colonne; se il numero è inferiore a 2 o superiore a 20, viene emesso un messaggio d'errore e non viene eseguito il disegno.

Se il numero esce dai limiti predetti si ottiene un messaggio d'errore:

> *La dimensione richiesta è 1,*
> *e invece dev'essere compresa tra 2 e 20!*

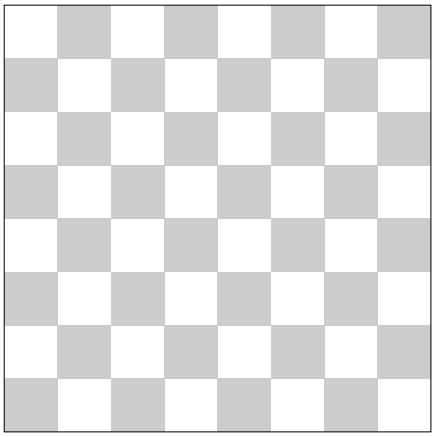
In vista di due cicli `for` per inserire le caselle chiare e scure nella tabella, vengono caricati due contatori LaTeX, *k1* e *k2*, con cui contare il numero di ripetizioni delle inclusioni delle celle colorate di scuro sopra un background già caricato di celle colorate di chiaro.

Finalmente vengono eseguiti i vari cicli `for` e le caselle vengono messe in posizione.

Complessivamente il file di estensione con i comandi appena descritti contiene quanto appare nel codice 1 nella pagina 30.

Alcuni esempi compaiono nella figura 2.

Scacchiera 8x8 con lato 0,75 cm



Scacchiera 9x9 con lato 0,5 cm

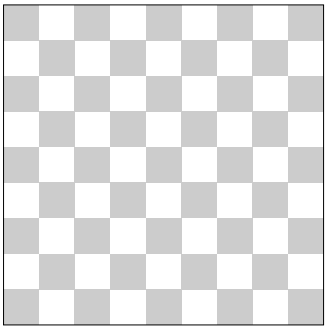


Figura 2: Due scacchiere con numeri di righe

3 La soluzione di Beccari

Beccari è un utente convinto della bontà dell'ambiente picture e non usa spesso TikZ, pur riconoscendone le infinite funzionalità. Egli ritiene che il semplice ambiente picture del nucleo di LaTeX sia altrettanto valido per certi lavori particolari, e quello della scacchiera è uno di questi. Egli usa abitualmente le estensioni documentate da Lamport (1994) nel lontano 1994, diventate effettive quasi 10 anni dopo con la pubblicazione del pacchetto pict2e (Gäßlein *et al.*, 2016) e a cui egli stesso diede qualche contributo.

Egli ha anche sperimentato le funzionalità dei pacchetti etoolbox (Lehman e Wright, 2017) e xparse (The LaTeX3 project, 2017); il primo offre molte funzionalità interessantissime per chi scrive macro; il secondo mette a disposizione dei comandi potentissimi per definire nuovi comandi e nuovi ambienti che accettano una moltitudine di argomenti diversi, delimitati e non delimitati; obbligatori o facoltativi, e anche semplici token, il più comune dei quali è l'asterisco (già usato dal nucleo di LaTeX per alcuni ambienti e per alcuni comandi). Egli usa anche abitualmente le estensioni del linguaggio nativo di TeX descritte nella documentazione di etex (The $\mathcal{NTS}$ Team, 1998) e che da diversi anni sono incorporate in tutti gli interpreti del sistema TeX.

Perciò il suo pacchetto scacchieraCB è dipendente da ε-TeX, ma non c'è bisogno di invocare nulla perché le sue funzionalità, benché poco usate dagli utenti del sistema TeX, fanno già parte dei suoi motori di composizione; il pacchetto scacchieraCB dipende da pict2e, etoolbox e xparse che vanno caricati *prima*. Utile, se si vogliono usare le "color expressions", è il pacchetto xcolor, il cui ordine di caricamento non è rilevante.

Beccari ha messo insieme questi tre pacchetti e l'ambiente picture per definire alcune macro che permettono di costruire sia scacchiere sia cruciver-

ba con una moltitudine di varianti sfruttando le potenti funzionalità, specialmente quelle di xparse. Egli ha calcolato che la sua macro di base potrebbe accettare argomenti facoltativi e obbligatori in 128 modi diversi. Il lettore non si spaventi, però: di questi 128 modi diversi in pratica se ne usano una dozzina, ma quel numero la dice lunga sulla potenza delle funzionalità dei comandi definiti con xparse.

Chi scrive si scusa per questa lunga premessa, ma siccome le scacchiere di Beccari e i suoi cruciverba sono molto variegati, era necessario spiegare come e perché.

In sostanza, Beccari definisce un comando `\scacchierainterna` sul quale si basano gli altri comandi e ambienti: il comando `\ComandoScacchiera` e gli ambienti scacchiera e cruciverba. Questi sono poi affiancati da alcune macro che servono per collocare delle cose particolari in celle scelte della scacchiera o del cruciverba: `\GenBox`, `\ScacBox`, `\NumBox`, oltre al comando `\setfontsize` per impostare in modo più semplice il corpo dei caratteri senza dover perder tempo a calcolarne a mano lo scartamento. Egli ha definito anche i comandi `\usecs` e `\whilenum`: il primo molto simile ad un comando di etoolbox; il secondo per evitare di usare il comando interno del nucleo di LaTeX `\@whilenum` che, contenendo il carattere `@`, non è usabile direttamente dall'utente se copia il codice 4 nella pagina 35 nel suo preambolo (invece nei file `.sty` il carattere `@` è considerato una lettera ed è perfettamente lecito).

3.1 La scacchiera di base

La scacchiera di base è formata dallo sfondo, anche colorato di un colore diverso dal bianco, con il reticolo dei quadretti. Dentro questa scacchiera di N righe e M colonne possono apparire caselle scure, di colore anche diverso dal nero, alternati alle caselle chiare (come specificato dal bando del concorso), ma possono apparire anche caselle scure in posizioni "random", scelte dall'utente senza una regola particolare. Nel caso dei cruciverba, caso particolare di scacchiere, possono apparire anche i numerini delle definizioni orizzontali e verticali.

