Numero 15
Aprile 2013

ArsTEXnica

Rivista italiana di TEX e LATEX

GUIT

http://www.guitex.org/arstexnica

GUIT – Gruppo Utilizzatori Italiani di TEX

ArsTEXnica è la pubblicazione ufficiale del GUIT

ArsTEXnica è la prima rivista italiana dedicata a TEX, a LATEX ed alla tipografia digitale. Lo scopo che la rivista si prefigge è quello di diventare uno dei principali canali italiani di diffusione di informazioni e conoscenze sul programma ideato quasi trent'anni fa da Donald Knuth.

Le uscite avranno, almeno inizialmente, cadenza semestrale e verranno pubblicate nei mesi di Aprile e Ottobre. In particolare, la seconda uscita dell'anno conterrà gli Atti del Convegno Annuale del GUIT, che si tiene in quel periodo.

La rivista è aperta al contributo di tutti coloro che vogliano partecipare con un proprio articolo. Questo dovrà essere inviato alla redazione di ArsTEXnica, per essere sottoposto alla valutazione di recensori. È necessario che gli autori utilizzino la classe di documento ufficiale della rivista; l'autore troverà raccomandazioni e istruzioni più dettagliate all'interno del file di esempio (`.tex`). Tutto il materiale è reperibile all'indirizzo web della rivista.

Gli articoli potranno trattare di qualsiasi argomento inerente al mondo di TEX e LATEX e non dovranno necessariamente essere indirizzati ad un pubblico esperto. In particolare tutorials, rassegne e analisi comparate di pacchetti di uso comune, studi di applicazioni reali, saranno bene accetti, così come articoli riguardanti l'interazione con altre tecnologie correlate.

Di volta in volta verrà fissato, e reso pubblico sulla pagina web, un termine di scadenza per la presentazione degli articoli da pubblicare nel numero in preparazione della rivista. Tuttavia gli articoli potranno essere inviati in qualsiasi momento e troveranno collocazione, eventualmente, nei numeri seguenti.

Chiunque, poi, volesse collaborare con la rivista a qualsiasi titolo (recensore, revisore di bozze, grafico, etc.) può contattare la redazione all'indirizzo:

`arstexnica@sssup.it`.

Nota sul Copyright

Associarsi a GUIT

Fornire il tuo contributo a quest'iniziativa come membro, e non solo come semplice utente, è un presupposto fondamentale per aiutare la diffusione di TEX e LATEX anche nel nostro paese. L'adesione al Gruppo prevede una quota di iscrizione annuale diversificata: 30,00 € soci ordinari, 20,00 (12,00) € studenti (junior), 75,00 € Enti e Istituzioni.

Indirizzi

Gruppo Utilizzatori Italiani di TEX:
c/o Ufficio Statistica
Scuola Superiore Sant'Anna
Piazza Martiri della Libertà 33
56127 Pisa, Italia.
`http://www.guitex.org`
`guit@sssup.it`

Redazione ArsTEXnica:
`http://www.guitex.org/arstexnica/`
`arstexnica@sssup.it`

Codice ISSN 1828-2369

Stampata in Italia
Pisa: 15 Aprile 2013

ArsTEXnica

Rivista italiana di TEX e LATEX

Numero 15, Aprile 2013

Gruppo Utilizzatori Italiani di TEX

Editoriale

Gianluca Pignalberi

Puff... pant... *Oppòrcamiseria!* Sono di nuovo in ritardo!

Come ormai consueto, anche questo mio ultimo appuntamento col numero dispari di ArsTEXnica esce con un trimestre di ritardo. Spero che il mio ritardo sia compensato, anche solo in parte, dal merito degli autori di averci fornito i lavori di cui vi parlerò tra breve.

Prima, però, dobbiamo parlare del prossimo GUIT*meeting*, dall'anno scorso itinerante. Quest'anno, dopo diverse richieste, numerose ipotesi e svariati tentativi, approderemo a Roma. Roma mi è cara per diversi motivi: vicinanza geografica, frequentazione dell'università, collaborazioni lavorative. Quando il nostro socio Michele Scarpiniti mi ha comunicato che avrebbe chiesto il permesso di ospitare il nostro meeting alla segreteria della sua facoltà, e ancor più quando mi ha detto di averlo ottenuto, *capirai, a mme me s'è 'llargato 'r còre* (Mario Brega nel ruolo di A(u)gusto, Borotalco, 1981). Dunque è ufficiale: il GUIT*meeting2013* sarà ospitato nella sala del chiostro della facoltà di Ingegneria dell'università Sapienza, in via Eudossiana 18, a due passi dal Colosseo (per l'occasione ribattezzato AnfiTEXatro Flavio). Correte alla terza di copertina per leggere il call for papers.

La data del meeting ha suscitato altre discordanze: per il secondo anno lo teniamo in concomitanza del Linux Day (e, per non farci mancare niente, anche in concomitanza dell'ultimo giorno del TUG meeting). Molti utenti di LATEX sono anche utenti di Linux e, per cause diverse, hanno maggior necessità di partecipare al Linux Day che non al GUIT*meeting*. Mi rendo conto sia delle priorità che delle necessità. In mancanza di una discussione di gruppo ho preferito proseguire con la linea dell'ultimo sabato di ottobre, tantopiù che una discussione interna al Consiglio Direttivo ha indicato la data scelta come comunque buona. Sarà opportuno che la discussione si allarghi a una votazione dell'intera associzione così che l'indicazione sia la più condivisa possibile, ma questo sarà compito, verosimilmente, del nuovo Consiglio Direttivo.

Superato il meeting, infatti, decadranno le cariche del Consiglio Direttivo e anche quella di direttore di ArsTEXnica. Continuo ad aspettare e sperare che una sfilza di persone proponga la propria candidatura. Ho ricevuto qualche timido riscontro, ma servono dei "sì" chiari e netti. Lasciatemi abusare di questo spazio con delle considerazioni personali e riguardanti la mia persona. Quando sei anni fa ricevetti la proposta di candidarmi a direttore di ArsTEXnica, fui lusingato e spaventato al contempo. Ho cercato di svolgere il mio incarico senza far rimpiangere Massimiliano Dominici. Quando alla scadenza del mio mandato nessuno si fece avanti per rimpiazzarmi accettai di prolungare il mio incarico. Purtroppo dovetti anche accettare di candidarmi a una posizione preminente nel Consiglio Direttivo perché non c'erano soci che per statuto potessero farlo (tra quelli che accettavano di candidarsi). Sebbene non ci sia modo di "intrallazzare" perché il nostro tesoriere è di un rigore e di una precisione ultraterreni e non permetterebbe neanche l'ipotesi di "magheggi", la permanenza della stessa persona nella stessa carica per anni e anni (per esempio, il presidente e il tesoriere nell'AVIS del mio comune) mi ha sempre fatto storcere il naso (le suddette persone vedono, curiosamente, la candidatura di altri come un non riconoscimento al loro lavoro. Questo stile mi ricorda i varî Kim Il Sung, Kim Jong Il, Kim Jong Un... Avremo un'AVIS dinastica?). Dunque, non fate in modo che io debba ricandidarmi perché

1. non ne ho la benché minima intenzione;
2. ho già ricoperto troppe cariche per i miei gusti (al più potrei accettare di rimanere in Consiglio Direttivo come semplice consigliere).

Ora possiamo parlare dei contenuti. Apriamo con due articoli di Claudio Fiandrino, entrambi dedicati a Ti*k*Z. Nel primo l'autore espone diversi metodi per disegnare i diagrammi Entità-Relazione. Tali metodi spaziano da quello fornito dalla libreria standard `er` all'approccio orientato agli oggetti fornito dalla libreria `er-oo` passando per il pacchetto `Ti`*`k`*`Z-er2` e per il programma GraphViz. Nel secondo articolo l'autore descrive la libreria `Sa-Ti`*`k`*`Z` e i suoi strumenti per disegnare delle architetture di rete. Il successivo articolo, a cura del sottoscritto, vi intrattiene descrivendo come ho convinto LATEX a riprodurre un layout non banale: parte della prima pagina de La Settimana Enigmistica. Nell'articolo propongo un metodo semiautomatico per costruire schemi di parole incrociate, schemi decisamente diversi da quelli per le *crosswords* permessi da alcuni pacchetti disponibili su CTAN. Massimiliano Dominici, autore del quarto articolo, mostra come l'uso di Pandoc possa aiutare la generazione di documenti LATEX e HTML a partire dallo stesso sorgente trattato col markup "leggero" offerto dal linguaggio Markdown.

Proseguiamo con due articoli di Claudio Beccari. Il primo, seconda puntata sull'opera filologica

di recupero dei dialetti, ci parla di LaTeX e del friulano. Non c'è altro da dire se non che da anni l'autore è impegnato nel dotare LaTeX di pattern di sillabazione e dizionari ed è un compito che gli riesce sempre benissimo. In questo caso l'articolo è dotato di *somari*. Il secondo articolo pone, e risolve, la sfida di inserire le filigrane nei documenti senza usare i pacchetti dedicati. L'articolo è stato scritto a partire da un post nel forum e usato come pretesto per una piccola gara lanciata dall'autore nello stesso forum, gara il cui resoconto leggerete proprio su queste pagine.

In chiusura troviamo l'articolo di Luigi Scarso dedicato a LuajitTeX, cioè la variante di LuaTeX che usa un compilatore just-in-time per Lua. Di Scarso sappiamo molto, ad esempio che è attivissimo nella comunità ConTeXt, che si occupa di molti argomenti con eclettismo invidiabile (ne trovate parte sui vecchi numeri di ArsTEXnica) e che è membro del Consiglio Direttivo. Quello che non tutti sanno, però, è che il suo lavoro relativo a Swiglib e al supporto dinamico di librerie in LuaTeX è stato giudicato così importante dal TUG da avergli garantito un *grant* (una sovvenzione, o borsa di studio). Ottimo risultato per un ottimo lavoro.

Ora vi lascio alla lettura, alla meditazione e alla preparazione al meeting. Happy TeXing.

▷ Gianluca Pignalberi
`g dot pignalberi at alice dot it`

Drawing ER diagrams with Ti*k*Z

Claudio Fiandrino

Abstract

The paper will illustrate some techniques to represent Entity-Relationship (ER) diagrams with Ti*k*Z. In particular, it will focus on the standard internal library `er`, on the external package Ti*k*Z-er2, on the external tool Graphviz and on the object-oriented approach provided by the `er-oo` library.

Sommario

L'articolo illustrerà alcune tecniche per rappresentare i diagrammi Entità-Relazione (ER) con Ti*k*Z. In particolare si concentrerà sulla libreria standard interna `er`, sul pacchetto esterno Ti*k*Z-er2, sul programma esterno Graphviz e sull'approccio orientato agli oggetti fornito dalla libreria `er-oo`.

1 Introduction

The Entity-Relationship (ER from now on) model is a common way to model and represent databases. Peter Chen proposed the model specification in (Chen, 1976). The model is built using three main blocks:

- entities;
- relationships;
- attributes.

Entities are real-world items or concepts that can exist independent of one another and are uniquely identified. Examples of *physical* entities are "computer" or "car", while *concept* entities are "customer order" or "payment". Their standard notation is a rectangle. The entity "student" is represented in figure 1.

Figure 1: Entity graphical representation

Entities are linked by relationships, which describe the relation the entities share. Such relations can be classified according to the so-called degree of the relationship, an index of relevance that represents how many entities the relationship involves. Relationships are represented with diamonds as shown in figure 2, where a "student" and a "professor" are linked by a *thesis*. According to common diagram conventions, a relation in which entities participate more than once is called *total* or *surjective* or *recursive* and the visual link is double line. The relation "supervision" in a "team", shown in figure 3, is *total* because some team members will be supervisors, others will be supervisee.

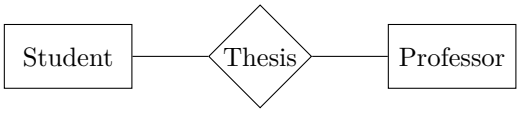


Figure 2: Relationship graphical representation

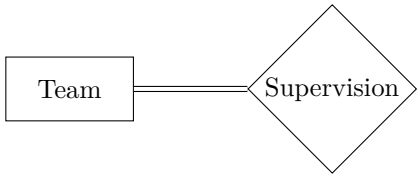


Figure 3: Total relation graphical representation

At last, both entities and relationships can have attributes to describe particular properties. Notice that attributes themselves can have attributes and are called *composite attributes*; for example, the "address" attribute for a "person" entity could be described by "street" and "city", two attributes in this case. Attributes are represented with ovals which border changes according to the type of the attribute:

solid border for *simple* attributes;

dashed border for *derived* attributes. Attributes are derived when we infer them from entities or relationships (e.g., the "age" attribute could derive from the entity "person");

double border for *multi*-attributes. Likewise total relationships, a multi-attribute has more than one value per entity or relationship (e.g., the attribute "phone" for the entity "person").

Figure 4 shows every possible attributes in a single example.

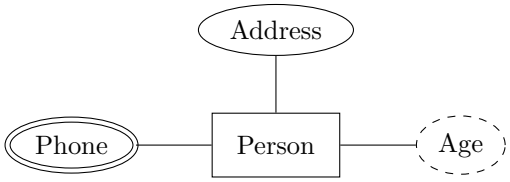


Figure 4: Attributes graphical representation

While this short introduction to ER models is not exhaustive, it provides the sufficient background to understand this paper. Further references are `http://wofford-ecs.org/`

dataandvisualization/ermodel/material.htm and http://users.informatik.uni-halle.de/~brass/db04/c3_ermod.pdf.

The remainder of the paper is organized as follows: section 2 will focus on the er library, section 3 will describe the TikZ-er2 package, section 4 will show how Graphviz works and section 5 will mention the er-oo module. Every sections will show the same example programmed (drawn) using the discussed tool. At last, section 6 will conclude the paper.

2 The library er

2.1 Usage

TikZ provides plenty of standard libraries, including er (Tantau, 2010, section 31 Entity-Relationship Diagram Drawing Library). As every TikZ libraries, the user has to load it in the preamble:

```
\usepackage{tikz}
\usetikzlibrary{er}
```

The library defines the keys necessary to represent standard entities, relationships and attributes. Those keys are:

- entity to represent the entity nodes;
- relationship to represent the relationship nodes;
- attribute to represent the attribute nodes;
- key attribute to represent a key attribute node;

and they are applied to TikZ nodes by the command

```
\node[key type] (label) at (position)
 {text};
```

The user can customize nodes style modifying the every entity, every relationship and every attribute keys. The ways to globally customize nodes of an ER diagram, according to TikZ habits, are:

- ```
  \tikzset{every entity/.style={
      ...
      customization
      ...
   },
  }
  ```

- ```
  \tikzstyle{every entity}=[
      ...
      customization
      ...
  ]
  ```

The latter way is discouraged in favor of the former.

The main advantage of using the er library is that users do not need to use external packages or tools: standard TeX Live or MiKTeX let them immediately operate. Unfortunately users are always requested to spatially organize the diagram and, optionally, to customize elements style. When the diagram is large, finding a good layout can be tricky and time-consuming.

2.2 A real example with er

We will now see how to draw a simple diagram composed by two entities, "person" and "tool", linked by the only relationship "uses". Notice that more than one person may use the same tool and a tool can use other tools. This diagram will be taken as reference to compare the code of all the techniques shown in the paper.

Listing 1 shows a complete minimal example which result appears in figure 5.

As described in subsection 2.1, the library only provides keys for basic elements, so it has been necessary setting up the multi attribute, derived attribute and total relationships styles. Notice that defining these style is very simple: it is just needed to use the already present attribute style along with the necessary customization. That is:

```
\tikzset{multi attribute/.style={
  attribute,
  double distance=1.5pt
  }
}
```

This particular procedure carries a very important advantage: the new attribute type will inherit its *father* style attribute. Indeed, it is perfectly possible to manually set up the multi attribute:

```
\tikzset{multi attribute/.style={
  ellipse,
  minimum size=1.5\baselineskip,
  draw,
  double distance=1.5pt,
  every multi attribute
  }
}
```

which prevents a later automatic style customization.

A specific command \key has been created to distinguish the key attribute. Indeed the er library does not provide any methods to underline a key attribute and just emphasizes it using italic.

Notice how all the elements have been manually positioned with keys above, below, left and right; they could have even been combined for finer results (e.g., above right). This explains why it is important that each node has its own *name*: in this way, it is possible to place it relatively to already defined nodes (e.g., above of=prevnode). The elements are 7em distant from each other thanks to the key node distance: it is the distance

```
\documentclass[a4paper,11pt,x11names]{article}

\usepackage{tikz}
\usetikzlibrary{er}
\tikzset{multi attribute/.style={attribute,double distance=1.5pt}}
\tikzset{derived attribute/.style={attribute,dashed}}
\tikzset{total/.style={double distance=1.5pt}}
\tikzset{every entity/.style={draw=orange, fill=orange!20}}
\tikzset{every attribute/.style={draw=MediumPurple1, fill=MediumPurple1!20}}
\tikzset{every relationship/.style={draw=Chartreuse2, fill=Chartreuse2!20}}
\newcommand{\key}[1]{\underline{#1}}

\begin{document}
\begin{tikzpicture}[node distance=7em]
\node[entity] (person) {Person};
\node[attribute] (pid) [left of=person] {\key{ID}} edge (person);
\node[attribute] (name) [above left of=person] {Name} edge (person);
\node[multi attribute] (phone) [above of=person] {Phone} edge (person);
\node[attribute] (address) [above right of=person] {Address} edge (person);
\node[attribute] (street) [above right of=address] {Street} edge (address);
\node[attribute] (city) [right of=address] {City} edge (address);
\node[derived attribute] (age) [right of=person] {Age} edge (person);
\node[relationship] (uses) [below of=person] {Uses} edge (person);
\node[entity] (tool) [below of=uses] {Tool} edge[total] (uses);
\node[attribute] (tid) [left of=tool] {\key{ID}} edge (tool);
\node[attribute] (tname) [right of =tool] {Name} edge (tool);
\end{tikzpicture}
\end{document}
```

Listing 1: Exploiting the er library

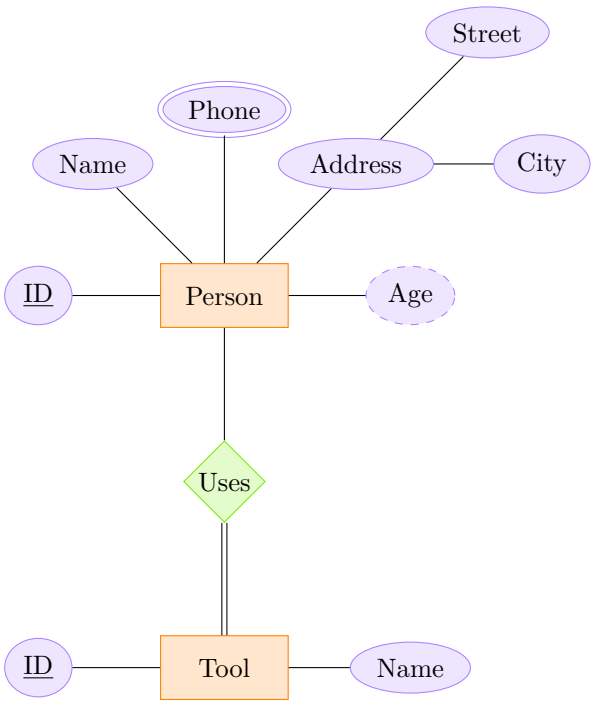


Figure 5: The reference ER diagram

between the anchor `center` of each pair of nodes. This distance applies to every node in tikzpicture, but it could be locally redefined in case one element should be shifted a bit; the right keys to use for that are `xshift` and `yshift`.

```
\node[multi attribute, xshift=1cm,
  yshift=1cm]
  (phone) [above of=person] {Phone}
  edge (person);
```

It is not important where we place the options, i.e., we can set them *before* or *after* the type of the node. However, it is important to highlight that the syntax

```
\node[options]
  (name) [position] {label}
  edge (destination);
```

makes possible to put and connect a node to an already existing `destination` node. Users usually write nodes in Ti*k*Z all together and then create the links using `draw` or `path`. That particular procedure allows to be fast, but the `destination` has to be already defined when a new node is attached to it. Thus:

```
\node[multi attribute] (phone)
  [above of=person] {Phone}
  edge (person);
\node[entity] (person) {Person};
```

will not work, but rather it will arise the following error:

```
! Package pgf Error:
No shape named person is known.
```

3 The package *TikZ-er2*

3.1 Usage

The *TikZ-er2* package (`https://www.assembla.com/wiki/show/tikz-er2`) provides a more detailed set of styles than the `er` library. Unfortunately, the package is not part of CTAN and thus does not come along with TEX Live or MiKTEX. The users wishing to use it have to install it by themselves.

A closer look at the package unveils its good structure: not only it provides the same styles as `er` does, it also has different types of attributes, entities and connections. Indeed it distinguishes among simple and total relationship with styles `link` and `total` styles respectively.

Just like `er`, the user is the only customizer of elements. She will do it in the same way she did with `er` because the *TikZ-er2* defines the same styles. These styles are in the form `every current-style`, thus customizing them could involve, again, `tikzset`. Positioning of elements is left to the user too. Ti*k*Z library `positioning` could be of help, but do not expect stumbling results: only GraphViz, introduced in section 4 can save time and some effort in placing the nodes.

3.2 A real example with *TikZ-er2*

TikZ-er2 allows to obtain the same result already shown in figure 5. The listing 2 shows that it is not necessary to write new styles and the users can only concentrate in customizing and placing elements.

4 GraphViz

4.1 Why GraphViz?

GraphViz is a graph-deployment program. Since an ER diagram *is* graph which vertices are diagram elements (entities, relationships and attributes) and edges are links between elements, it is straightforward to think of drawing an ER diagram with Graphviz. Readers can learn the basics about GraphViz-Ti*k*Z interaction in Fiandrino (2012). In this paper we will use dot2texi (Fauske, 2008) already present in TEX Live and MiKTEX as it simplifies the needed interaction.

GraphViz makes the user ignore how to place elements because it has specific algorithms to accomplish this task. They can be activated by options like circo or neato. However, elements styles are not foreseen and users has to design styles by themselves.

The compiler will compile the main file with the option -shell-escape because it has to translate the dot language with the underlying dot2tex application. For instance, let `main.tex` be the main file; the command to compile it is:

```
pdflatex -shell-escape main.tex
```

Of course, the user has to check for the presence of GraphViz and dot2tex in her system before using this approach. The paper Fiandrino (2012) describes the complete installation procedure for Ubuntu.

4.2 A real example with GraphViz

Listing 3 shows how to exploit GrapViz to draw the ER diagram and, at the same time, shows the major novelties of this approach.

As already mentioned, it is necessary to build all the styles describing every necessary elements. They are in the preamble and are always defined via `tikzset`. Notice that the styles `multi attribute` and `derived attribute` inherit from the already defined style `attribute` in this case too.

The major novelty and facility introduced by GraphViz is that each element is not placed in a given position, but its description is given by choosing the category it belongs to. We make the choice with the notation `style="<category>"` inside square brackets:

```
Person [style="entity"];
...
Phone [style="multi attribute"];
...
Uses [style="relationship"];
```

```
\documentclass[a4paper,11pt,x11names]{article}

\usepackage{tikz-er2}
\tikzset{every entity/.style={draw=orange, fill=orange!20}}
\tikzset{every attribute/.style={draw=MediumPurple1, fill=MediumPurple1!20}}
\tikzset{every relationship/.style={draw=Chartreuse2, fill=Chartreuse2!20}}

\begin{document}
\begin{tikzpicture}[node distance=7em]
\node[entity] (person) {Person};
\node[attribute] (pid) [left of=person] {\key{ID}} edge (person);
\node[attribute] (name) [above left of=person] {Name} edge (person);
\node[multi attribute] (phone) [above of=person] {Phone} edge (person);
\node[attribute] (address) [above right of=person] {Address} edge (person);
\node[attribute] (street) [above right of=address] {Street} edge (address);
\node[attribute] (city) [right of=address] {City} edge (address);
\node[derived attribute] (age) [right of=person] {Age} edge (person);
\node[relationship] (uses) [below of=person] {Uses} edge (person);
\node[entity] (tool) [below of=uses] {Tool} edge[total] (uses);
\node[attribute] (tid) [left of=tool] {\key{ID}} edge (tool);
\node[attribute] (tname) [right of =tool] {Name} edge (tool);
\end{tikzpicture}
\end{document}
```

Listing 2: Exploiting the *TikZ-er2* package

This is the standard way to locally customize the elements in GraphViz. Since the various categories are also the styles names, the elements will automatically inherit their properties once converted in Ti*k*Z code. The label outside the square brackets is used to later connect the elements and it also appears in the diagram by default. Identifying key attributes could be a problem but GraphViz makes it possible to customize even this label with the notation `label="<label>"` (always inside square brackets). For example:

```
pid [style="attribute",
     label="\underline{ID}"];
```

In this way, the picture will use the label set with the key `label`, but for the connection phase it is the label *outside* the square brackets that matters, the *name*. Notice that the *name*s should be unique inside the dot2tex environment, therefore it is possible to exploit the key `label` also to differentiate the elements that might assume the same *name*; in the example this property has been used for:

```
Name[style="attribute"];
...
tname[style="attribute",label="Name"];
```

Elements position is decided by GraphViz and it is activated with the option neato: this automatically locates the elements near to other elements to which they are connected to.

The connections creation phase is a very simple task: the syntax is `<element1> -> <element2>`. Automatically, these connenctions inherith the provided style with:

```
edge [style="simple relation"];
```

This definition allows to declare the style globally. It will hold for all the connections unless we locally override the global definition:

```
Tool -> Uses[style="total relation"];
```

that sets up the connection to be of type *total*.

The result obtained with this approach is shown in figure 6; the compilation with the option `-shell-escape` will also create the files

```
main-dot2tex-fig1.dot
main-dot2tex-fig1.tex
```

provided the main file is named `main.tex`. The first one contains just the dot code and it can be opened with any Dot viewer while the second contains the tikzpicture code. These files can be seen as auxiliary files created in the translation process from the dot syntax to the Ti*k*Z one.

5 The object-oriented programming approach

5.1 A short introduction to object-orientation in Ti*k*Z

The concept of object-oriented programming in LaTeX graphics is still something quite new although LuaTeX seems to offer an interesting potentiality as per Giacomelli (2012).

On the contrary, the present work is focused on the module `oo` directly provided by Ti*k*Z. It can be loaded with

```
\usepgfmodule{oo}
```

in the preamble.

```
\documentclass[a4paper,11pt,x11names]{article}

\usepackage{tikz}
\usetikzlibrary{automata,shapes}
\usepackage{dot2texi}
\tikzset{entity/.style={draw=orange, fill=orange!20}}
\tikzset{attribute/.style={ellipse,draw=MediumPurple1, fill=MediumPurple1!20}}
\tikzset{multi attribute/.style={attribute,double}}
\tikzset{derived attribute/.style={attribute,dashed}}
\tikzset{relationship/.style={diamond,draw=Chartreuse2, fill=Chartreuse2!20}}
\tikzset{simple relation/.style={-}}
\tikzset{total relation/.style={-,double,double distance=1.5pt}}

\begin{document}
\begin{tikzpicture}
\begin{dot2tex}[styleonly,mathmode,codeonly,neato,options=-s]
digraph G {
edge [style="simple relation"];
// nodes
Person [style="entity"];
pid [style="attribute",label="\underline{ID}"];
Attribute [style="attribute"];
Name [style="attribute"];
Phone [style="multi attribute"];
Address [style="attribute"];
Street [style="attribute"];
City [style="attribute"];
Age [style="derived attribute"];
Uses [style="relationship"];
Tool [style="entity"];
tid [style="attribute",label="\underline{ID}"];
tname [style="attribute",label="Name"];
// edges
Person -> pid;
Person -> Attribute;
Person -> Name;
Person -> Phone;
Person -> Address -> Street;
Person -> City;
Person -> Age;
Person -> Uses;
Tool -> tid;
Tool -> tname;
Tool -> Uses[style="total relation"];
}
\end{dot2tex}
\end{tikzpicture}
\end{document}
```

Listing 3: Exploiting GraphViz

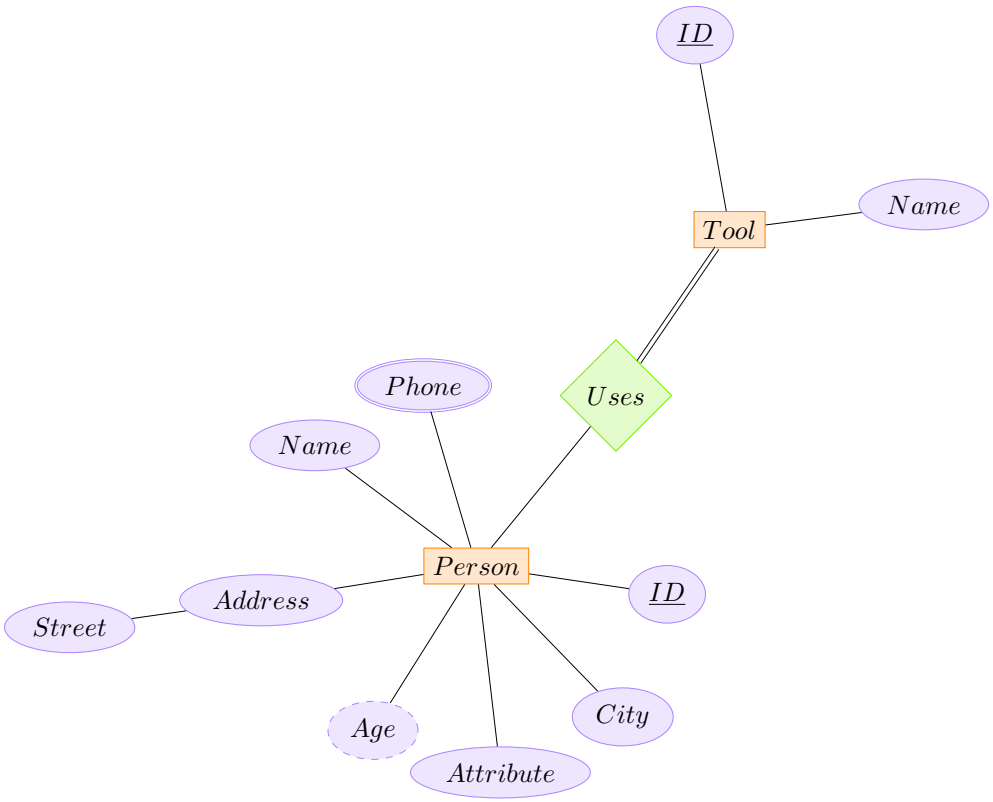


FIGURE 6: The ER diagram realized with GraphViz

The module provides a generic macros set to build classes, methods, attributes and objects. At the moment, as far as I know, there are no libraries developed in this way. This approach merges the advantages of the object-oriented paradigm along with the Ti*k*Z syntax.