La particolarità della scacchiera di base con le caselle scure prescritte dal concorso è l'algoritmo con cui vengono collocate; esse infatti si trovano solo nelle caselle diagonali dirette da nord ovest a sud est. Tenuto conto delle coordinate di ciascuna di queste caselle, si constata che la somma dei loro indici (ascissa e ordinata) è sempre un numero pari (si ricordi che l'origine degli assi coincide con il vertice in basso a sinistra della scacchiera complessiva, e le coordinate di ciascuna casella sono quelle del suo vertice in basso a sinistra). In questo modo, scorrendo con due cicli gli indici delle righe e delle colonne, ci vuole una casella scura quando la loro somma è pari. Nel codice 4 della pagina 35 si riconoscono infatti le quattro righe che svolgono questa operazione:

```
\I=0\J=0\relax
\whilenum{\N > \I}{{\whilenum{\M > \J}{\unless
\ifodd\numexpr\I+\J \put(\I,\J){\nero}\fi
\J=\numexpr\J+1}}\I=\numexpr\I+1}}%
```

Si notino i comandi `\unless` e `\numexpr` di ε-TeX; il primo rovescia l'esito di un test; il secondo esegue i calcoli nei registri della CPU del calcolatore, invece di usare i registri interni del programma di composizione. Forse in questo caso sono eccessivi [1], ma in generale è meglio ricorrere a questi comandi di ε-TeX, piuttosto che ai comandi nativi o a quelli messi a disposizione dal pacchetto calc (Thorup *et al.*, 2014), raccomandazione valevole anche per i calcoli sulle lunghezze rigide ed elastiche.

Nel codice 4 si nota anche l'uso dei comandi di base dell'interprete del linguaggio LaTeX: `\countdef` e `\dimendef` usati con numeri di registri numerici e dimensionali maggiori di 255. Questi registri sono usabili grazie a ε-TeX; assegnare loro un nome simbolico evita di commettere errori nella programmazione, ma è importante però invocare tali operazioni dentro un gruppo o un ambiente, cosicché alla chiusura del gruppo o dell'ambiente quei registri vengano ripristinati ai valori che essi avevano prima della loro apertura. Ciò evita spiacevoli interferenze con l'uso di quei registri da parte di altre funzioni del programma che si sta eseguendo. A questo scopo servono anche i comandi `\bgroup` ed `\egroup`, che rappresentano delle graffe implicite più visibili delle graffe esplicite ma usabili anche in situazioni insolite[2].

Disponendo di font con i disegni dei pezzi della dama o degli scacchi si possono definire altre macro di posizionamento di diversi oggetti nelle caselle; ma mentre le definizioni presenti nel file di estensione `ScacchieraCB.sty` restano disponibili, esse possono servire da modello per creare altri comandi per collocare questi particolari oggetti; non se ne parla qui, perché esula dall'argomento di questo articolo.

La sintassi per creare questa scacchiera di base è fornita dal comando `\scacchierainterna` (vedi il codice 4 nella pagina 35)

1. Senza usare i comandi di ε-TeX, il comando per gestire i contatori TeX sarebbe `\advance \I by 1 \relax`; nel nostro caso non ci sarebbe una grossa differenza, ma per espressioni più complesse bisogna scrivere altri comandi che implicano caricare e scaricare i risultati intermedi in macro intermedie; il pacchetto calc offre un'interfaccia più comoda, ma il via vai fra le macro interne resta immutato. calc era utile e necessario prima che ε-TeX diventasse parte integrante degli interpreti del sistema TeX.

2. Quanto delimitato da graffe esplicite deve contenere un numero intero di graffe accoppiate `{}`, anche annidate, altrimenti l'interprete segnala un errore. Le situazioni insolite sono quando il gruppo viene aperto con una macro e chiuso con un'altra, situazioni che talvolta è necessario affrontare, ma non in questo pacchetto.

```
\scacchierainterna{⟨*⟩}[⟨largh⟩]%
  {⟨righe⟩,⟨colonne⟩}{⟨casella
scura⟩,⟨sfondo⟩}%
  (⟨x_off⟩,⟨y_off⟩)
```

i cui argomenti sono:

⟨*⟩ facoltativo; se presente genera la scacchiera di base, senza nulla nelle caselle; se assente genera la scacchiera di base con le caselle scure disposte come previsto dal bando di concorso.

⟨*largh*⟩ facoltativo; indica la larghezza esplicita dell'intera scacchiera; di default vale `0.7\linewidth`, per cui può essere usato sia per composizione a piena pagina, sia per la composizione in due colonne. Sulla base di questo valore e in base al numero di colonne della scacchiera viene calcolata la `\unitlength` dell'ambiente grafico picture.

⟨*righe*⟩ obbligatorio; specifica il numero N di righe dalla scacchiera.

⟨*colonne*⟩ facoltativo; se presente specifica il numero M delle colonne, se assente viene assunto pari a N; se è assente non è necessario usare la virgola di separazione.

⟨*casella scura*⟩ e ⟨*sfondo*⟩ facoltativi nonostante siano racchiusi fra graffe; se ⟨*casella scura*⟩ è presente specifica il colore delle caselle scure; di default è `black`; se non lo si vuole specificare, ma si vuole specificare il colore dello ⟨*sfondo*⟩, bisogna indicare anche la virgola; se non si vuole specificare lo ⟨*sfondo*⟩ si può specificare la virgola, ma non è necessario; il colore di default dello ⟨*sfondo*⟩ è `white`. Per i nomi dei colori si devono usare i nomi inglesi definiti dal pacchetto xcolor (Kern, 2016), o altri colori definiti mediante quel pacchetto; con xcolor, si possono usare le *color expressions*, ma negli argomenti del pacchetto scacchieraCB è necessario "nascondere" queste espressioni dentro un gruppo esplicito, perché i punti esclamativi che vi sono usati interferirebbero con il funzionamento di questo pacchetto; si scriverà pertanto `{blue!50!white}` invece di `blue!50!white`.

⟨x_{off}⟩ e ⟨y_{off}⟩ sono facoltativi, ma se specificati devono esserlo entrambi; servono per spostare l'intera scacchiera in alto con il valore ⟨y_{off}⟩ positivo, e a destra con ⟨x_{off}⟩ positivo. Di default essi valgono entrambi zero; essi vanno espressi in unità del grafico cioè come prefisso dell'unità di misura `\unitelength`

Come si capisce dal numero di informazioni facoltative, le possibilità di quelle presenti o assenti sono 7, quindi ci sono $2^7 = 128$ combinazioni. Ovviamente non sono tutte utili (per esempio chi specificherebbe `{blue,}` invece di `{blue}`? Chi specificherebbe `{,}` o `{}` invece di evitare completamente di esprimere l'opzione?), ma ci sono.

Si noti che `\scacchierainterna` è solo di servizio per i comandi e gli ambienti del prossimo paragrafo, infatti esso apre l'ambiente picture ma non lo chiude incondizionatamente; lo chiude solo se si usa il comando `\ComandoScacchiera`.