In my opinion, the object-oriented paradigm is extremely useful to draw pictures that have common features repeated several times. An ER diagram falls exactly in this category because its entities, relationships and attributes have common features: the set of rules to represent them so that the standard is accomplished. Moreover, these elements are repeated several times because in each diagram there are usually several entities, relationships and attributes.

From an object-oriented point of view, things with common features are called *objects*. The following command shows how to create a new object:

```
\pgfoonew \obj=new constructor()
```

where `constructor()` is a *method* of a given *class*. In particular, the *constructor* method is devoted to instantiate new objects. Each object belongs to a given class; classes are to be defined as:

```
\pgfooclass{c-name}{
 ...
 code
 ...
}
```

while methods will be defined with:

```
\method m-name(parameters) {
 ...
 code
 ...
}
```

A class is characterized by *attributes* that describe objects properties. It is only possible to customize or activate these properties with methods. Attributes can be defined with

```
\attribute a-name;
```

in case they do not have a predefined value, or with

```
\attribute a-name=value;
```

when they do have a predefined value. For example, an object which prints some text may have an attribute `text` without a predefined value while the attribute `color text` could be set to, e.g., `blue` if the text should be mainly printed in `blue`.

5.2 The er-oo library

To make this work significant I developed a Ti*k*Z library named er-oo. You can download it from https://github.com/cfiandra/er-oo. As any other library, it will be loaded in the preamble with:

```
\usepackage{tikz}
\usetikzlibrary{er-oo}
```

after installing it. The library is ultimately a package, so the recommended way to install it is putting the files in the personal tree:

- `tikzlibraryer-oo.code.tex` and `er-oo.dtx` under `../texmf/tex/latex/er-oo/`
- `er-oo.pdf` under `../texmf/doc/er-oo/`

where `er-oo` is a directory to be created *ad hoc* and the `..` denotes the missing path which depends on the distribution and the operating system running on the machine. It also works copying `tikzlibraryer-oo.code.tex` in the directory of the main file. This solution, however, makes the library work only for that document while the former solution makes it work for all documents regardless of their position in the file system.

The library defines three classes for entities, relationships and attributes with a predefined look (at least in terms of colors), unlike the other techniques seen in this paper. The user is still free to change the basic look according to her personal preference with the apposite methods. Available methods are the same for all the classes though the class `attribute` has a further method named `set type`.

Before unveiling the code that draws the usual ER diagram, a short introduction to the library might be useful. Specifically I will take the `attribute` class as the reference because it provides all methods.

The list of the currently defined attributes is:

```
\attribute text;
\attribute border color=er-purple;
\attribute fill color=er-purple!20;
\attribute text color=black;
\attribute label;
\attribute type;
\attribute width=1.5cm;
\attribute height=0.35cm;
```

Some attributes *have* a default value, related to the elements look. The attributes *without* a default value are those requiring an input from the user: `text` is the attribute devoted to set the label of the element inside the diagram, the `label` attribute is responsible to identify an element (by its *name*) and `type` allows to select the attribute category (standard, multi attribute or derived attribute). This set of attributes is very good since it provides a good customization potentiality, but could be improved. A new attribute could be:

```
\attribute text opacity=1;
```

to set the opacity of the text. A good idea in defining new attributes is to use names similar to the keys already provided by Ti*k*Z: in such a way, the attribute will set the corresponding key with the default value (1 in this case to indicate an opaque text) maintaining names consistence between the Ti*k*Z layer and the object-oriented upper layer.

The user sets the attributes value with methods. The attribute `label`, for example, has a correspondent method:

```
\method set label(#1) {
 \pgfooset{label}{#1}
}
```

Methods, however, are not only designed to set attributes. Two methods can place one element: `draw` and `place`. The first one accepts as arguments a pair of coordinates (x, y) while the second one wants as argument a position relative to another element. Here is the definition of the method `draw`:

```
\method draw(#1,#2) {
\node [ellipse,
 attribute type={\pgfoovalueof{type}},
 draw=\pgfoovalueof{border color},
 fill=\pgfoovalueof{fill color},
 text=\pgfoovalueof{text color},
 minimum width=\pgfoovalueof{width},
 minimum height=\pgfoovalueof{height},
 ] (\pgfoovalueof{label})
 at (#1,#2) {\pgfoovalueof{text}};
}
```

Notice that the macro `\pgfoovalueof` sets Ti*k*Z keys inside `\node` retrieving values of the corresponding attributes. This also holds for `text` and `label` of the element.

Three alternative methods link elements: `connect`, `multi connect` and `total relation`. The first method simply draws a link between two elements exploiting the usual `\draw` syntax; the second, instead, links a single element (specified in the first argument) to a list of elements (specified in the second argument), as you may see in its definition:

```
\method multi connect(#1,#2) {
  \foreach \i in {#2}{
    \draw[-] (#1)--(\i);
  }
}
```

Notice that the usage should be in this form:

```
\myobject.multi connect(1,{2,3,4})
```

Braces are needed to protect the list of items in the second argument, since the list has to be comma separated.

Finally, the `total relation` is the method needed in case the relationship has to be of type *total*. Unfortunately it is not possible to use these methods subsequently as it happens in the traditional object-oriented languages. This means that the following syntax

```
\myobject.method one().method two()
```

and so the following code

```
\myobject.connect(1,2).total()
```

are forbidden (i.e., wrong).

To make code writing an easier and quicker process, it is possible mixing methods and creating new methods as a composition of pre-existent ones; for example, it is the case of:

```
\method set and draw(#1,#2,#3,#4) {
 \pgfoothis.set label(#1)
 \pgfoothis.text(#2)
 \pgfoothis.draw(#3,#4)
}
```

Thanks to the macro \pgfoothis the *contained* methods are always applied to the current object using the *container* method; for instance:

```
\myobject.set and draw(a,b,0,0)
```

is translated into

```
\myobject.set label(a)
\myobject.text(b)
\myobject.draw(0,0)
```

5.3 A real example with oo-er

As listings 1 and 2, also listing 4 provides the result already shown in picture 5.

In order to use the oo-er library, at first it is needed to instantiate the objects using the special method constructor:

```
\pgfoonew \myentity=new entity()
\pgfoonew \myrel=new relationship()
\pgfoonew \myattr=new attribute()
```

From that point on, objects are able to invoke those methods defined by each class. For example, the first entity, "tool", is placed with the method `set and draw`:

```
\myentity.set and draw(tool,Tool,1,0)
```

Since no other elements are currently present in the picture, it is not possible to place it with `set and place` as its parameters refer to an already placed element. We can therefore use `set and place` for "tool" attributes:

```
\myattr.set and place(tool-id,
  \underline{ID},left of=tool)
\myattr.set and place(tool-name,
  Name,right of=tool)
```

Indeed, once the "tool" entity has been placed, it is possible to refer to it via its *name* `tool`.

After the attributes definition, there is the connection phase:

```
\myentity.multi connect(tool,
  {tool-id,tool-name})
```

because the recommended way to proceed is to define and immediately connect the entity with its own attributes.

Notice that it seems possible to optimize the `multi connect` or `connect` methods in order to use only one argument. Assuming

```
\method x-multi connect(#1) {
 \foreach \i in {#1}{
   \draw[-]
     (\pgfoovalueof{label})--(\i);
 }
}
```

the previous connection may be obtained with

```
\myentity.x-multi connect(
  tool-id,tool-name)
```

since the method refers to the last placed object. However, in this way, it becomes mandatory to connect attributes and entities as soon as they are located because

```
\myentity.set and draw(x,x,0,0)
\myattribute.set and draw(x1,x1,1,0)
\myattribute.set and draw(x2,x2,0,1)
\myentity.set and draw(y,y,0,0)
\myentity.x-multi connect(x1,x2)
```

will not connect the entity "x" to the attributes "x1" and "x2", but rather "y" to "x1" and "x2". This does not happens with the current definition of `multi connect`:

```
\myentity.set and draw(x,x,0,0)
\myattribute.set and draw(x1,x1,1,0)
\myattribute.set and draw(x2,x2,0,1)
\myentity.set and draw(y,y,0,0)
\myentity.multi connect(x,{x1,x2})
```

correctly connects the entity "x" to the attributes "x1" and "x2".

6 Conclusion

The paper presented several techniques to draw ER diagrams with TikZ. These techniques mainly differ in terms of the programming style: the usual TikZ syntax is the base of the er library and the TikZ-er2 package, the dot language is required to exploit GraphViz and the object-oriented programming style is the key feature of the er-oo.

Users may prefer one tool or another according to their personal preferences or programming style, but they could wisely pick the tool according to the ER diagram size. In fact, GraphViz is recommended for large ER diagrams because of the dot capability to automatically place elements.

7 Acknowledgements

First of all I would like to thank Paulo Cereda; this paper indeed originated from an answer I gave on TeX.SX after a talk in chat with him: http://tex.stackexchange.com/q/78357/13304.

Last but not least, I would also like to thank the reviewer of the paper for his precious help in organizing my work better and making my bad English readable.

References

Chen, P. (1976). «The entity-relationship model—toward a unified view of data». *ACM Transactions on Database Systems (TODS)*, **1**(1), pp. 9–36.

Fauske, K. M. (2008). *The dot2texi package*. URL `http://www.ctan.org/pkg/dot2texi`.

```
\documentclass{article}

\usepackage{tikz}
\usetikzlibrary{er-oo}

\begin{document}
\begin{tikzpicture}[node distance=2.75cm]
\pgfoonew \myentity=new entity()
\pgfoonew \myrel=new relationship()
\pgfoonew \myattr=new attribute()
\myentity.set and draw(tool,Tool,1,0)
\myattr.set and place(tool-id,\underline{ID},left of=tool)
\myattr.set and place(tool-name,Name,right of=tool)
\myentity.multi connect(tool,{tool-id,tool-name})
\myrel.set and place(rel,Uses,above of=tool)
\myrel.total relation(rel,tool)
\myentity.set and place(per,Person,above of=rel)
\myattr.set and place(per-id,\underline{ID},left of=per)
\myattr.set type(derived attribute)
\myattr.set and place(per-age,Age,right of=per)
\myattr.set type() % to reset the derived attribute style
\myattr.set and place(per-name,Name,above left of=per)
\myattr.set type(multi attribute)
\myattr.set and place(per-phone,Phone,above of=per)
\myattr.set type() % to reset the multi attribute style
\myattr.set and place(per-addr,Address,above right of=per)
\myattr.set and place(street,Street,above right of=per-addr)
\myattr.set and place(city,City,right of=per-addr)
\myattr.multi connect(per-addr,{street,city})
\myentity.multi connect(per,{per-id,per-age,per-name,per-phone,per-addr,rel})
\end{tikzpicture}
\end{document}
```

Listing 4: Exploiting the object-oriented library `er-oo`

Fiandrino, C. (2012). «Graphviz e Ti*k*Z». *ArsTEXnica*, (13), pp. 4–10. URL `http://www.guit.sssup.it/arstexnica/`.

Giacomelli, R. (2012). «Grafica ad oggetti con LuaTEX». *ArsTEXnica*, (14), pp. 53–71. URL `http://www.guit.sssup.it/arstexnica/`.

Tantau, T. (2010). *The Ti*k*Z and PGF Packages*. URL `http://www.ctan.org/pkg/pgf`.

▷ Claudio Fiandrino
`claudio dot fiandrino at gmail dot com`

S*a*-Ti*k*Z: a library to draw switching architectures

Claudio Fiandrino

Abstract

The article illustrates how it is possible to draw some types of switching architectures in a simple manner. The S*a*-Ti*k*Z library provides not only the keys devoted to the drawing part, but also the keys devoted to customize the aspect of the architectures in the spirit of the Ti*k*Z syntax.

Sommario

L'articolo illustra come disegnare alcuni tipi di architetture di rete in modo semplice. La libreria S*a*-Ti*k*Z definisce non soltanto chiavi per il disegno, ma anche per personalizzare l'aspetto delle architetture utilizzando come base la sintassi di Ti*k*Z.

1 Introduction

Switching architectures are a combination of hardware and software that essentially move out to the proper direction data coming into the network. Each network node is also called *switching unit* or *module* and, in combination with the integrated circuits, form the hardware part of the architecture; the software part, instead, is in charge of switching the paths the data should follow.

Ti*k*Z is nowadays one of the most used packages for drawing purposes in LaTeX. It is built in a modular fashion by means of libraries: a part from some fundamental ones always loaded, the user is in charge of loading the libraries he needs in the preamble of his document. For example:

```
\usepackage{tikz}
\usetikzlibrary{list of libraries}
```

The list of libraries should be set as a comma-separated list.

The package/library S*a*-Ti*k*Z has been written in such a way that it could be loaded both as a package or as a Ti*k*Z library; therefore:

```
\usepackage{tikz}
\usetikzlibrary{switching-architectures}
```

and

```
\usepackage{sa-tikz}
```

are both valid.

S*a*-Ti*k*Z in particular, is able to represent the following switching architectures:

- Clos Networks;
- Benes Networks.

The project is also available on GitHub and its home page is http://cfiandra.github.com/Sa-TikZ/: several examples are provided and there is even a presentation in which the connections between input and output ports of the architecture are added one at a time.

The rest of the work is organized as follows:

- section 2 provides an overview on the architectures and the basic keys;
- section 3 explains how the drawing mechanism is implemented, considering also the related keys devoted to customize the drawing aspect;
- section 4 shows the keys devoted to quickly draw Clos and Benes Networks: these approaches allows to create network examples;
- section 5 concludes the work.

2 Overview on the architectures

2.1 Clos Networks

Clos Networks are 3-stages networks in which there is full interconnection among the consecutive stages; for example, a module belonging to the first stage has paths towards all the modules in the second stage.

A Clos Network with N input ports and M output ports is defined as shown in figure 1:

- m_1 represents the number of input ports per module in the first stage;
- r_1 represents the number of modules in the first stage;
- m_2 represents the number of input ports per module in the second stage;
- r_2 represents the number of modules in the second stage;
- m_3 represents the number of input ports per module in the third stage;
- r_3 represents the number of modules in the third stage.

Thus:

- $N = m_1 \cdot r_1$
- $M = m_3 \cdot r_3$

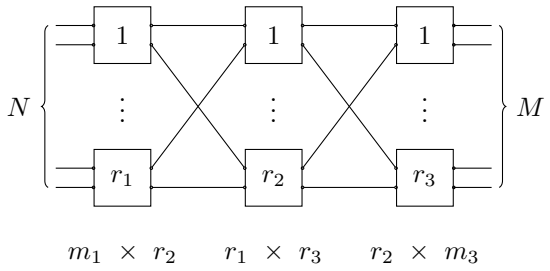


FIGURE 1: Clos network definition

Usually, as design parameters are always provided the number of inputs and outputs ports N and M and the number of modules in the first and third stage r_1 and r_3. The other parameters, m_1, m_3, r_2 and m_2, are automatically computed by S*a*-Ti*k*Z; notice that the number of modules in the second stage r_2 defines the type of the network:

- when $r_2 \geq m_1 + m_3 - 1$ the network is said to be *Strictly non Blocking*;
- when $r_2 = \max\{m_1, m_3\}$ the network is said to be *Rearrangeable*.

This is very important because the two types of networks have not only very different behaviours, but also a different structure. S*a*-Ti*k*Z, very simply, computes these constraints in background upon choosing the network type:

- the `clos snb` key defines a Strictly non Blocking Clos Network by computing:

```
\pgfmathtruncatemacro\rtwo{\mone+\mthree-1}
```

- the `clos rear` key defines a Rearrangeable Clos Network by computing:

```
\pgfmathtruncatemacro\rtwo{%
 max(\mone,\mthree)
}
```

The macro `\pgfmathtruncatemacro` allows to define other macros (`\rtwo` in this case) with a truncated value; this because, in Ti*k*Z, the result of an operation is something like `1.0`, where the notation `.0` means *an angle of 0 degree from the anchor placed on the left of the dot*. With a truncated macro instead, automatically, the result of an operation is stored as `1` in order to avoid this problem.

The macros `\mone` and `\mthree` are defined as:

```
\pgfmathtruncatemacro\mone{\N/\rone}
\pgfmathtruncatemacro\mthree{%
 \M/\rthree
}
```

where `\N`, `\rone`, `\M` and `\mone` are all macros associated to `pgfkeys`; their definition is similar, for example, `\N` is:

```
\pgfkeys{/tikz/.cd,%
      N/.initial=10,%
      N/.get=\N,%
      N/.store in=\N,%
}%
```

The `/tikz/.cd` is needed to locate the key under the path `/tikz`: this is helpful because later on the usual `\tikzset` could be used to assign values to the key. The key-path is very important because trying to assign values to a key using a wrong path not only makes the assignment null, but also gives rise to an error. By means of `N/.initial=10` it is possible to assign an initial value to the key; note that this is not the *default* value: indeed, it is used just in the first picture when no values for `N` are provided. When some values are assigned to `N` in a picture, they are stored in the macro `\N` (`N/.store in=\N`) and, for subsequent uses within the picture, `N` keeps the value of `\N` in case no other values are specified: this is possible by means of `N/.get=\N`.

The keys `clos snb` and `clos rear` should be introduced within the usual Ti*k*Z `node` syntax; an example:

```
\begin{tikzpicture}
\node[clos rear]{};
\end{tikzpicture}
```

The major potentiality of the library is that the architectures are drawn by simply choosing the design parameters; to this purpose the following keys should be used:

- `N` and `M` to choose the number of input/output ports for the first and third stage respectively;
- `r1` and `r3` to choose the number of modules for the first and third stage respectively.

Using the same input parameters allows to obtain different networks: examples are shown in figures 2 and 3.

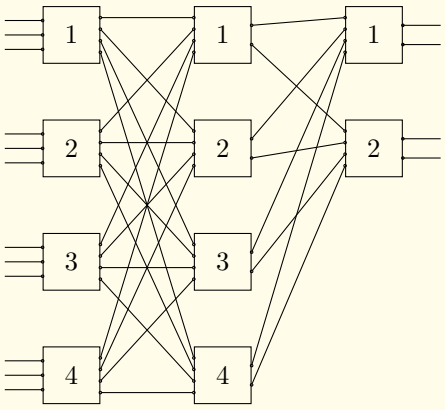
Notice one important fact: since the keys `clos snb` and `clos rear` need the input parameters specification, the possible customization should be declared before specifying the network type. This holds also for all the other keys representing network types.

2.2 Benes Networks

A Benes Network is a symmetric Rearrangeable Clos Network factorized with a factor 2 exploiting 2×2 modules.

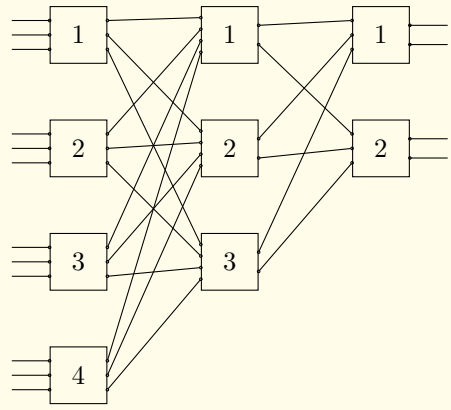
The number of the stages is not fixed, but depends on the size of the network. S*a*-Ti*k*Z takes into account this fact by providing two keys:

- the `benes` key displays the network without showing all the stages: the aspect is similar to a Rearrangeable Clos Network with three stages where the second one contains macro-modules hiding the actual inner stages;




```
\begin{tikzpicture}
\node[N=12,M=4,r1=4,r3=2,clos snb]{};
\end{tikzpicture}
```

FIGURE 2: Strictly non Blocking Clos Network




```
\begin{tikzpicture}
\node[N=12,M=4,r1=4,r3=2,clos rear]{};
\end{tikzpicture}
```

FIGURE 3: Rearrangeable Clos Network

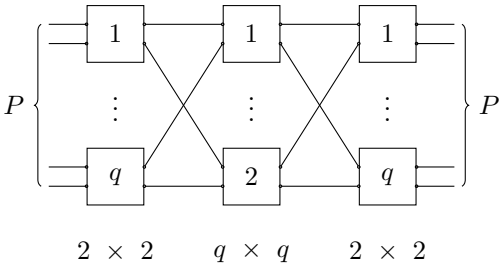


FIGURE 4: Benes Network definition

- the `benes complete` key displays the network showing all the stages.

The modules are 2×2: this implies that the number of input and output ports per module is fixed. As a consequence, the parameters m_1, m_2 and m_3 now have no meaning; indeed, in the code their respective macros `\mone`, `\mtwo` and `\mthree` been gathered into a single macro `\m`, defined as:

```
\pgfmathtruncatemacro{\m}{2}
```

The network simmetry means that the number of total input and output ports are the same; for the purpose, it has been defined a new key `P`, which simply replaces the previous keys `N` and `M`. Its definition is very similar to `N`'s one:

```
\pgfkeys{/tikz/.cd,%
      P/.initial=8,%
      P/.get=\P,%
      P/.store in=\P,%
}%
```

Actually, `P` can not assume any values: since the architecture is factorized with a factor 2 it can assume just

$$P = 2^p \qquad p = 2, 3, 4, \ldots$$

At the moment, S*a*-Ti*k*Z does not provide any check mechanism, therefore the user is responsible for the correct key setting.

The fixed number of input and output ports and the network symmetry influence the number of modules per stage. In the first and last stage, the parameters r_1 and r_3 and the respective macros `\rone` and `\rthree` have been gathered into a single macro `\r`, defined as:

```
\pgfmathtruncatemacro{\r}{\P/\m}
```

In the graphical representation of a Benes Network in figure 4, `\r` is called q.

The last stage is actually the third stage in case the `benes` key has been used: the second stage, thus contains only 2 macro-modules as per the definition of Rearrangeable Clos Network. If instead the `benes complete` key has been used, the last stage should be computed by means of the following formula:

$$\mathcal{S} = 2 \log_2 P - 1 \tag{1}$$

For example, if `P=16`:

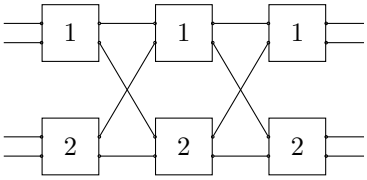
$$\mathcal{S}_{16} = 7$$

In this case, all the stages have the same number of modules, which is equal to q. The interconnections among all the stages are automatically drawn by S*a*-Ti*k*Z thanks to a specific algorithm "dBnc": its explanation, in detail, is reported in the appendix of the manual FIANDRINO (2013). The algorithm in principle works regardless the network size (specified by `P`), but for internal limitations of PGF it is difficult to characterize networks bigger than 128×128.

Examples of the difference between the `benes` and `benes complete` networks are shown in the

figures 5 and 6: since no value is given to P, the initial one is considered, that is 8.

Notice that each macro module in figure 5 actually comprises another Benes Network; an 8×8 Benes Network, indeed, is built recursively with two 4×4 Benes Networks:



For example, in a 32×32 Benes Network, it is possible to identify:

- two 16×16 Benes Networks;
- four 8×8 Benes Networks;
- eight 4×4 Benes Networks.

3 The drawing mechanism

The drawing mechanism is in charge of:

- drawing the modules of the various stages;
- drawing the interconnections between the modules.

3.1 The modules

The drawing method for the modules is very simple for clos snb, clos rear and benes networks because the number of the stages is always fixed, 3, while for benes complete networks is more difficult.

The idea behind all is to draw a path by positioning the modules at a given distance one after the other; the number of modules is always known: it is \rone, \rtwo and \rthree for clos snb and clos rear networks; as for benes networks, instead, it is \r for the first and third stage and 2 for the second stage.

For example, the code drawing the modules in the first stage is:

```
\foreach \i in {1,...,\rone}{
 \path let
 \n1 = {int(0-\i)},
 \n2={0-\i*\moduleysep}
  in node[module,#1,
    module opacity, yshift=1cm]
  (r1-\i) at +(0,\n2) {
   \pgfmathparse{int(multiply(\n1,-1))}
   \pgfmathresult};
...
}
```

During the path creation two things are computed:

- the label of the module by means of \n1 = {int(0-\i)}. Notice that the macro \i which iterates is not a truncated macro: this mechanism allows to print, after the multiplication by -1 done through \pgfmathparse{int(multiply(\n1,-1))}, a truncated number thanks to the int operation. Actually, the method will be improved in the next version by simply using the count option of the foreach.
- the distance of each module thanks to \n2={0-\i*\moduleysep}; notice that \n2 is used as y coordinate in at +(0,\n2): this means that at each iteration a constant distance is increased; the distance ultimately depends on \moduleysep, customizable through the module ysep key:

```
\pgfkeys{/tikz/.cd,%
 module ysep/.initial={1.5},%
 module ysep/.get=\moduleysep,%
 module ysep/.store in=\moduleysep,%
}%
```

Some styles and a personalization aspect are applied to the node: module,#1,module opacity,yshift=1cm; the module style is simply defined as:

```
\tikzset{module/.style={%
    draw,
    rectangle,
    minimum size=\modulesize,
    font=\modulefont,
    }
}
```

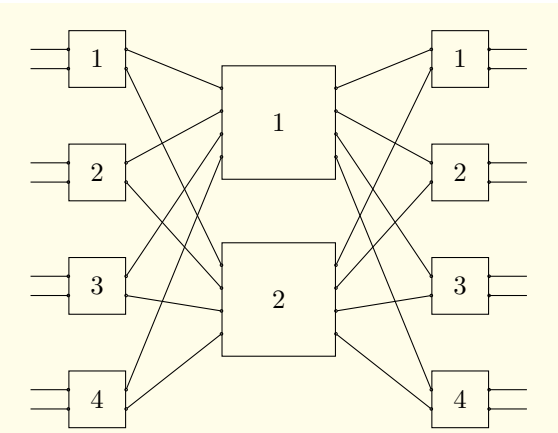
where \modulesize and \modulefont are macros representing the keys module size and module font.

The argument #1 of the style is peculiar: it helps to masquerade the labels by means of the key module label opacity. As all the TikZ opacity keys, when it is equal to 1, it does not alter the visibility, but when it is set to 0, it makes opaque the label. An example of usage is figure 7. The previous setting is implemented with the module opacity style, who sets the label with the desired opacity:

```
\tikzset{module opacity/.style={
  text opacity=\modulelabelopacity,
  }
}
```

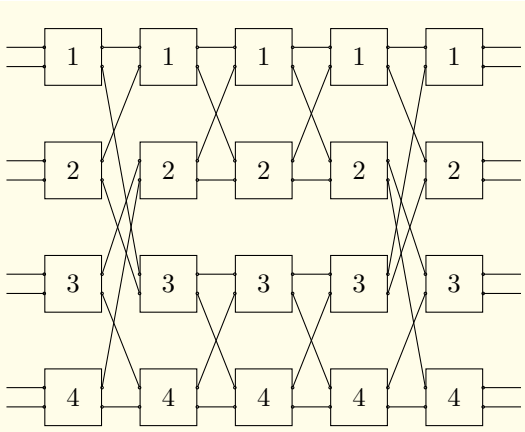
since the \modulelabelopacity is a macro customizable by the key module label opacity:

```
\pgfkeys{/tikz/.cd,%
 module label opacity/.initial={1},%
 module label opacity/.get=%
  \modulelabelopacity,%
 module label opacity/.store in=%
```



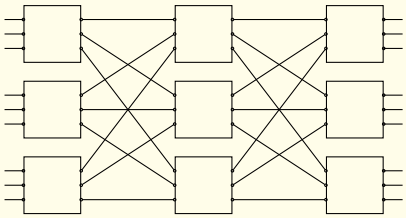

```
\begin{tikzpicture}
\node[benes]{};
\end{tikzpicture}
```

FIGURE 5: Benes Network




```
\begin{tikzpicture}
\node[benes complete]{};
\end{tikzpicture}
```

FIGURE 6: Benes complete Network



```
\tikzset{module size=0.75cm,
 pin length factor=0.5,
 module ysep=1,
 module xsep=2}
\begin{tikzpicture}[N=9,r1=3,M=9,r3=3]
\node[clos rear={module label opacity=0}]{};
\end{tikzpicture}
```

FIGURE 7: Clos Network without the labels

```
  \modulelabelopacity,%
}%
```

Finally, the `yshift=1cm` is just needed to let the first modules be placed on the `0` y coordinate: indeed, using `\n2={0-\i*\moduleysep}` shifts everything down and causes a possible waste of space.

For the second and third stages the procedure is identical: what changes is the x coordinate. For example, in the second stage it is:

```
at +(\modulexsep,\n2)
```

while in the third stage:

```
at +(2*\modulexsep,\n2)
```

What is `\modulexsep`? It is simply a macro associated to the `module xsep` key responsible to customize the horizontal distance between the stages; it is defined as:

```
\pgfkeys{/tikz/.cd,%
  module xsep/.initial={3},%
  module xsep/.get=\modulexsep,%
  module xsep/.store in=\modulexsep,%
}%
```

In the `benes complete` networks, this procedure is applied to all the stages; their number is computed by means of (1), which is translated in code with:

```
\pgfmathtruncatemacro{\stages}{
 2*round(log2(\P))-1
}
```

3.2 The interconnections

The interconnection drawing part is the core of the library. S*a*-Ti*k*Z is designed not only to automatically draw all the interconnections between input and output ports of the modules, but also to provide a method to subsequently improve the final picture by adding, for example, paths representing the connections between an input and output port of the architecture.

3.2.1 Locating the ports

In order to draw the interconnections, the first thing to do is to locate the input and output ports: this phase takes into account the input parameters because, according to their values, the number of ports changes. Supposing that there are `2` input and `3` output ports in a module, the ports are placed at the same distance one with respect to the others; this distance is computed according to:

$$in_{dist} = 1/(2+1) \qquad out_{dist} = 1/(3+1) \quad (2)$$

The code locating the ports of the first module of the `clos snb` networks is:

```
% INPUTS MODULE 1
% the number of inputs module one
% is exactly \mone
\pgfmathsetmacro\roneintervalspace{
 1/(\mone+1)
 }
\foreach \roneinput
 [evaluate=\roneinput as \roneinterval
  using \roneintervalspace*\roneinput]
 in {1,...,\mone}
 \draw ($(r1-\i.north west)!\roneinterval!
 (r1-\i.south west)-(0.5*\pinlength,0)$)
 node[scale=0.1]
 (r1-\i-front input-\roneinput){}-
 ($(r1-\i.north west)!\roneinterval!
 (r1-\i.south west)$)
 node[circle,draw,scale=0.1]
 (r1-\i-input-\roneinput) {};

% OUTPUTS MODULE 1
% the number of outputs of module one is
% the number of modules stage 2 \rtwo
\pgfmathsetmacro\roneintervalspace{
 1/(\rtwo+1)
}
\foreach \roneoutput
 [evaluate=\roneoutput as \roneinterval
  using \roneintervalspace*\roneoutput]
 in {1,...,\rtwo}
 \node[circle,draw,scale=0.1]
  (r1-\i-output-\roneoutput)
  at($(r1-\i.north east)!\roneinterval!
  (r1-\i.south east)$)  {};
```

The macro `\roneintervalspace` is computed according to (2): input and output differ according to the design parameters which influence the values of `\mone` and `\rtwo` (recall that they are the number of input ports per module of the first stage and the number of modules in the second stage respectively). It is interesting to analyze how the ports are ultimately located: this part is computed by `evaluate=\roneoutput as \roneinterval using \roneintervalspace*\roneoutput` within the foreach. The expression simply says that each port, progressively enumerated from 1, is located at an increasing distance with respect to the first port: indeed, the macro `\roneinterval` is set to the product of the basic distance `\roneintervalspace` times the counter reproseting the current port `\roneoutput`. The third port is located down 3 times than the first one, for example. After this definition, `\roneinterval` is used within the position definition by exploiting a facility of the `calc` library: it allows to determine a new position in terms of the distance between two known positions.