3.2 L'ambiente scacchiera

In apertura l'ambiente scacchiera imposta a `false` la variabile booleana `ScacCommand` e poi chiama il comando `\scacchierainterna` senza argomenti (argomenti che però `\scacchierainterna` legge come prima cosa in apertura dell'ambiente); in chiusura semplicemente chiude l'ambiente picture, quindi fra l'apertura e la chiusura si possono inserire ⟨*altri oggetti*⟩; la sintassi perciò è la seguente:

```
\begin{scacchiera}⟨argomenti
  per \scacchierainterna⟩
⟨altri oggetti⟩
\end{scacchiera}
```

Disegna una scacchiera con le caselle scure alternate con le caselle chiare (se non viene specificato l'asterisco, per il quale si veda il prossimo comando), come specificato dal bando; gli ⟨*altri oggetti*⟩ possono essere inseriti mediante i comandi `\GenBox` e, eventualmente, con `\ScacBox` e `\NumBox`, o con qualunque altro comando valido nell'ambiente picture.

3.3 L'ambiente cruciverba

L'ambiente cruciverba è molto simile all'ambiente scacchiera ma usa l'asterisco senza che debba farlo l'utente; non inserisce nessuna casella scura; la sintassi è la stessa:

```
\begin{cruciverba}⟨argomenti per
  \scacchierainterna senza asterisco⟩
⟨altri oggetti⟩
\end{cruciverba}
```

3.4 Le scatole speciali

Come già detto, sono disponibili tre scatole speciali con le seguenti sintassi:

```
\GenBox(⟨x,y⟩)[⟨posizione⟩](⟨dimensioni⟩)%
  {⟨oggetto⟩}
\NumBox(⟨x⟩,⟨y⟩)margnumero{⟨corpo⟩}
\Scacbox(⟨x⟩,⟨y⟩)
```

e con gli argomenti seguenti:

⟨*x,y*⟩ coppia di valori separati da virgola; obbligatori. Essi sono le coordinate dello spigolo inferiore sinistro della cella dove si vuole inserire qualche cosa. Di fatto sono numeri interi, perché l'unità di misura grafica è data proprio dalla lunghezza del lato del quadrato della cella. Si tenga presente che le coordinate della prima cella in basso a sinistra dell'intera scacchiera ha coordinate `(0,0)`.

⟨*posizione*⟩ facoltativa; è indicata da una coppia di lettere non inconsistenti scelte fra `l, r, t, b, c` per indicare dove si vuole mettere l'⟨*oggetto*⟩ dentro la cella. In sostanza sono gli stessi codici di posizione che si usano in ambiente picture come argomento facoltativo al comando `\makebox`. L'espressione "non inconsistenti" ricorda che ci sono degli accoppiamenti inconsistenti, come `[tb]`, perché un oggetto non può essere contemporaneamente allineato in alto e in basso.

⟨*dimensioni*⟩ facoltative; sono le dimensioni espresse in frazioni di `\unitlength` della scatola che contiene l'oggetto; di default valgono `(1,1)`. Per esempio si può specificare `(0.5,0.5)` per specificare una scatola (quadrata) il cui lato è la metà di quello della cella.

⟨*oggetto*⟩ obbligatorio; è quanto si vuole inserire in una cella; può essere qualunque cosa che l'ambiente picture accetti e sappia mettere in posizione.

⟨*numero*⟩ obbligatorio: è il ⟨*numero*⟩ che nei cruciverba va posto ad una piccola distanza dell'angolo superiore sinistro di una casella chiara.

⟨*corpo*⟩ facoltativo; è costituito dal coefficiente da dare a `\unitlength` per definire il corpo del ⟨*numero*⟩; di default vale `{0.3}`. È raro doversene servire, ma il comando che ne fa uso, `\setfontsize`, viene descritto nel prossimo paragrafo ed è a disposizione dell'utente, qualora lo volesse usare. Si veda la spiegazione di `\setfontsize` qui sotto.

3.5 Il corpo dei numeri dei cruciverba

Il corpo del numero delle celle numerate dei cruciverba, come si è visto sopra, si può scalare al valore desiderato.

Il comando che produce l'effetto ha la sintassi seguente:

`\setfontsize[`⟨*allargamento*⟩`]{`⟨*corpo*⟩`}`

La ⟨*dimensione del corpo*⟩ è una qualunque dimensione esplicita, per esempio, `{2mm}`, `{0.5\unitlength}`, e simili, o un numero reale senza l'indicazione dell'unità di misura che allora viene assunta di default pari al punto tipografico. L'⟨*allargamento*⟩ di default è impostato pari a `1.2`, ma opzionalmente l'utente può specificare valori un poco maggiori o minori. Il comando di fatto imposta l'⟨*allargamento*⟩ come fattore del parametro `\linespread` e imposta con `\fontsize` uno scartamento pari al corpo; poi dà il comando `\selectfont` che in pratica modifica lo scartamento per il fattore specificato con l'argomento facoltativo. I calcoli, quindi, vengono eseguiti dai comandi del nucleo di LaTeX. La definizione di `\setfontsize` è nelle ultime due righe del codice 4 nella pagina 35.

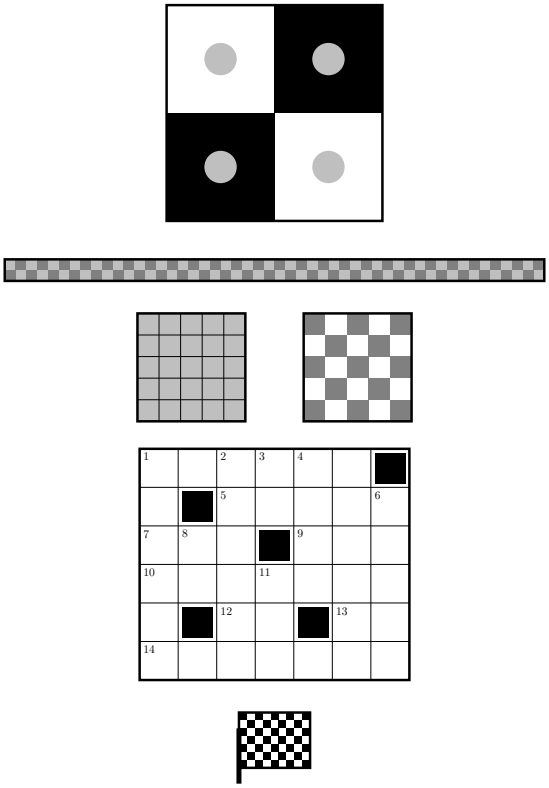


Figura 3: Alcuni esempi costruiti con il pacchetto scacchieraCB

Si noti, però, che questo comando opera correttamente solo con font continuamente scalabili, quali sono i font OpenType usati in questo testo, i Latin Modern, sempre consigliabili quando si compone con pdfLaTeX, i font Type 1 in generale. Non funziona invece con i font Computer Modern (semplici o super) a causa di come sono definiti i loro file di descrizione `.fd`. In realtà non sarebbe impossibile modificare questi file, ma cesserebbero di funzionare volendo ottenere il file finale in formato DVI con questi font; si veda in particolare Beccari (2017, figura 2.1).

3.6 Alcuni esempi

Nella figura 3 ci sono alcuni esempi costruiti con il software contenuto nel pacchetto scacchieraCB. Non sono esposti esempi colorati con colori vivaci, perché questa rivista viene stampata in bianco e nero e tonalità di grigio; d'altra parte l'uso dei colori 'gray' e 'lightgray' è sufficientemente esemplificativo, nonostante l'aspetto visivo di quei colori sia modesto.

1. Il primo e il quarto esempio mostrano due scacchiere costruite rispettivamente con l'ambiente scacchiera e il comando `\ComandoScacchiera` senza usare l'asterisco; nel primo si è usato il comando `\GenBox` per sovrapporre in tutte le celle il pallino grigio; i codici sono i rispettivamente i seguenti:

```
\ProvidesPackage{ScacchieraCB}[2018-02-10 v.1.0 Per creare scacchiere e cruciverba]
%
\providecommand*\usecs[1]{\csname#1\endcsname}
\providecommand*\whilenum[2]{\usecs{@whilenum}#1\do{#2}}
%
\newif\ifScacCommand \ScacCommandfalse
\newcommand\ComandoScacchiera{\bgroup\ScacCommandtrue\scacchierainterna}
\newenvironment{cruciverba}{\ScacCommandfalse\scacchierainterna*}{\end{picture}}
\newenvironment{scacchieraCB}{\ScacCommandfalse\scacchierainterna}{\end{picture}}
%                                                                    Scacchiera di base
\NewDocumentCommand\scacchierainterna{s O{0.7\linewidth} m G{black,white} D(){0,0}}{%
%                                                                      Registri
\countdef\N=256   \countdef\M=260%
\countdef\I=258   \countdef\J=262%
\countdef\No=264  \countdef\Mo=266
\def\splitOffsets##1,##2!{\No=##1\relax\Mo=##2}
\splitOffsets#5!%
\def\toglivirgola##1,!{##1}%
\def\righecolonne##1,##2!{\M=##1\relax%                          \M righe e \N colonne
\def\ScacA{##2}\relax
\ifx\ScacA\empty
   \N=\M\relax
\else
   \expandafter\N\toglivirgola##2!\relax
\fi
\ifnum\M<1\relax\M=1\fi
\ifnum\N<1\relax\N=1\fi}%
\def\colori##1,##2!{%
  \def\scacco{##1}\ifx\scacco\empty
  \def\scacco{\color{black}}\else
  \def\scacco{\color{##1}}\fi
  \def\sfondo{##2}\ifx\sfondo\empty\def\sfondo{\color{white}}\else
  \ifdefstring{\sfondo}{,}{\def\sfondo{\color{white}}}%
  {\edef\sfondo{\noexpand\color{\toglivirgola##2!}}}\fi
}%
\def\Colori{#4}%
\ifx\Colori\empty\colori,!\else\colori#4,!\fi
%
\righecolonne#3,!%
\dimendef\Num=256\dimendef\Den=258%
\Num=1pt\Den=\N\Num
\unitlength=\dimexpr #2*\Num/\Den\relax
\I=\numexpr\N+\No\relax
\J=\numexpr\M+\Mo\relax
%                    Disegno della griglia con contorno e sfondo con o senza contenuti
\noindent\begin{picture}(\I,\J)(-\No,-\Mo)
\put(0,0){\sfondo\polygon*(0,0)(\N,0)(\N,\M)(0,\M)}%
\IfBooleanTF{#1}{%                                        Se ''true'' si fa cruciverba
\def\nero{\scacco\polygon*(0.1,0.1)(0.9,0.1)(0.9,0.9)(0.1,0.9)}%
\multiput(0,0)(1,0){\numexpr\N+1}{\line(0,1){\M}}
\multiput(0,0)(0,1){\numexpr\M+1}{\line(1,0){\N}}
}{%                                                    altrimenti si fa scacchiera
\def\nero{\scacco\polygon*(0,0)(1,0)(1,1)(0,1)}%
\I=0\J=0\relax
\whilenum{\N > \I}{{\whilenum{\M > \J}{\unless
\ifodd\numexpr\I+\J \put(\I,\J){\nero}\fi
\J=\numexpr\J+1\relax}}\I=\numexpr\I+1\relax}}%
\put(0,0){\linethickness{1pt}\polygon(0,0)(\N,0)(\N,\M)(0,\M)}
\ifScacCommand\end{picture}\egroup\fi}
%                                                                        Scatole
\NewDocumentCommand\GenBox{ D(){0,0} O{cc} D(){1,1} m }{%
  \put(#1){\makebox(1,1)[bl]{\makebox(#3)[#2]{#4}}}}
\NewDocumentCommand\NumBox{D(){0,0} m G{0.3}}{%
  \put(#1){\makebox(1,1)[cc]{\makebox(0.8,0.8)[tl]{\setfontsize{#3\unitlength}#2}}}}
\def\ScacBox(#1){\put(#1){\makebox(1,1)[bl]{\nero}}}
%                                                                  Corpo dei font
\newcommand*\setfontsize[2][1.2]{%
\linespread{#1}\fontsize{#2}{#2}\selectfont}
```

Codice 4: Codice della soluzione di Beccari

```
\begin{scacchiera}[0.4\linewidth]{2}
  \GenBox(0,0){\color{lightgray}\circle*{0.3}}
  \GenBox(1,1){\color{lightgray}\circle*{0.3}}
  \GenBox(1,0){\color{lightgray}\circle*{0.3}}
  \GenBox(0,1){\color{lightgray}\circle*{0.3}}
\end{scacchiera}

\ComandoScacchiera[0.2\linewidth]{5}%
    {,lightgray}
```

2. Il secondo ed il terzo esempio sono creati entrambi con `\ComandoScacchiera` ma nel terzo si è usato l'asterisco, mentre nel secondo si sono specificate 2 righe e 50 colonne. I codici sono rispettivamente i seguenti:

```
\ComandoScacchiera[\linewidth]{2,50}%
    {gray,lightgray}

\ComandoScacchiera*[0.2\linewidth]{5}%
    {,lightgray}
```

3. Il quinto esempio mostra un cruciverba; il suo codice è piuttosto lungo, perché ogni casella nera e ogni numerino va introdotto con un comando apposta, in quanto le coordinate delle caselle dove mettere quegli oggetti non sono determinate a priori.

```
\begin{cruciverba}[0.5\linewidth]{6,7}
  \ScacBox(1,1)   \ScacBox(3,3)
  \ScacBox(1,4)   \ScacBox(4,1)
  \ScacBox(6,5)
  \NumBox(0,5){1} \NumBox(1,3){8}
  \NumBox(2,5){2} \NumBox(4,3){9}
  \NumBox(3,5){3} \NumBox(0,2){10}
  \NumBox(4,5){4} \NumBox(3,2){11}
  \NumBox(2,4){5} \NumBox(2,1){12}
  \NumBox(6,4){6} \NumBox(5,1){13}
  \NumBox(0,3){7} \NumBox(0,0){14}
\end{cruciverba}
```

4. Il sesto esempio mostra quella bandiera con il suo manico che viene sventolata nelle corse di Formula 1 al passaggio per il traguardo finale. Potrebbe essere un'alternativa (insolita) all'uso del Q.E.D. alla fine delle dimostrazioni matematiche. Il codice è il seguente:

```
\begin{scacchiera}[10mm]{7,9}
\put(0,-2,){\makebox(0,0)[bl]{%
   \linethickness{2pt}\line(0,1){7}}}
\end{scacchiera}
```

   e vi si mostra l'uso di comandi propri dell'ambiente picture all'interno dell'ambiente.