Then, for each port to be deployed, nodes with styles `circle,draw,scale=0.1` and particular names are instantiated: the names allows to later access the ports. The name choice is not causal; for internal ports, comprising the input and output ports of the all stages, the name is in the form: `r`*stage number*-*module number*-`input-`*port number*; for example: `r1-2-input-1`.

Notice that, for input ports of the first stage and output ports of the third stage, the port locating phase is not sufficient, but it is necessary also to draw the pins (the lines going out from the modules); in the previous code indeed, it is possible to notice a `\draw` mechanism which is able to realize this. Moreover, the pins have proper names which identify their final point: `r`*stage number*-*module number*-`front output-`*port number*; for example: `r3-3-front output-2`.

3.2.2 Drawing the interconnections

Once the ports have been located, it is possible to draw the interconnections. They concern: output ports of the first stage, input and output ports of the second stage and input ports of the third stage.

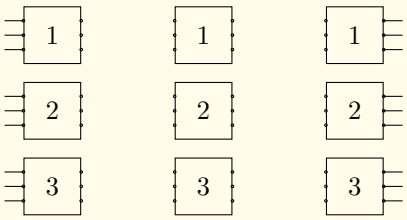
The code to create the interconnections for the `clos snb` networks is:

```
% Test if connections should be removed
\ifconnectiondisabled
 \relax
\else
 % DRAWING CONNECTIONS
 %% from r1 to r2
 \foreach \startmodule in {1,...,\rone}{
   \foreach \conn in {1,...,\rtwo}
     \draw(r1-\startmodule-output-\conn)
      -
      (r2-\conn-input-\startmodule);
 }
 %% from r2 to r3
 \foreach \startmodule in {1,...,\rthree}{
    \foreach \conn in {1,...,\rtwo}
      \draw(r3-\startmodule-input-\conn)
       -
       (r2-\conn-output-\startmodule);
 }
\fi
```

At first, a check if the connections should be displayed or not it is performed; for some purposes, perhaps, it is needed to have them hidden. Ti*k*Z allows also to define conditionals keys; indeed the `\ifconnectiondisabled` is defined as:

```
% disable the connections
\newif\ifconnectiondisabled%
\pgfkeys{/tikz/.cd,
 connections disabled/.is if=
  connectiondisabled
}%
\pgfkeys{/tikz/.cd,
 connections disabled/.default=.
 false
}%
```

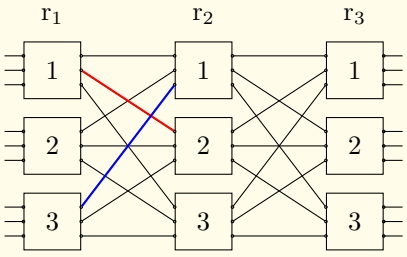
Therefore, it is sufficient to set to `true` the key `connections disabled` as per the example shown in figure 8.




```
\tikzset{module size=0.75cm,
pin length factor=0.5,
module ysep=1,
module xsep=2,
connections disabled=true}
\begin{tikzpicture}[N=9,r1=3,M=9,r3=3]
\node[clos rear]{};
\end{tikzpicture}
```

FIGURE 8: Rearrangeable Clos Network without connections

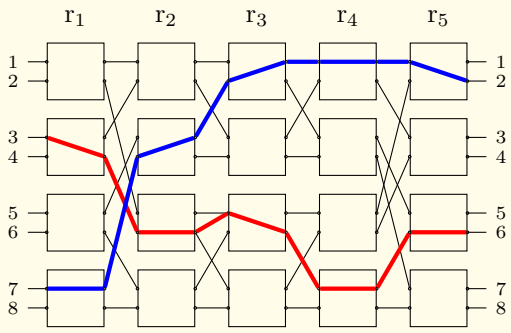
By default, `connections disabled` is set to `false`, so the drawing procedure can start. For each start module in the starting stage and for each output port this module has, a connection is established to the proper ending module of the next stage. Since during the location of the ports, they have been placed according to the input parameters, everything works with a simple trick: the starting module is identified by `r1-\startmodule-output-\conn` while the ending module by `r2-\conn-input-\startmodule`). In this way, correctly, `r1-1-output-2` is connected to `r2-2-input-1` and `r1-3-output-1` is connected to `r2-1-input-3` as per figure 9.




```
\tikzset{module size=0.75cm,module ysep=1,
pin length factor=0.5,module xsep=2,}
\begin{tikzpicture}[N=9,r1=3,M=9,r3=3]
\node[clos rear]{};
\foreach \x in {1,...,3}{
 \node[above=0.1cm of r\x-1] {r$_\x$};
}
\draw[red,thick] (r1-1-output-2)-
 (r2-2-input-1);
\draw[blue,thick] (r1-3-output-1)-
 (r2-1-input-3);
\end{tikzpicture}
```

FIGURE 9: Highlighting connections in a Clos Network

The procedure explained so far works for `clos snb`, `clos rear` and `benes` networks: for `benes`




```
\tikzset{module size=0.75cm,
  pin length factor=0.5,
  module ysep=1,
  module xsep=2,
}
\begin{tikzpicture}
\node[benes complete={
 module label opacity=0
 }]{};

% Labels: input ports
\newcounter{port}\setcounter{port}{0}
\foreach \startmodule in {1,...,4}{
 \foreach \port in {1,...,2}{
  \stepcounter{port}
  \node[left] at
  (r1-\startmodule-front input-\port)
  {\scriptsize{\theport}};
  }
}

% Labels: output ports
\setcounter{port}{0}
\foreach \startmodule in {1,...,4}{
 \foreach \port in {1,...,2}{
  \stepcounter{port}
  \node[right] at
  (r5-\startmodule-front output-\port)
  {\scriptsize{\theport}};
  }
}

% Labels: modules
\foreach \x in {1,...,5}{
 \node[above=0.1cm of r\x-1] {r$_\x$};
}

% Connection paths
\draw[red,ultra thick](r1-2-input-1)-
(r1-2-output-2)-(r2-3-input-2)-
(r2-3-output-2)-(r3-3-input-1)-
(r3-3-output-2)-(r4-4-input-1)-
(r4-4-output-1)-(r5-3-input-2)-
(r5-3-output-2);
\draw[blue,ultra thick](r1-4-input-1)-
(r1-4-output-1)-(r2-2-input-2)-
(r2-2-output-1)-(r3-1-input-2)-
(r3-1-output-1)-(r4-1-input-1)-
(r4-1-output-1)-(r5-1-input-1)-
(r5-1-output-2);
\end{tikzpicture}
```

FIGURE 10: Adding elements to the architecture

complete networks things are a bit different. Indeed, in order to draw the interconnections for that architecture the "dBnc" algorithm has been developed: before, to the best of my knowlegde, no automatic tools were able to perform such a task.

The algorithm takes into account the symmetric property of the Benes Networks and defines specific rules which are able to deal with the network size and the variable number of stages. In particular, the target is achieved thanks to the concepts of *module applicability range* and *repeatability* of the rules.

One of the peculiarities of the algorithm is that it needs to determine whether a port number is odd or even: Ti*k*Z itself does not provide a function within its mathematical engine. Thus, S*a*-Ti*k*Z defines a macro `\pgfmathisodd` able to perform the check and to store the result into another macro:

```
\newcommand*{\pgfmathisodd}[2]{
 \pgfmathparse{mod(#1,2)}
 \pgfmathtruncatemacro\res\pgfmathresult
 \global\expandafter\edef\csname
 #2\endcsname{\res}
}
```

An example of usage:

```
\pgfmathisodd{32}{output}
\ifnum\output=1
  \node{\output};
\fi
```

3.3 Adding elements

By means of the names of the modules' ports and the pins it is possible to add elements to the architecture; these elements could be:

- port labels;
- connection paths within the architecture;
- stage labels.

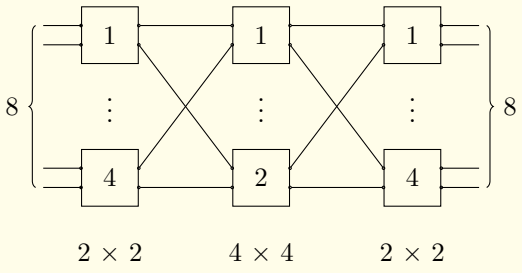
A complete example is shown in figure 10.

4 Network examples

This section takes into account how to create example pictures of switching architectures; the main keys devoted to the purpose are:

- `clos snb example`;
- `clos rear example`;
- `clos example with labels`.

With the first two keys it is possible to draw examples showing the design parameters. Although there are no equivalent keys for Benes Networks, it is even possible to represent them by simply selecting in the proper way the design parameters; an example is figure 11.




```
\begin{tikzpicture}[N=8,r1=4,M=8,r3=4]
\node[clos rear example] {};
\end{tikzpicture}
```

Figure 11: Benes Network Example

An example of Clos Network, instead, is shown in figure 12.

In these drawings the number of modules per stage is fixed; in the code, this simplification is evident looking at how the modules are placed. For example, the modules in the third stage are located with the code:

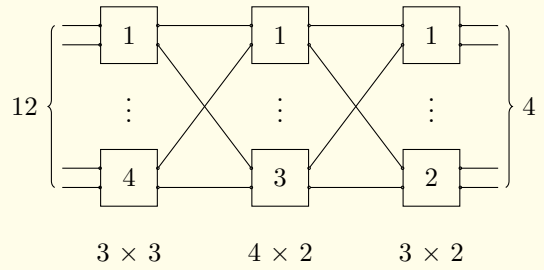
```
\node[module,#1,module opacity]
 (r3-1) at (2*\modulexsep,0) {1};
\node[below of=r3-1,yshift=0.75ex]
 (r3-dots) {\vdots};
\node[module,#1,module opacity,
 below of=r3-dots]
 (r3-2) {\rthree};
```

No particular methods are needed: simple Ti*k*Z nodes are drawn with the proper styles, described in subsection 3.1. Also the interconnection phase is more simple: just two input/output ports per module are needed, but, for simplicity in developing the code, the mechanism used is still the one described in subsection 3.2. It is interesting to analyze how the braces comprising the input and output ports are realized: thanks to the possibility to access to the pin names, `front input` and `front output` port, the braces are added by means of the `decorations.pathreplacing`.

The definition is:

```
\draw[decorate,decoration={brace}]
 ($(r1-2-front input-2)-(0.1,0)$)-
 ($(r1-1-front input-1)-(0.1,0)$)
 node[midway,
  left=0.1cm,
  set math mode labels]
 {\Nlabel};
\draw[decorate,decoration={brace}]
 ($(r3-1-front output-1)+(0.1,0)$)-
 ($(r3-2-front output-2)+(0.1,0)$)
 node[midway,
  right=0.1cm,
  set math mode labels]
 {\Mlabel};
```

The `calc` library helps in positioning the start and ending point of the brace a little shifted with



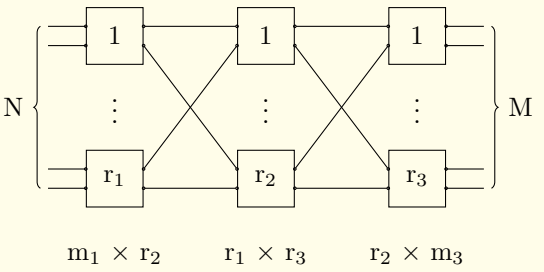

```
\begin{tikzpicture}[N=12,M=4,r1=4,r3=2]
\node[clos rear example]{};
\end{tikzpicture}
```

FIGURE 12: Rearrangeable Clos Network Example

respect to the input and output ports; in order to display the label, a node has been introduced with the options:

- `midway` to put it in the middle of the path;
- `left/right` to make the label not overlap the brace;
- `set math mode labels` is a key discussed later.

With respect to the keys `clos snb example` and `clos rear example`, the `clos example with labels` key differs only in what it prints: generic labels rather than the design parameters. In order to customize the labels, S*a*-Ti*k*Z defines several keys with the syntax: `<key> label` where `<key>` is one of the design parameters; for example, the key `N label` allows to customize the label defining the total number of input ports of the first stage. The figures 1 and 4 are drawn in this way.

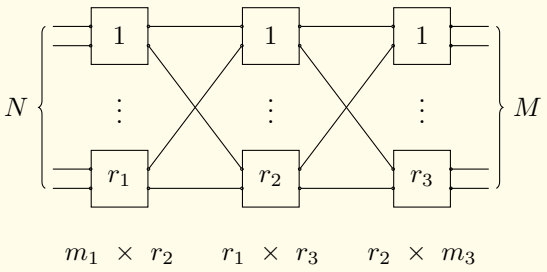



```
\begin{tikzpicture}[N=8,r1=4,M=8,r3=4]
\node[clos example with labels] {};
\end{tikzpicture}
```

FIGURE 13: Clos Network Example with labels

When using `clos example with labels` networks, the input parameter definition is useless; for instance, figure 13 shows that those definitions are simply ignored.

What it is also possible to see from figure 13, is that the letters are displayed in roman; to have them in italic, the key `set math mode labels` is of help as per figure 14. The key `set math mode`




```
\begin{tikzpicture}[
 set math mode labels=true]
\node[clos example with labels] {};
\end{tikzpicture}
```

FIGURE 14: Clos Network Example with labels in italic

`labels` is a boolean key by default set to false; its definition is a bit complex:

```
\tikzset{math mode labels/.style={%
   execute at begin node=$,%
   execute at end node=$,%
  }
}
\pgfkeys{/tikz/.cd,%
   use math mode labels/.is choice,%
   use math mode labels/true/.style={%
    math mode labels
   },%
   use math mode labels/false/.style={},%
}%

\tikzset{set math mode labels/.style={%
 use math mode labels=#1,%
 },%
 set math mode labels/.default=false,%
}
```

To understand the definition it is necessary to start from the key `math mode labels`; it is a key that before and after the text node *physically* adds a pair of $ thanks to `execute at begin node=$` and `execute at end node=$`: in such a way, the text of the node is rendered in math-mode. Subsequently, a boolean key `use math mode labels` is defined: when the choice is set to `true` then the key `math mode labels` becomes active. On top of `use math mode labels` there is, finally, `set math mode labels`: it receives an argument passed to `use math mode labels` and this argument by default is set to `false`. To summarize, when `set math mode labels=true` is set, `true` is passed to `use math mode labels`; this makes `math mode labels` active. As a consequence the labels are rendered in italic.

5 Conclusions

The article presented S*a*-Ti*k*Z by discussing the code of the library and showing several examples.

This is an example of how the Ti*k*Z code can be used for practical purposes providing an easy and manageable tool thanks to the key-interface. Indeed, one of the aims of the library, is to give to the students the possibility to verify the correctness of their exercises; as an instrument, it has been practically used in the academic course of Switch and Router Architectures held by prof. Paolo Giaccone in Politecnico di Torino.