4 Intermezzo

Vale la pena di ricordare che i programmi `latex`, `pdflatex`, `xelatex` e `lualatex` non disegnano direttamente nulla; preparano un file di uscita che contiene dei comandi nativi di TEX, `\special`, il cui argomento contiene, appunto, informazioni 'speciali' destinate ad istruire i programmi di stampa o di visualizzazione di eseguire i disegni. Sia TikZ sia pict2e, usati da Molinaro e da Beccari, non fanno altro che trasformare le macro del linguaggio TEX o LATEX in istruzioni per quei programmi di visualizzazione e di stampa. Di per sé il formato DVI originale di Knuth o quello esteso usato da XeLATEX affidano la loro visualizzazione ad altri programmi; per semplicità diciamo che il programma `dvipdfmx` traduce quei dati speciali in istruzioni del linguaggio PDF nel momento in cui il formato DVI viene trasformato in quello PDF, mentre i programmi `pdflatex` e `lualatex` producono direttamente il formato PDF.

I pacchetti TikZ e `pict2e`, quindi, non sono altro che delle interfacce fra l'utente e il linguaggio PDF per trasformare le istruzioni di composizione, comprese quelle per il disegno, in istruzioni inserite nel file finale ma destinate al programma di visualizzazione.

5 La soluzione di Giacomelli

La soluzione di Giacomelli, che viene descritta in questo paragrafo, sfrutta il linguaggio Lua per inserire direttamente le istruzioni di disegno nel file PDF prodotto da `lualatex` senza ricorrere a nessun pacchetto di interfaccia. In realtà la soluzione di Giacomelli prima crea poi usa una libreria in linguaggio Lua che estende quanto è già contenuto dentro il compilatore `luatex`.

Il lettore che voglia approfondire il linguaggio Lua ha a disposizione il testo originale del suo creatore (Ierusalimschy, 2016) e il manuale di istruzioni del compilatore `luatex`, (The LuaTEX development team, 2017).

5.1 Lo scopo del codice

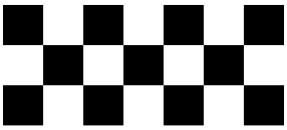
Lo scopo del codice di Giacomelli è quello di creare un comando `\scacchieraRG` che accetti due argomenti con la sintassi seguente:

`\scacchieraRG{`⟨*colonne*⟩`}{`⟨*righe*⟩`}`

che possa essere usato per disegnare una scacchiera conforme ai requisiti del concorso, salvo il fatto che è in grado di disegnare una scacchiera rettangolare con numeri di ⟨*righe*⟩ e ⟨*colonne*⟩ specificate "a piacere"; per esempio il codice[3]:

```
\scacchieraRG{7}{3}
```

e produce la figura:



Quello che Giacomelli si propone di fare è scrivere direttamente nel file di uscita le istruzioni in linguaggio PDF attraverso il linguaggio Lua.

3. Si ricorda che in questo articolo i nomi delle macro sono stati estesi con le iniziali dei rispettivi autori proprio per non generare conflitti.

5.2 I comandi primitivi del linguaggio PDF

Per conoscere questi comandi primitivi del linguaggio PDF, il protocollo pubblico della Adobe è veramente troppo lungo e complesso, ma il documento Hàn Thế Thành *et al.* (2017) contiene il necessario. Qui si ricorda solo quali siano i comandi nativi di PDF per disegnare un rettangolo:

$\langle x \rangle$ $\langle y \rangle$ $\langle w \rangle$ $\langle h \rangle$ `re`

dove $\langle x \rangle$ e $\langle y \rangle$ rappresentano le coordinate del vertice inferiore sinistro; $\langle w \rangle$ e $\langle h \rangle$ la larghezza e l'altezza.

Per non confondere le istruzioni di disegno è bene che ogni oggetto sia racchiuso dentro un stack insieme allo spessore delle linee del disegno e al modo di congiungersi delle spezzate che compongono l'eventuale contorno. Quindi al codice di base indicato sopra bisogna aggiungere qualche altro comando; in particolare tutto va incluso nell'argomento del comando `\pdfliteral`, tanto che il codice seguente

```
\pdfliteral{
q
0   0 32 32 re
32 32 32 32 re
f
S
Q
}
```

inserito all'interno di un documento `.tex`, permette di disegnare due rettangoli (quadrati) neri, uno con il vertice nella coordinata `(0,0)` e l'atro con il vertice nella coordinata `32,32`, entrambi aventi base e altezza uguali a 32 unità. I codici `q` e `Q` servono per aprire e chiudere uno stack (rispettivamente gli equivalenti di `push` e `pop`) e gli altri per le altre caratteristiche del disegno: `f` per `fill` e `S` per `stroke`. Come si vede il codice è compattissimo, anche se sostanzialmente si tratta delle stesse cose, scritte in modo meno prolisso, che si possono scrivere in PostScript.

Queste unità, sia ben chiaro, devono essere i "big point", i punti PostScript, con la sigla TEX `bp`, raramente usati direttamente dagli utenti, ma richiesti espressamente dal linguaggio PDF[4].

Quello che si ottiene con quel codice è semplicemente una piccola scacchiera di 2 righe e due colonne:

4. Anche TEX di default usa i punti tipografici e non i punti PostScript; la differenza è minima ma c'è: infatti 1 pt = (1/72,27) in ≈ 0,35146 mm mentre 1 bp = (1/72) in ≈ 0,35278 mm con una differenza di un poco meno del 0,4%, piccola ma non trascurabile. Ne consegue che quei 32 bp sono circa 10 mm.

Tuttavia il codice mostrato sopra non à ancora completo, perché non occupa lo spazio corrispondente alle sue dimensioni effettive. Vedremo fra poco come fare.

5.3 La funzione Lua `build_pdfliteral()`

Ora entra in scena il linguaggio Lua; la funzione Lua `build_pdfliteral()` ha il compito di generare il codice che descrive tutti i quadrati della scacchiera. Perciò dovrà ricevere in ingresso il numero di caselle in righe e colonne.