References

FIANDRINO, C. (2013). *Sa-TikZ*. URL `http://www.ctan.org/pkg/sa-tikz`. Available typing in the terminal `texdoc sa-tikz`

▷ Claudio Fiandrino
`claudio dot fiandrino at gmail dot com`

Realizzazione di un layout complesso: la prima pagina de La Settimana Enigmistica

Gianluca Pignalberi

Sommario

Realizzare una pagina dalla struttura complicata con LaTeX coinvolge la sinergia di più pacchetti specializzati. Vediamo come usare Ti*k*Z, shapepar e textpos nel tentativo di realizzare parte della copertina di un vecchio numero de La Settimana Enigmistica.

Abstract

Making a complex page layout with LaTeX involves the usage of more than one specialized packages. We will see how to use Ti*k*Z, shapepar and textpos while attempting to reproduce part of the cover of an old issue of La Settimana Enigmistica.

1 Introduzione

Recentemente ho dovuto festeggiare il compleanno particolare di un cliente dell'edicola della mia famiglia, affezionato da anni (da ben prima che io nascessi) alla Settimana Enigmistica. Il distinto signore ha compiuto 100[1] anni nel pieno delle facoltà fisiche e mentali.

La mia urgenza coi regali è da sempre quella di intuire cosa potrebbe piacere al festeggiato (sono rare le volte in cui regalo qualcosa, in genere libri o dischi, che mi è particolarmente piaciuto e che vorrei condividere con altri). Questa volta l'urgenza era amplificata dall'età del festeggiato: cosa mai può volere un centenario?

La prima cosa che ho pensata è stata: "Perché non il primo numero della Settimana Enigmistica?" Credetemi: è pressoché introvabile, essendo uscito nel 1932. Non l'ho trovato, ovviamente, al Servizio Arretrati dell'editore; non l'ho trovato su eBay. Avevo troppo poco tempo per setacciare altri mercatini online o fisici alla ricerca della pubblicazione, ricerca che sarebbe potuta essere lunga e infruttuosa.

Trovata una scansione molto rovinata e armato dei soli GIMP e LaTeX ho cercato di fare un lavoro dignitoso e dotato di una copertina supplementare a coprire il tutto.

LaTeX non è il programma che chiunque vorrebbe usare per impaginare un progetto complesso. Il fatto che usare un programma visuale di tipografia renda più semplici alcuni passaggi è fuori discussione. È altrettanto fuori discussione il fatto che LaTeX e la sua programmabilità tolgono molto del lavoro tedioso lasciando intatta la parte più creativa, in questo caso quella programmativa.

Nell'articolo parlerò brevemente delle operazioni di pulizia delle pagine della rivista (nel paragrafo 2) per passare poi alla realizzazione della copertina supplementare, cioè alle sezioni di interesse specifico: il paragrafo 3 descriverà succintamente la copertina realizzata, il paragrafo 4 esporrà come disegnare lo schema del cruciverba con Ti*k*Z, il paragrafo 5 rappresenterà le operazioni di realizzazione dei paragrafi di forma arbitraria e del pacchetto shapepar, quindi il paragrafo 6 parlerà del posizionamento degli elementi sulla pagina mediante textpos.

2 Piccola digressione sul fotoritocco

La preparazione dell'interno del regalo ha comportato diverse ore di lavoro in un campo, quello del fotoritocco, che avrebbe poco a che vedere con la tipografia digitale. Molto tempo per un numero di pagine esiguo: solo sedici. Il PDF reperibile in rete (`http://avaxhome.ws/magazines/hobbies_leisure_time/sett_enigm_001.html`) è in condizioni degradate, con pagine strappate, macchiate, storte e la prima anche scritta in un angolo.

Ho effettuato un intervento di pulizia imperfetto, aiutato in tutto da GIMP. Ne vedete un risultato nella figura 1. Il fatto che fino alla versione 2.6 GIMP non gestisca nativamente il colore CMYK[2] non è stato pregiudizievole della qualità finale perché dovevo ottenere un risultato conforme al bianco e nero — con alcune parti in grigio — originale.

Vediamo un beve elenco descrittivo delle operazioni eseguite:

Conversione: trasformazione dell'originale PDF in un formato raster adatto al fotoritocco. Nel caso in esame ho preferito il PNG al TIFF visto che è considerato migliore per i disegni al tratto, oltre a essere direttamente importabile da LaTeX. Ho imposto la conversione a 300 dpi e questa ha restituito le pagine con tutti i difetti e le sfumature presenti in origine e pronte per l'elaborazione.

1. Scrivo cento in cifre anziché in lettere per rimarcare il traguardo.

2. La recente versione 2.8 di GIMP aveva tra le caratteristiche attese proprio la gestione nativa del colore CMYK.

LA SETTIMANA ENIGMISTICA

Redazione - Amministrazione - Pubblicità
MILANO (132)
Via Enrico Nöe, 43 - Tel. 20-866

Periodico di Giochi, Enigmi, Parole crociate, Scacchi, Dama, Bridge, Sciarade, ecc.

Un numero . . L. 0,50 - Arretrato . . L. 1.—
ABBONAMENTI: Anno L. 20 - Semestre L. 11.—

PREMESSA

Ci siamo studiati di presentarvi, nei limiti della maggiore semplicità e praticità possibili, quello che all'estero vien chiamato « La Palestra del cervello » e cioè un giornale che, completo pur essendo economico, vi riesca divertente e istruttivo nel contempo, il compagno che sappia scacciarvi la noia quando vi assale, quello che in treno vi faccia dimenticare la monotonia del viaggio, che alla sera insomma, dopo il traviglio diurno, vi conceda lo spasso di un'ora di riposo e di diletto insieme, il passatempo, in una parola, il più economico per tutti: dall'enigmista più provetto alla semplice massaia, dallo scacchista esperto al principiante di Dama, dal vecchio pensionato al giovinetto studente.

Il nostro programma è specificatamente espresso nella materia già pubblicata in questo numero e non à bisogno di speciali parole perchè possa essere meglio illuminato e chiarito.

LA DIREZIONE.

2. Crittogramma

1.	1	2	3	4	5	3	6	7		generale ateniese
2.	8	9	10	3	11	12	3	9		animale domestico
3.	1	10	1	11	2	1	13	7		registro civico
4.	14	1	12	15	5	1	2	7		assegnare il prezzo
5.	1	2	16	1	6	3	9			mobile
6.	12	1	15	2	9					pianta della gloria
7.	6	15	9	6	7	10	9			intestino tenue
8.	9	2	10	1	16	7	10	5	9	fregio
9.	10	3	10	3	14	7				capitale assira
10.	1	8	3	10	9					seme d'uva
11.	5	3	16	17	2	9				impronta
12.	9	2	3	9	12	9				orologio
13.	10	1	2	8	9	5	3	8	9	come l'oppio
14.	9	10	3	8	7					pietra preziosa
15.	10	1	18	1	2	3	9			martire santo

Al posto delle cifre vanno destinate le corrispondenti lettere che formino le parole di cui è dato al lato il significato. A cifra uguale corrisponde lettera uguale. La prima e la quarta colonna di lettere, lette successivamente dall'alto al basso, daranno un noto proverbio.

ABBONATEVI! Concorrerete al premio e riceverete per un anno il Giornale a Casa!

Figura 1: Una pagina della rivista prima e dopo le operazioni di fotoritocco.

Rotazione e taglio: raddrizzamento e rifilatura di alcune delle pagine, digitalizzate storte e oltre il bordo.

Equalizzazione: distribuzione uniforme dei colori o dei toni di grigio presenti nell'immagine. Ho equalizzato tutte le pagine per tentare di non eliminare le parti inchiostrate che si confondono col colore molto grigio della carta. In alcuni casi ho dovuto equalizzare solo alcune aree prima di equalizzare tutta la pagina.

Binarizzazione: imposizione di una soglia prima della quale tutti i colori diventano nero e oltre la quale i colori diventano bianco. Purtroppo è inevitabile che molti difetti rimangano (macchie isolate o concomitanti all'inchiostro) e alcune parti del documento spariscano (grazie, filetti e altri particolari).

Colorazione: eliminazione dei pixel indesiderati colorandoli di bianco se neri e viceversa. La qualità di quest'ultima operazione dipende molto dalla quantità di tempo e di pennelli a disposizione.

In tre casi queste operazioni hanno subìto l'integrazione di un'altra operazione un po' più complessa ma necessaria a preservare il grigio presente sulle scacchiere nei giochi di dama e scacchi. Varii passi articolano l'operazione che va eseguita dopo la rotazione e il taglio:

Livelli: Per prima cosa bisogna creare una copia dell'immagine e porla in un livello separato dall'originale.

Equalizzazione e binarizzazione: bisogna poi equalizzare e binarizzare il nuovo livello così come visto in precedenza.

Opacità: una volta che il livello è stato reso bianco e nero bisogna ridurre l'opacità (cioè aumentare la trasparenza) del livello stesso. In questo modo i pixel bianchi restano bianchi mentre i pixel neri degradano a grigi.

A questo punto possiamo equalizzare e binarizzare anche il livello principale ricordandoci, prima di effettuare la colorazione, di unire i livelli riportandoli a uno solo.

3 La copertina supplementare

Avendo deciso di corredare il numero del settimanale con una copertina, era fondamentale capire cosa metterci. Alla fine ho deciso di comporre la prima di copertina in maniera molto semplice. Come vediamo nella figura 2, dall'alto in basso, la copertina è composta da una specie di testatina riportante l'età e la data di compleanno del festeggiato e due filetti uno sotto l'altro, tutto a ricordare la testatina tipica del settimanale. Quindi, centrati (o, come dice un mio collega *graphic designer*, imbandierati a epigrafe), una didascalia


100 ANNI 28 SETTEMBRE 2012

EDIZIONE SPECIALE DE

LA SETTIMANA ENIGMISTICA

PER IL

CENTESIMO COMPLEANNO

DI

VITTORIO VITTORI

A CURA DE

LA SUA EDICOLANTE

BENILDE PICCIONI

FIGURA 2: Copertina del regalo.


IL GIORNALE DI ENIGMISTICA PRIMO PER FONDAZIONE E PER DIFFUSIONE

Anno XXXIX - N. 2001 Spedizione in abbonamento postale (Gr. 2/70) 1 Agosto 1970

LA SETTIMANA ENIGMISTICA

ESCE IL SABATO

PERIODICO DI GIUOCHI - ENIGMI - PAROLE CROCIATE SCACCHI - DAMA - REBUS - SCIARADE ecc.

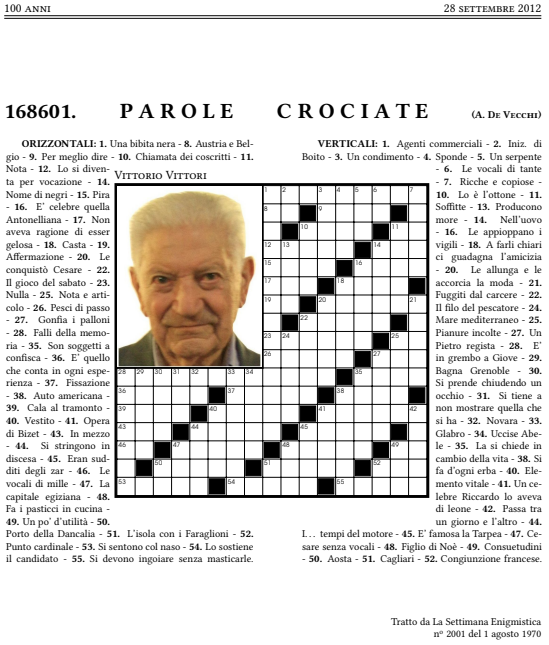
168601. PAROLE CROCIATE (A. De Vecchi)

FIGURA 3: Copertina de La Settimana Enigmistica nº 2001 del 1 agosto 1970.

a indicare la "specialità" del contenuto, il destinatario e il curatore. Il titolo/logo, inframmezzato alla didascalia, è tratto dalla pagina Wikipedia de La Settimana Enigmistica. Ho usato il font Linux Libertine maiuscoletto per tutte le scritte.

Nella quarta di copertina volevo un altro elemento caratterizzante il settimanale riprodotto. Avendo trovato in rete una scansione della prima pagina di un numero de La Settimana Enigmistica del 1970 (mostrata nella figura 3), ho deciso di riprodurre quella continuando però a usare Linux Libertine come font principale e Avant Garde in luogo di quello originale, di concezione simile a Futura, per i numeri del cruciverba. Non potevo tralasciare la testatina uguale a quella della copertina. Vediamo il risultato nella figura 4.

Mentre la produzione della copertina non ha segreti neanche per un novizio di LaTeX, la produzione della quarta è un po' più articolata e le dedicheremo i prossimi paragrafi per enuclearla nel dettaglio.


100 ANNI 28 SETTEMBRE 2012

168601. PAROLE CROCIATE (A. De Vecchi)

VITTORIO VITTORI

Tratto da La Settimana Enigmistica nº 2001 del 1 agosto 1970

FIGURA 4: Quarta di copertina del regalo.

4 Lo schema del cruciverba

Il primo elemento che ho disegnato è stato lo schema del cruciverba. Ho fatto una ricerca sul CTAN per cercare un pacchetto per disegnare schemi di parole incrociate (*crossword*) ma ne ho trovati solo di diversi, più simili alle parole composte sullo Scarabeo (*Scrabble*) che non quello tipico dei cruciverba. Dunque, armato di tipometro/righello, carta, matita e Ti*k*Z (TANTAU, 2010) ho misurato e riprodotto lo schema voluto.

Vediamone le misure, ricordando che sono solo approssimative viste le misure in gioco e la porosità della carta da stampa su cui ho rilevato le misure. Lo schema è quadrato di misura $10{,}4 \times 10{,}4$ cm. Lo spessore della cornice è 0,8 mm; le caselle sono quadrate in numero di 17×17 (quindi 6,11 mm se

scegliamo lo spessore `thin` dei filetti in Ti*k*Z) col numerino posto nell'angolo in alto a sinistra. Lo scavo per l'immagine è di 8×10 caselle e la sua cornice misura 0,6 mm. Infine le caselle annerite hanno una corona quadrata bianca di circa 0,5 mm.

Il codice per disegnare il cruciverba (senza numeri né caselle annerite ma con l'immagine) è il seguente:

```
1  \begin{tikzpicture}
2  \pgfsetlinewidth{.8mm}
3  \draw (-.04,-.04) rectangle
       (10.44,10.43);
4  \pgfsetlinewidth{.6mm}
5  \draw (0,4.277) rectangle
       (4.888,10.4);
6  \draw[step=6.11mm,thin] (0,0)
       grid (10.4,10.4);
7  \pgfdeclareimage[width=4.84cm]{
       vitt}{vittorio2.jpg};
8  \pgftext[at=\pgfpoint{0.02cm
       }{4.31cm},left,base]{
       \pgfuseimage{vitt}};
9  .
10 .
11 .
12 \end{tikzpicture}
```

La riga 2 del codice impone uno spessore delle linee di 0,8 mm e la riga 3 disegna il quadrato esterno facendo attenzione che tutta la linea sia esterna allo schema. Quindi viene impostato lo spessore delle linee a 0,6 mm (riga 4) per disegnare la cornice dell'immagine (riga 5). La riga 6 contiene il comando per disegnare la griglia a cui impostiamo il passo e lo spessore dei filetti. Infine la riga 7 dichiara un alias all'immagine. Questo verrà usato alla riga successiva per porre l'immagine entro il suo scavo.

Avrei potuto porre i numeri e le caselle annerite una per una, cosa che in un certo senso va fatta comunque. Un conto è però scrivere ogni volta il comando per porre l'elemento nella coordinata assoluta con ripetizione della scrittura del codice e del calcolo esatto delle coordinate, un conto è scrivere un paio di macro che prendano delle coordinate relative al numero di casella e l'eventuale numerino da stampare e poi facciano tutto da sé.

Ho dunque scritto due macro: `\putblind` e `\putnumber`. Servono rispettivamente a posizionare le caselle annerite e i numeri nelle caselle. Questi sono i loro codici:

```
\newcommand{\putnumber}[3]{%
  \postocoord{#1}{#2}
  \pgftext[at=\pgfpoint{\xcoord
     }{\ycoord},left,base]{
     \begingroup\fontsize{0.5
     \fontdimen6\font}{
     \baselineskip}\fontfamily{
     pag}\selectfont\raisebox{10
     pt}{#3}\endgroup}
}
\newdimen\blindside
\setlength\blindside{.511cm}
\newcommand{\putblind}[2]{%
  \postocoord{#1}{#2}
  \draw [thin, fill] (\xcoord,
     \ycoord) rectangle (\xcoord
     +\blindside,\ycoord+
     \blindside);
}
```

Notiamo due cose: entrambe dipendono da una terza macro chiamata `\postocoord` (da *position to coordinate*) che prende in input due numeri relativi alle coordinate — posizione orizzontale e verticale — delle caselle (nel nostro caso i numeri andranno da 1 a 17 come il numero di caselle per asse, ma la macro non controlla il rispetto del vincolo) e `\putnumber` imposta la dimensione del font dei numeri, oltre che il font stesso, a basso livello perché la vecchia versione di Ti*k*Z usata allora non accetta i comandi tipici di LaTeX per selezionare la dimensione del font.

Rimane ora da studiare la macro `\postocoord`. Il suo codice è il seguente:

```
\newcount{\tmp}
\newdimen{\xcoord}
\newdimen{\ycoord}
\newdimen{\shift}
\setlength{\shift}{.051cm}
\def\scalefact{.61}
\newcommand{\postocoord}[2]{%
  \tmp=#1
  \advance\tmp by -1
  \xcoord=\tmp cm
  \xcoord=\scalefact\xcoord
  \advance\xcoord by \shift
  \tmp=#2
  \advance\tmp by -1
  \ycoord=\tmp cm
  \ycoord=\scalefact\ycoord
  \advance\ycoord by \shift
}
```

Ognuna delle due coordinate viene normalizzata per sottrazione a 0 (ricordiamo che per noi utenti della macro le coordinate partono da 1), quindi viene trasformata in una posizione assoluta nell'unità di misura corretta moltiplicandola per `\scalefact` e sommandola a `\shift`.

Non ci resta che scrivere all'interno della figura Ti*k*Z, dopo aver disegnato lo schema, una serie di comandi `\putblind{x}{y}` (per esempio `\putblind{10}{4}`) e `\putnumber{x}{y}{numero}` (per esempio `\putnumber{9}{17}{1}`) con `x` la posizione delle ascisse, `y` la posizione delle ordinate e `numero` il

numero da stampare nella casella per avere lo schema completo come quello della figura 4.

Notiamo che la posizione delle caselle annerite è più decentrata man mano che ci spostiamo a destra e verso l'alto nello schema. Dopo aver consegnato il regalo ho pensato che applicare un correttivo legato ai filetti al parametro di spiazzamento (`\shift`) avrebbe senz'altro giovato. Una rapida prova in tal senso mi ha mostrato che la posizione delle caselle annerite migliora; peggiora però il posizionamento dei numeri che ora è ottimale.

A questo punto posso ipotizzare che sia da ripensare la macro che trasforma le posizioni in coordinate.

5 Paragrafi con forme arbitrarie

La lista delle definizioni è divisa in due colonne, una relativa alle definizioni orizzontali e l'altra relativa a quelle verticali. Lo schema del cruciverba è ospitato in uno scavo nelle colonne. Il risultato finale per chi debba disegnare il layout della pagina è quello di avere un paio di colonne a forma di parentesi quadre con dentro lo schema del cruciverba.

Sebbene possiamo descrivere la forma delle colonne ricorrendo a TEX, ho trovato più comodo ricorrere al pacchetto shapepar (Arseneau, 2006), realizzato proprio per semplificare il compito di disegnare dei paragrafi con forma arbitraria e per superare nel contempo le difficoltà del comando `\parshape` di TEX.

In base alla sintassi della descrizione delle forme, le due "parentesi" aperta e chiusa hanno le seguenti definizioni:

```
1  \def\obracketshape{%
2    {0}
3    {0}b{0}\\
4    {0}t{0}{10}\\
5    {.8}t{0}{10}\\
6    {.9}t{0}{4.2}\\
7    {15.5}t{0}{4.2}\\
8    {15.6}t{0}{10}\\
9    {17}t{0}{10}\\
10   {17}e{1}
11 }
12 \def\cbracketshape{%
13   {0}
14   {0}b{0}\\
15   {0}t{1.5}{9.5}\\
16   {.8}t{1.5}{9.5}\\
17   {.9}t{6.8}{4.2}\\
18   {15.5}t{6.8}{4.2}\\
19   {15.6}t{1.5}{9.5}\\
20   {17}t{1.5}{9.5}\\
21   {17}e{1}
22 }
```

Alle righe 2 e 13, come da richiesta del pacchetto, definiamo il punto centrale delle definende forme. Nel caso in esame il punto centrale corrisponde all'estremità sinistra delle parentesi. Sebbene la corrispondenza sia geometricamente sbagliata, il codice descrive bene e senza errori quanto proposto. Le righe 3 e 14 danno inizio alla forma (flag `b`, da *begin text block*), forma che inizia nel punto 0[3] della linea più alta. Man mano che procediamo a "disegnare" le parentesi verso il basso specifichiamo che le linee di testo fino a una certa altezza (.8, righe 4, 5, 15 e 16 del codice) sono lunghe 10 o 9.5 (la differenza è stata necessaria per far venire le due colonne ben bilanciate), mentre da una certa altezza in poi (.9, righe 6, 7, 17 e 18 del codice) le righe sono lunghe solo 4.2. Da quel punto in poi (righe 8, 9, 19 e 20) le righe sono di nuovo lunghe 10 o 9.5. Tutte le righe descritte hanno il flag `t` da *text block*. Le due definizioni delle forme terminano con la definizione di fine forma (flag `e`, *end text block*) che impone la terminazione nel punto (1,17). Ho trovato le quantità adatte per approssimazioni successive a partire dai numeri trovati negli esempi allegati al pacchetto. Il tutto ha richiesto un paio d'ore tra lettura del manuale e prove numeriche. Tali quantità sono risultate adatte allo scopo e mi sono riservato di indagare sul perché in un secondo momento ancora di là da venire.

Questo caso molto semplice non ha avuto bisogno di ricorrere agli altri flag forniti dal pacchetto e necessari per gestire casi più complessi e articolati.

6 Posizionare gli elementi sulla pagina

L'ultimo problema da risolvere era quello di posizionare gli elementi sulla pagina nel giusto posto. Il pacchetto textpos (Gray, 2010) è *il* pacchetto. Serve a posizionare dei riquadri di testo virtuali anche in posizioni assolute sulla pagina ed è quanto mi serviva per mettere bene le colonne e il cruciverba.

Quando dichiariamo l'uso del pacchetto nel preambolo del documento dovremo specificare con l'opzione `absolute` la nostra intenzione di usare delle posizioni assolute, quindi dovremo specificare il passo delle coordinate. Io mi trovo bene con i centimetri, pertanto dopo lo `\usepackage` dichiarerò

```
\setlength{\TPHorizModule}{1cm}
\setlength{\TPVertModule}{\TPHorizModule}
```

Ora siamo pronti a posizionare qualunque cosa nel documento dando delle coordinate che corrisponderanno ai centimetri sulla pagina. Il punto della pagina corrispondente a (0,0) è quello in alto a sinistra.

Per iniziare la colonna delle definizioni orizzontali ho usato il seguente codice:

```
1  \begin{textblock}{6}(0,2)
```

3. Il pacchetto shapepar non chiede di specificare le unità di misura.

```
\begin{minipage}{6cm}
\shapepar{\obracketshape}
\hskip1.5em\textbf{ORIZZONTALI:}
\num{1}{Una bibita nera}
\num{8}{Austria e Belgio}
\num{9}{Per meglio dire}
```

e per finirla, invece:

```
\num{50}{Porto della Dancalia}
\num{51}{L'isola con i
   Faraglioni}
\num{52}{Punto cardinale}
\num{53}{Si sentono col naso}
\num{54}{Lo sostiene il
   candidato}
\ultnum{55}{Si devono ingoiare
   senza masticarle}
\end{minipage}\end{textblock}
```

Allo stesso modo, per iniziare la colonna delle definizioni verticali basta scrivere:

```
\begin{textblock}{6}(10,2)
\begin{minipage}{6cm}
\shapepar{\cbracketshape}
\hskip1.5em\textbf{VERTICALI:}
\num{1}{Agenti commerciali}
\num{2}{Iniz. di Boito}
\num{3}{Un condimento}
\num{4}{Sponde}
\num{5}{Un serpente}
\num{6}{Le vocali di tante}
\num{7}{Ricche e copiose}
```

mentre la fine è affidata a:

```
\num{50}{Aosta}
\num{51}{Cagliari}
\ultnum{52}{Congiunzione
   francese}
\end{minipage}\end{textblock}
```

Le due macro `\num` e `\ultnum` definite come

```
\newcommand{\num}[2]{\textbf
   {#1.} #2 -}
\newcommand{\ultnum}[2]{\textbf
   {#1.}~#2.}
```

servono solo a formattare le definizioni in base allo stile imposto dal settimanale.

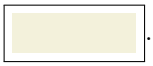
Mi sono accorto troppo tardi che la mancanza di '~' tra `#2` e `-` in `\num` consente il triste inconveniente di mandare a capo il trattino invece di ancorarlo in fin di riga.

7 Conclusioni

Realizzare un layout complesso con LaTeX può essere un'operazione disagevole. Ricorrere all'uso di pacchetti specifici può però semplificare alcuni compiti. Nell'articolo abbiamo discusso come realizzare un layout che riproduca in parte la prima pagina di un vecchio numero de La Settimana Enigmistica. Per far ciò abbiamo usato i pacchetti tikz, shapepar e textpos. L'articolo contiene anche una breve digressione dedicata al fotoritocco, specificamente alla pulizia di pagine rovinate dei documenti.

Epilogo

Naturalmente abbiamo consegnato il regalo al festeggiato. Lui lo ha tenuto in mano e sfogliato a lungo, visibilmente commosso. I suoi familiari, tra cui le uniche due persone che sapevano in anticipo del progetto, lo hanno giudicato il regalo più originale tra quelli ricevuti, senz'altro il più pensato e azzeccato.

Nella mia smania di ripulire le pagine per riottenere un aspetto "nuovo", infine, non ho considerato la possibilità di ricreare invece un aspetto anticato, ad esempio mettendo come sfondo un colore giallo ocra molto tenue, magari più marcato vicino ai bordi, come quello nel riquadro seguente:  .

Ringraziamenti

I miei ringraziamenti vanno obbligatoriamente all'editor e al correttore di bozze dell'articolo. I loro suggerimenti ne hanno aumentato sia la valenza "didattica" che la completezza. Se nel lettore permarranno dubbi sul contenuto, sarà stato per mio demerito che non ho saputo tradurre i suggerimenti in un contenuto fruibile.

Riferimenti bibliografici

Arseneau, D. (2006). *shapepar.sty*. URL `http://www.ctan.org`. Lo leggete dando `texdoc shapepar` al prompt del terminale o di DOS.

Gray, N. (2010). *Textpos: absolute positioning of text on the page*. URL `http://www.ctan.org`. Lo leggete dando `texdoc textpos` al prompt del terminale o di DOS.

La Settimana Enigmistica. «La settimana enigmistica.svg». URL `http://upload.wikimedia.org/wikipedia/commons/8/84/La_Settimana_Enigmistica.svg`.

Tantau, T. (2010). *The TikZ and* pgf *Packages*. URL `http://www.ctan.org`. Lo leggete dando `texdoc tikz` al prompt del terminale o di DOS.

▷ Gianluca Pignalberi
`g dot pignalberi at alice dot it`

Una panoramica su Pandoc

Massimiliano Dominici

Sommario

Questo articolo presenta una breve panoramica di Pandoc, un programma che consente di convertire testi formattati nel linguaggio Markdown in diversi formati di uscita, tra cui LaTeX e HTML.

Abstract

This paper is a short overview of Pandoc, an utility for the conversion of Markdown formatted texts in many output formats such as LaTeX and HTML.

1 Introduzione

Pandoc è un programma scritto in linguaggio Haskell che si propone come tramite tra alcuni linguaggi di markup leggero da una parte (in particolare Markdown) e alcuni formati 'finali' (tra cui LaTeX e HTML) dall'altra.[1] Sulla sua pagina web (MacFarlane, 2013) il programma è pubblicizzato come una sorta di "coltellino svizzero" per la conversione tra diversi formati e in effetti è in grado di leggere in input anche sorgenti scritti in una versione 'base' di LaTeX o HTML. Tuttavia, la sua utilità in questo senso si assottiglia di molto non appena il documento da convertire comincia a presentare una complessità interna non banale: per esempio, Pandoc non è in grado di tradurre in altri formati i comandi e gli ambienti LaTeX definiti dall'utente. Viceversa, si dimostra utile quando si deve realizzare un documento in diversi formati contemporaneamente, per esempio perché si sta scrivendo la documentazione di un programma e la si vuole distribuire per la stampa su carta (PDF via LaTeX), per la lettura su tablet o su uno dei vari lettori di eBook (ePub) e per la consultazione online (HTML). In simili frangenti, non è improbabile cominciare a scrivere il documento in uno dei formati finali (per esempio, e di solito, LaTeX) per accorgersi *solo alla fine* che convertirlo in altri formati comporta ostacoli più o meno significativi, poiché non sempre i formati di arrivo sono completamente compatibili con quello di partenza. È consigliabile, invece, scegliere come formato iniziale un linguaggio 'neutro' che non imponga le proprie idiosincrasie sui prodotti finali. Un buon candidato a questo ruolo di apripista è uno dei linguaggi di *markup leggero* esistenti: particolarmente interessante è Markdown, del quale Pandoc è un ottimo interprete.

1. Strettamente parlando, LaTeX non è un formato finale nello stesso senso in cui lo sono il PDF o i formati tipici di word processor come Microsoft Word o LibreOffice, per esempio. Tuttavia, e lo si vedrà nel corso della trattazione, è sottinteso che nell'ottica dell'utente di Pandoc, LaTeX rappresenta un prodotto finale o quanto meno intermedio.

2 Linguaggi di markup leggero: Markdown

Prima di entrare nel vivo dell'argomento, è bene spendere qualche parola sui linguaggi di markup 'leggero' (*lightweight markup*, nel gergo informatico). Essi si prefiggono l'esplicito obiettivo di minimizzare l'impatto delle istruzioni di marcatura all'interno del documento, mettendo l'accento sulla *leggibilità* del testo da parte di un *essere umano*, anche se quest'ultimo non dovesse conoscere le (poche) convenzioni su cui si basa il programma interprete per applicare al documento la formattazione voluta.