La soluzione ideale per produrre la stringa di descrizione `pdfliteral` è usare una tabella Lua come buffer e poi concatenare gli elementi con la funzione di libreria `table.concat()`. Dovremo inoltre rispettare il vincolo "casella in basso a sinistra sempre nera".

Per considerare l'alternanza di caselle nere e bianche è possibile far uso di una variabile booleana nominata per esempio `is_black`:

```
-- w : numero di caselle in orizzontale
-- h : numero di caselle in verticale
-- hs: misura in big point della singola casella
local function build_pdfliteral(w, h, hs)
  local y = 0
  local is_odd = true
  local fmt = string.format
  local fdim = fmt(''%d %d re'', hs, hs)
  local fret = ''%d %d '' .. fdim

  local t = {}
  t[#t + 1] = ''q''

  for _ = 1, h do
    local is_black = is_odd
    local x = 0
    for _ = 1, w do
      if is_black then
          t[#t + 1] = fmt(fret, x, y)
          is_black = false
      else
          is_black = true
      end
      x = x + hs
    end
    y = y + hs
    is_odd = not is_odd
  end

  t[#t + 1] = ''f''
  t[#t + 1] = ''S''
  t[#t + 1] = ''Q''
  return table.concat(t, ''\n'')
end
```

Il cuore della funzione è basato su due cicli `for` annidati. Per i più curiosi, si è utilizzata un'espressione idiomatica di Lua per riempire in sequenza l'array `t`, usando l'operatore `#` che restituisce la dimensione dell'oggetto. Così l'intero `#t + 1` è l'indice della prossima posizione libera nell'array.

Altra nota interessante da evidenziare sul linguaggio Lua riguarda l'assegnazione della funzione `string.format()` alla variabile `fmt` in una delle

prime righe del codice. Non si tratta solamente della comodità dell'abbreviazione del nome, ma dell'efficienza di esecuzione del codice perché l'accesso alle variabili locali è più veloce di quello alle variabili globali. La cosa può sorprendere chi non conosca questo linguaggio: in Lua le funzioni sono oggetti di prima classe, cioè sono valori assegnabili alle variabili come qualsiasi altro tipo.

Eseguendo la funzione in uno script Lua per riprodurre il codice iniziale, potremo verificare che effettivamente essa produce il risultato corretto. Per esempio la chiamata:

```
print(build_pdfliteral(2, 2, 32))
```

fornisce il codice pdfliteral dell'esempio precedente (che qui si ripete):

```
q
0 0 32 32 re
32 32 32 32 re
f
S
Q
```

Tuttavia, invece di verificare ogni volta il flag `is_black`, si potrebbe rendere un po' più efficiente la funzione considerando che su ciascuna riga della scacchiera è sufficiente raddoppiare l'incremento della coordinata $\langle x \rangle$ e disegnare sempre la casella come nera:

```
local function build_pdfliteral(w, h, hs)
    local y = 0
    local is_odd = true
    local fmt = string.format
    local fdim = fmt(''%d %d re'', hs, hs)
    local fret = ''%d %d '' .. fdim

    local t = {}
    t[#t + 1] = ''q''

    for _ = 1, h do
        local start = is_odd and 1 or 2
        local x = is_odd and 0 or hs
        for _ = start, w, 2 do
            t[#t + 1] = fmt(fret, x, y)
            x = x + 2*hs
        end
        y = y + hs
        is_odd = not is_odd
    end

    t[#t + 1] = ''f''
    t[#t + 1] = ''S''
    t[#t + 1] = ''Q''
    return table.concat(t, ''\n'')
end
```

Ma c'è una terza soluzione: invece di pensare alla scacchiera come a un insieme semplice di quadrati, essa può essere considerata come la sovrapposizione di due griglie regolari traslate una rispetto all'altra del vettore corrispondente alla diagonale di una casella. In questo modo il codice diviene più semplice e comprensibile.

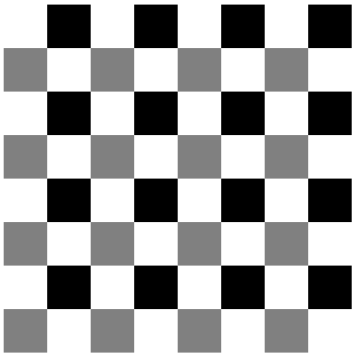


Figura 5: Schema della scacchiera a otto caselle che dimostra come la si possa disegnare sovrapponendo due diverse griglie regolari di quadrati, quella grigia e quella nera. Questa soluzione è quella utilizzata per implementare il comando `\scacchieraRG`.

Si definisca allora la funzione ausiliaria `grid()` che scriva in un buffer il codice grafico dei quadrati di una griglia regolare di passo doppio della dimensione delle caselle, a partire dalla posizione 1 oppure 2. Se si indica per `start_pos` il valore 1 si intende disegnare la griglia la cui casella in basso a sinistra è nella posizione `(1, 1)` , mentre con il valore 2 si intende la seconda griglia sovrapposta che inizia dalla casella nella posizione `(2, 2)`:

```
local function grid(buf, hs, w, h, start_pos)
    assert(start_pos == 1 or start_pos == 2)
    local fmt = string.format
    local fdim = fmt(''%d %d re'', hs, hs)
    local fret = ''%d %d '' .. fdim

    local delta = 2 * hs
    local y = start_pos == 1 and 0 or hs
    local x_0 = y
    for _ = start_pos, h, 2 do
        local x = x_0
        for _ = start_pos, w, 2 do
            buf[#buf + 1] = fmt(fret, x, y)
            x = x + delta
        end
        y = y + delta
    end
end
```

Nel listato, per calcolare il valore iniziale dell'ordinata $\langle y \rangle$ si è fatto ricorso all'equivalente Lua dell'operatore ternario del C++. Con questa funzione, quella principale si riduce alla seguente:

```
local function build_pdfliteral(w, h, hs)
    local t = {}
    t[#t + 1] = ''q''
    grid(t, hs, w, h, 1)
    grid(t, hs, w, h, 2)
    t[#t + 1] = ''f''
    t[#t + 1] = ''S''
    t[#t + 1] = ''Q''
    return table.concat(t, ''\n'')
end
```

Per la scacchiera 8×8 della figura 5, i quadrati grigi sono disegnati dalla prima chiamata a `grid()`, mentre con la seconda sono disegnati quelli colorati in nero.

5.4 La funzione `create_box()`

Con la seconda funzione si passa da Lua a LuaTEX. Useremo le librerie interne `node` per creare il nodo `pdfliteral`, e `tex` per inserire il nodo in un registro di tipo box. Non occorrono particolari aggiustamenti di *bounding box* perché i limiti inferiore e sinistro del disegno coincidono con gli assi coordinati:

```
local function create_box(boxname, w, h)
  -- fixed house dimension
  local house_size = 16
  -- pdfliteral node
  local n = node.new(
    ''whatsit'', ''pdf_literal''
  )
  n.data=build_pdfliteral(w, h, house_size)
  assert(
    tex.isbox(boxname),
    string.format(
     ''Box register [%s] doesn't exist'',
     boxname
    )
  )
  -- horizontal box
  local hbox = node.hpack(n)
  local dim_sp = tex.sp(
      tostring(house_size)..''bp''
  )
  hbox.width  = w * dim_sp
  hbox.height = h * dim_sp
  tex.box[boxname] = hbox
end
```

Le funzioni Lua possono essere inserite in un file esterno con estensione `.lua`. In questo modo non dovremo gestire la sovrapposizione dei diversi significati assegnati da TEX e da Lua ad alcuni caratteri speciali come il simbolo del percento. In Lua esso è l'operatore modulo, oppure la definizione delle classi di formato; in TEX è il simbolo di commento.

Il modo più semplice per creare un modulo di libreria in Lua è creare una tabella contenitore delle funzioni, restituendone il riferimento con un'istruzione finale `return`. In realtà solamente la funzione `create_box()` può essere considerata come pubblica, mentre la `build_pdfliteral()` e la sua ausiliaria `grid()` possono essere considerate come funzioni private non accessibili dall'utente.

Per fare questo definiremo come locali le ultime due, contando sul meccanismo della *closure*, perché continuino a essere accessibili internamente, mentre per la prima useremo la speciale sintassi seguente per memorizzare direttamente in un campo della tabella la funzione pubblica:

function ⟨*table_name*⟩.⟨*func_name*⟩(⟨*argomenti*⟩)
– ⟨*body*⟩
end

Il codice completo è riportato nella pagina 40.

5.5 Il risultato finale

Ora tutto è disponibile per disegnare la scacchiera. Si scrive il sorgente LuaLATEX che carica le funzioni dal file esterno `libsk.lua` posizionato nella stessa directory[5] e definisce la nuova macro `\scacchiera`. Quest'ultima passa subito il controllo a Lua che riceve in ingresso i valori delle dimensioni della scacchiera da disegnare assegnandoli a due variabili locali, per effetto dell'espansione.

Dopo viene chiamata la funzione `create_box()`. Al termine di questo passaggio nel registro `\libsk` ci sarà il disegno come se lo avessimo creato con la primitiva `\pdfliteral`. Perciò non ci rimane che utilizzare il registro tramite l'istruzione TEX `\box`.

Nel listato qui di seguito si è riportato il codice, ridondante per gli scopi principali, ma utile per mostrare al lettore come appare il preambolo di un tipico documento LuaLATEX. Rispetto a pdfLATEX non si carica più il pacchetto inputenc perché il sorgente deve essere codificato UTF-8, mentre si usa fontspec per l'impostazione dei font – qui, a titolo di esempio, si utilizzano il font Open Type Libertinus – e si sostituisce babel con polyglossia.

Ultima osservazione, si è inserito prima e dopo il disegno una lettera 'A' per verificare che la costruzione della scatola sia corretta e corrisponda alle dimensioni del disegno. Il risultato è mostrato nella figura 7.

```
% !TeX program = LuaLaTeX
\documentclass{article}

% font setup
\usepackage{fontspec}
\defaultfontfeatures{Ligatures=TeX}
\setmainfont{Libertinus Serif}
\setmonofont{Libertinus Mono}

% language setup
\usepackage{polyglossia}
\setmainlanguage[babelshorthands]{italian}

% si carica la libreria libsk.lua
\directlua{
    libsk = require ''libsk''
}
% si definisce il comando \scacchiera
\newbox\libskbox
\newcommand\scacchiera[2]{%
\directlua{
```

5. Usare `libsk.lua` per comporre documenti differenti con i relativi file conservati in cartelle differenti rende necessario copiarlo in altrettante cartelle. Si può evitare questa cosa in due modi: salvandolo in una cartella del proprio albero personale oppure inserendo nelle varie cartelle solo un link simbolico all'unica copia del file anzidetto. I link simbolici sono da sempre disponibili nei sistemi operativi Linux e Mac, e lo sono anche con i sistemi Windows da Win 8 in poi.

```
local libsk = {}

local function grid(buffer, house_size, w, h, start_pos)
    assert(start_pos == 1 or start_pos == 2)
    local fmt = string.format
    local fdim = fmt(''%d %d re'', house_size, house_size)
    local fret = ''%d %d '' .. fdim
    local y = start_pos == 1 and 0 or house_size
    local x_0 = y
    local delta = 2 * house_size
    for _ = start_pos, h, 2 do
        local x = x_0
        for _ = start_pos, w, 2 do
            buffer[#buffer + 1] = fmt(fret, x, y)
            x = x + delta
        end
        y = y + delta
    end
end

local function build_pdfliteral(w, h, house_size)
    local t = {}
    t[#t + 1] = ''q''
    grid(t, house_size, w, h, 1)
    grid(t, house_size, w, h, 2)
    t[#t + 1] = ''f''
    t[#t + 1] = ''S''
    t[#t + 1] = ''Q''
    return table.concat(t, ''\n'')
end

function libsk.create_box(boxname, w, h)
        -- fixed house dimension
        local house_size = 16
    -- pdfliteral node
    local n = node.new(
        ''whatsit'', ''pdf_literal''
    )
    n.data = build_pdfliteral(w, h, house_size)

    assert(
        tex.isbox(boxname),
        string.format(
           ''Box register [%s] doesn't exist'',
           boxname
        )
    )

    -- horizontal box
    local hbox = node.hpack(n)
    local dim_sp = tex.sp(tostring(house_size)..''bp'')
    hbox.width  = w * dim_sp
    hbox.height = h * dim_sp
    tex.box[boxname] = hbox
end

return libsk
```

Codice Lua 6: Codice da salvare in un file luask.lua che deve essere conservato in una cartella come viene spiegato nel testo.

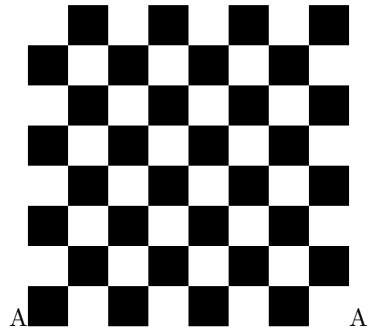


FIGURA 7: Il risultato finale della scacchiera creata con LuaTEX. Le lettere 'A' prima e dopo il disegno mostrano il corretto dimensionamento della scatola che contiene il disegno.

```
    local w = tonumber(#1)
    local h = tonumber(#2)
    libsk.create_box([[libskbox]], w, h)
}%
\leavevmode
\box\libskbox
}

\begin{document}
A\scacchieraRG{8}{8}A
\end{document}
```

5.6 Facile o difficile?

Osservando il codice Lua del file `libsk.lua` si potrebbe pensare che sia *difficile* scriverlo e quindi verrebbe voglia di rinunciarvi. Secondo l'opinione di Giacomelli, Lua è in realtà molto più semplice di TEX e la soluzione descritta non fa uso di nessun pacchetto esterno, ma tutto il lavoro è fatto internamente a LuaLATEX.

6 Confronti e commenti finali

Alla fine di questa carrellata sulle tre soluzioni mostrate, molto diverse l'una dall'altra, si possono fare alcuni confronti.

Non c'è dubbio che la più semplice sia quella di Molinaro, la più versatile sia quella di Beccari e la più professionale quella di Giacomelli.

I codici sono molto diversi: quelli di Molinaro e di Giacomelli attuano con i rispettivi strumenti un metodo originale per comporre la scacchiera sovrapponendo due scacchiere sfalsate in diagonale di una casella contenenti caselle di colore scuro con passo 2, ma i metodi seguiti sono diversi; entrambi in un certo senso dimensionano queste due scacchiere in modo che siano basate su dimensioni pari a quelle specificate, modulo 2. Siccome gli strumenti disponibili sono diversi, il modo di attuare in pratica questa operazione modulo 2 è molto diversa.

La soluzione di Beccari usa invece una sola scacchiera, ma usa un algoritmo originale per decidere se in una data casella si debba inserire una casella scura; tuttavia, siccome Beccari pensa anche alla creazione di cruciverba, sembra abbastanza ovvio che non si sia preoccupato di escogitare un "trucco" così efficace come quello delle due scacchiere sovrapposte e sfalsate.

La versatilità si vede anche confrontando gli argomenti dei comandi o degli ambienti che creano le rispettive scacchiere. Abbiamo le seguenti differenze:

Molinaro

```
{⟨righe⟩}{⟨dimensione casella⟩}
```

Beccari

1. La specificazione più semplice:

   ```
   {⟨righe⟩}
   ```

2. La specificazione più complessa:

   ```
   ⟨*⟩[⟨larghezza scacchiera⟩]%
     {⟨righe⟩,⟨colonne⟩}{⟨colori⟩}%
       [⟨offset⟩]
   ```

Giacomelli

```
{⟨colonne⟩}{⟨righe⟩}
```

È altrettanto ovvio che se le soluzioni di Molinaro e di Giacomelli avessero la stessa versatilità di quella di Beccari, i loro codici dovrebbero allungarsi moltissimo, specialmente se gli autori volessero rimanere completamente dentro la loro impostazione di base, TikZ per Molinaro e `libsk.lua` per Giacomelli; ma i confronti non si devono fare su questo livello, dato che il concorso non lo prevedeva. Non prevedeva nemmeno che l'intera scacchiera avesse un contorno evidente, ma Beccari e Molinaro hanno provveduto in merito, mentre Gacomelli non l'ha previsto.

Invece il lettore ha modo di imparare diversi approcci e diversi modi di affrontare lo stesso problema; l'analisi critica di questi approcci e dei diversi modi esposti permette di usare la strategia più opportuna quando il lettore dovesse affrontare problemi la cui soluzione non sia così ovvia come potrebbe sembrare a prima vista.

7 Epilogo

I tre autori hanno livelli di dimestichezza con LATEX molto diversa; Maurizio Molinaro è socio del GUIT dal 2014; il suo contributo più importante e visibile alla comunità degli utenti è il bell'articolo (MOLINARO, 2017) sulla tipografia relativa al gioco degli scacchi. In un certo senso ci si sarebbe stupiti se l'autore non avesse presentato una sua soluzione al problema del concorso.

Claudio Beccari lavora con LATEX da più di 30 anni; ha una grandissima esperienza alle spalle, ma è molto affezionato alle cose tradizionali, una delle quali è l'ambiente picture sul quale ha

investito molte energie, non solo collaborando allo sviluppo di pict2e ma anche costruendo per suo uso e consumo grossi pacchetti basati su pict2e per estenderne le funzionalità al disegno di schemi particolari, come i circuiti elettronici e i disegni di diagrammi e istogrammi.

Roberto Giacomelli lavora con LaTeX da una dozzina di anni e lo usa abitualmente per la sua attività professionale; recentemente ha pubblicato diversi articoli su ArsTEXnica dove mostra le funzionalità di cui dispone LuaLaTeX per venire incontro a moltissimi problemi che si presentano nella documentazione relativa all'esercizio della sua professione.

Riferimenti bibliografici

Beccari, C. (2017). «Font e tipografia – pdfLaTeX, XeLaTeX, LuaLaTeXe i font». PDF document. Scaricabile da http://www.guitex.org/home/images/doc/GuideGuIT/guidafont.pdf.

Fischer, U. (2014). *chessboard: A package to print chessboards*. Versione 1.7; documento PDF leggibile con `texdoc chessboard`.

Gässlein, H., Niepraschk, R. e Tkadlec, J. (2016). «The pict2e package». PDF document. Leggibile con `texdoc pict2e`.

Hàn Thế Thành, Rahtz, S., Hagen, H., Henkel, H., Jackowski, P. e Schröder, M. (2017). «The pdfTeX user manual». PDF document. Leggibile con `texdoc pdftex-a`.

Ierusalimschy, R. (2016). *Programming in Lua*. Lua.org, 3ª edizione.

Kern, U. (2016). «Extending LaTeX's color facilities: the xcolor package». PDF document. Leggibile con `texdoc xcolor`.

Lamport, L. (1994). *LaTeX. A Document Preparation System*. Addison-Wesley, 2ª edizione.

Lehman, P. e Wright, J. (2017). «The etoolbox package». PDF document. Leggibile con `texdoc etoolbox`.

Molinaro, M. (2017). «Introduzione alla composizione di testi scacchistici». *ArsTEXnica*, (25).

The LaTeX3 project (2017). «The xparse package». PDF document. Leggibile con `texdoc xparse`.

The LuaTeX development team (2017). *LuaTeX Reference Manual*, 1.0.4 edizione.

The $\mathcal{NTS}$ Team (1998). «The ε-TeX manual». PDF document. Leggibile con `texdoc etex`.

Thorup, K. K., Jensen, F. e Rowley, C. (2014). «The calc package». PDF document. Leggibile con `texdoc calc`.

▷ Claudio Beccari
`claudio dot beccari at gmail dot com`

▷ Roberto Giacomelli
`giaconet dot mailbox at gmail dot com`

▷ Maurizio Molinaro
`maurizio dot molinaro3 at gmail dot com`