Sono due principalmente le aree in cui questi linguaggi trovano impiego: la documentazione di codice (reStructuredText, AsciiDoc, ecc.) e la gestione di contenuti web (Markdown, Textile, ecc.). Nel primo caso l'impiego è motivato dal fatto che la documentazione è contenuta nel codice stesso e deve essere leggibile con facilità da chi compulsa il sorgente, ma allo stesso tempo deve fornire la possibilità di essere presentata attraverso molti strumenti diversi (tradizionalmente PDF o HTML, ma ormai molti ambienti di sviluppo integrato includono funzioni di visualizzazione della documentazione interna). Nel secondo caso l'accento è spostato sulla facilità di immissione del testo da parte dell'utente. Molti *Content Management System* offrono dei plugin per usare l'uno o l'altro di tali linguaggi durante la fase di creazione dei contenuti. Lo stesso vale per i generatori di siti statici,[2] che forniscono un supporto nativo per almeno uno di essi e, spesso, la possibilità di usarne alternativamente altri. Anche i vari dialetti *wiki* possono essere considerati linguaggi di markup leggero.

2. I generatori di siti statici sono programmi che producono un sito web in HTML con contenuti predeterminati a partire da sorgenti che possono essere di vario genere. Le pagine sono generate in anticipo, normalmente su un computer locale, e poi caricate sul server per essere disponibili alle visite degli utenti. I siti così ottenuti somigliano molto ai vecchi siti scritti direttamente in HTML ma, a differenza di questi ultimi, nella fase di costruzione si possono usare template, condividere metadati tra diverse pagine, generare la struttura e il contenuto in maniera programmatica. I generatori di siti statici rappresentano un'alternativa ai più comuni generatori di siti web dinamici, le cui pagine sono generate solo al momento della richiesta da parte del visitatore.

L'effettivo grado di 'leggerezza' di questi linguaggi varia molto a seconda dello scopo in vista del quale essi sono stati concepiti. In generale, un linguaggio di markup leggero orientato alla documentazione di codice sarà più complesso e un po' meno leggibile di uno orientato alla creazione di contenuti per il web, che spesso non avrà la capacità di introdurre markup semantico.

Tra questi ultimi, Markdown nella sua versione originale è quello che si attiene in maniera più rigorosa all'approccio minimalista con cui i linguaggi di markup leggero sono stati concepiti. Il passo seguente è infatti indicativo delle intenzioni dell'autore, John Gruber, nel progettarlo:

> «Markdown è stato pensato per essere il più possibile facile da leggere e da scrivere. Tuttavia la leggibilità è l'elemento maggiormente enfatizzato. Un documento formattato con Markdown dovrebbe essere pubblicabile così com'è, in testo semplice, e non apparire disseminato di tag e istruzioni per la formattazione.»[3]

Inoltre, il formato finale di riferimento è HTML, tanto che Markdown accetta anche codice HTML grezzo. Gruber si è mantenuto sempre aderente a queste premesse iniziali e si è rifiutato di estendere il linguaggio oltre le specifiche iniziali. Questo ha portato a una proliferazione di varianti per cui quasi ogni singola implementazione presenta una versione 'migliorata' rispetto all'originale. Siti famosi come GitHub, reddit e Stack Overflow supportano ciascuno una propria versione di Markdown; lo stesso fanno programmi di conversione come MultiMarkdown e lo stesso Pandoc, che in più aggiungono nuovi formati di output. Non è il caso, in questa sede, di esaminare in dettaglio le diverse varianti; al lettore basterà avere una conoscenza di massima delle regole di formattazione basilari, così come sono riportate nelle tabelle 1 e 2. Per l'equivalente LaTeX, che non esiste nella versione originale di Markdown, le tabelle forniscono la traduzione più logica. Nel corso dell'articolo si vedrà come si comporta, in concreto, l'implementazione di Pandoc.

3 Una panoramica su Pandoc

Come detto in precedenza, Pandoc è innanzitutto un interprete di Markdown, che può trasformare in numerosi formati finali, tra cui HTML, LaTeX, ConTeXt, DocBook, ODF, OOXML, alcuni linguaggi di markup leggero come AsciiDoc, reStructuredText e Textile.[4] Pandoc offre anche limitate capacità di conversione di sorgenti LaTeX, HTML, DocBook, Textile e reStructuredText verso uno dei formati specificati sopra. Inoltre estende la sintassi di Markdown introducendo nuovi elementi e ampliando la configurabilità di quelli già presenti nell'originale.

3.1 Estensioni della sintassi di Markdown

L'insieme di elementi riconosciuti da Markdown è piuttosto limitato. Tabelle, note a piè di pagina, formule, citazioni e bibliografie non hanno nessun markup corrispettivo. Nelle intenzioni dell'autore, tutto ciò che oltrepassa le capacità del linguaggio dovrebbe essere espresso in HTML. Questo atteggiamento è ancora recepito da Pandoc (che permette anche l'uso di codice grezzo TeX solo per formati finali come LaTeX e ConTeXt, con una significativa eccezione che vedremo in seguito), tuttavia è reso in parte obsoleto, dato che con le estensioni introdotte da Pandoc l'utente ha la possibilità di usare la sintassi di Markdown per inserire questi elementi. Esaminiamo più da vicino queste estensioni.

Metadati

I metadati relativi a titolo, autore e data possono essere inseriti all'inizio del file in un blocco di testo, ciascuno preceduto dal carattere `%`, come si vede nell'esempio sottostante:

```
% Titolo
% Primo Autore; Secondo Autore
% 17/02/2013
```

Ogni elemento è opzionale e può quindi essere omesso, ma in questo caso deve essere lasciata in bianco la relativa riga (a meno che non si tratti dell'ultimo elemento, la data):

```
% Titolo
%
% 17/02/2013

%
% Primo Autore; Secondo Autore
% 17/02/2013

% Titolo
% Primo Autore; Secondo Autore
```

Note a piè di pagina

Dal momento che l'obiettivo principale di Markdown e dei suoi derivati è la leggibilità, il richiamo e il testo della nota vanno preferibilmente separati. È buona norma inserire quest'ultimo subito alla fine del capoverso cui la nota si riferisce, ma non è strettamente necessario: le note potrebbero trovarsi tutte insieme all'inizio o alla fine del documento, per esempio. Il richiamo consiste in un'etichetta arbitraria, racchiusa tra i caratteri `[^ ... ]`. Lo stesso richiamo, seguito dal carattere '`:`', dovrà precedere il testo della corrispettiva nota.

3. Gruber (2013), tradotto da `http://daringfireball.net/projects/markdown/syntax#philosophy`. Alla progettazione di Markdown ha dato un contributo essenziale anche Aaron Swartz.

4. L'elenco completo si trova in MacFarlane (2013).

Tabella 1: La sintassi di Markdown: elementi in linea.

Elemento	Markdown	LaTeX	HMTL
Collegamenti	`[link](http://esempio.net)`	`\href{link}{% http://esempio.net}`	`<a href="http://example.net/"> link</a>`
Enfasi	`_in evidenza_` `*in evidenza*`	`\emph{in evidenza}` `\emph{in evidenza}`	`<em>in evidenza</em>` `<em>in evidenza</em>`
Enfasi marcata	`__in evidenza__` `**in evidenza**`	`\textbf{in evidenza}` `\textbf{in evidenza}`	`<strong>in evidenza</strong>` `<strong>in evidenza</strong>`
Codice	`` `printf()` ``	`\verb|printf()|`	`<code>printf()</code>`
Immagini	`![Alt](/path/to/img.jpg)`	`\includegraphics{img}`	`<img src="/path/to/img.jpg" alt="Alt" />`

Tabella 2: La sintassi di Markdown: elementi a blocchi.

Elemento	Markdown	LaTeX	HMTL

Sezioni	`# Titolo #`
	`## Titolo ##`
	...
		`\section{Titolo}`
		`\subsection{Titolo}`
		...
			`<h1>Titolo</h1>`
			`<h2>Titolo</h2>`
			...

Testo citato	`> Questo capoverso`
	`> apparirà in`
	`> infratesto.`
		`\begin{quote}`
		`Questo capoverso`
		`apparirà in`
		`infratesto.`
		`\end{quote}`
			`<blockquote>`
			`<p>`
			`Questo capoverso`
			`apparirà in`
			`infratesto.`
			`</p>`
			`</blockquote>`

Elenchi puntati	`* Primo item`
	`* Secondo item`
	`* Terzo item`
		`\begin{itemize}`
		`\item Primo item`
		`\item Secondo item`
		`\item Terzo item`
		`\end{itemize}`
			`<ul>`
			`<li>Primo item</li>`
			`<li>Secondo item</li>`
			`<li>Terzo item</li>`
			`</ul>`

Elenchi numerati	`1. Primo item`
	`2. Secondo item`
	`3. Terzo item`
		`\begin{enumerate}`
		`\item Primo item`
		`\item Secondo item`
		`\item Terzo item`
		`\end{enumerate}`
			`<ol>`
			`<li>Primo item</li>`
			`<li>Secondo item</li>`
			`<li>Terzo item</li>`
			`</ol>`

Codice	`Capoverso di testo.`
	
	`    grep -i '£' file`
		`Capoverso di testo.`
		
		`\begin{verbatim}`
		`grep -i '£' file`
		`\end{verbatim}`
			`<p>Capoverso di testo.</p>`
			
			`<pre><code>`
			`grep -i '£' file`
			`</code></pre>`

Nel caso in cui la nota sia breve, è possibile inserirla direttamente nel testo, con una sintassi leggermente diversa. I due casi sono esemplificati qui di seguito:

```
Capoverso che contiene[^notalunga] una nota
troppo lunga per essere immersa direttamente
nel testo.

[^notalunga]: Nota troppo lunga per essere
immersa nel testo senza creare confusione.

Nuovo capoverso.^[Nota breve.]
```

Tabelle

Anche in questo caso la sintassi è determinata da criteri di leggibilità del testo. L'allineamento delle celle che formano la tabella può essere desunto dall'utente/lettore a partire dall'allineamento del testo rispetto alla linea tratteggiata che separa l'intestazione dalla tabella vera e propria e che deve essere *sempre* presente, anche quando l'intestazione è vuota.[5] Nel caso di tabelle con celle disposte su più righe, devono essere presenti anche una riga tratteggiata iniziale (a meno che l'intestazione sia vuota) e una finale. La larghezza relativa delle colonne viene presa in considerazione solo in presenza di tabelle multilinea. In ogni caso, non sono permesse celle che si estendono su più colonne o costruzioni che richiederebbero, in LaTeX, il pacchetto multirow. La didascalia, che può precedere o seguire la tabella, va preceduta da '`:`' (o, facoltativamente, da `Table:`) e va separata dal corpo della tabella da una riga bianca, come si mostra di seguito:

```
-----------------------------------------
     A destra    Centrato    A sinistra
------------- -------------- -------------
        Testo     Testo      Testo
    allineato   allineato    allineato
     a destra   al centro    a sinistra

  Nuova cella  Nuova cella   Nuova cella
-----------------------------------------

Table: Allineamenti
```

Una sintassi alternativa per le tabelle, e per specificare l'allineamento delle sue celle, prevede l'uso del carattere '`|`' per separare le colonne e del carattere '`:`' nella riga tratteggiata che segue l'intestazione. In questo caso non è necessario che il testo dell'intestazione e delle celle sia allineato, dal momento che a determinare l'allineamento è la disposizione dei vari ':', come si vede nell'esempio seguente.

```
A destra    | Centrato    | A sinistra
-----------:|:-----------:|:-------------
Testo       | Testo       | Testo
allineato   | allineato   | allineato
a destra    | al centro   | a sinistra
            |             |
Nuova cella | Nuova cella | Nuova cella

: Allineamenti tramite ':'
```

Nei casi precedenti, all'interno delle celle non è possibile inserire del materiale 'verticale' (capoversi multipli, blocchi di codice, elenchi). Per queste situazioni è necessario usare un tipo di tabelle cosiddetto 'a griglia'. Non è possibile specificare nessun allineamento, in aggiunta alle altre limitazioni che presentano gli altri due tipi. Ecco un esempio:

```
+-------------+-------------+----------------+
| Testo       | Elenchi     | Codice         |
+=============+=============+================+
| Capoverso.  | * Punto 1   | ~~~            |
|             | * Punto 2   | \def\PD{%      |
| Capoverso.  |             |  \emph{Pandoc} |
|             | * Punto 3   | ~~~            |
+-------------+-------------+----------------+
| Nuova cella | Nuova cella | Nuova cella    |
+-------------+-------------+----------------+
```

Figure

Markdown, come si può vedere dalla tabella 1, consente di inserire delle immagini con il codice

```
![Testo alternativo](/percorso/immagine)
```

dove il `testo alternativo` è la descrizione dell'immagine che il codice HTML fornisce nel caso in cui l'immagine non possa essere visualizzata. Pandoc aggiunge una sola estensione: l'immagine verrà trasformata in un oggetto '`figure`', completo di didascalia tratta dal `testo alternativo`, nel caso in cui questa appaia isolata in un capoverso.

Codici sorgente

Pandoc prevede la possibilità di aggiungere a un blocco di codice una serie di identificatori, classi e attributi la cui interpretazione varia a seconda del formato di output e delle opzioni passate al compilatore (in alcuni casi tali informazioni verranno semplicemente ignorate). Per fare ciò introduce una nuova sintassi alternativa per i blocchi di codice: invece di essere rappresentati da linee rientrate da quattro spazi, essi dovranno essere delimitati da sequenze di tre o più caratteri di tilde (`~~~`) o apostrofi rovesci (```). Nell'esempio seguente[6] vediamo un blocco di codice python a cui è stato associato, nell'ordine, un identificatore, la classe che lo segnala come codice python, un'ulteriore classe che impone di numerare le righe e infine un

5. In realtà, a determinare l'allineamento è l'intestazione, se presente, o la prima riga della tabella. Allineare il resto non è indispensabile ai fini della formattazione, ma aiuta il lettore.

6. Tratto dalla pagina web `http://wiki.python.org/moin/SimplePrograms`.

attributo per indicare il punto di partenza della numerazione.

```
~~~ {#bank .python .numberLines startFrom="5"}
class BankAccount(object):
    def __init__(self, initial_balance=0):
        self.balance = initial_balance
    def deposit(self, amount):
        self.balance += amount
    def withdraw(self, amount):
        self.balance -= amount
    def overdrawn(self):
        return self.balance < 0
my_account = BankAccount(15)
my_account.withdraw(5)
print my_account.balance
~~~
```

È possibile associare identificatori, classi e attributi anche a un frammento di codice inserito nel testo:

```
`printf`{.C} è una funzione di libreria che
stampa un output identificato.
```

Se non vengono specificate opzioni, Pandoc usa l'ambiente `verbatim` per i blocchi di codice senza evidenziazione e l'ambiente `Highlighting`, definito nel preambolo a partire dall'ambiente `Verbatim` del pacchetto fancyvrb, nel caso in cui le parole chiave siano evidenziate; l'evidenziazione è generata direttamente da Pandoc. L'opzione `--listings` fa sì che sia usato l'ambiente `lstlisting` in entrambi i casi.

Formule

Pandoc dispone di un buon supporto per le formule matematiche, che possono essere inserite tramite la comoda sintassi LaTeX. Le espressioni racchiuse tra due segni di dollaro verranno interpretate, come è naturale, come formule all'interno del testo; le espressioni racchiuse tra due doppi segni di dollaro verranno interpretate come formule fuori testo. Tutto ciò suona molto familiare all'utente di LaTeX.

Il modo in cui queste espressioni verranno rese dipende dai vari formati di output. Per l'output TeX (LaTeX/ConTeXt) le espressioni vengono passate tali e quali, tranne la sostituzione della coppia di delimitatori per le formule fuori testo (`$$...$$`) con `\[...\]`. Per HTML e formati derivati esistono diverse opzioni che possono essere passate al compilatore per determinare come verranno trattate le espressioni matematiche. Se non viene specificata nessuna opzione, Pandoc cercherà di renderle usando i caratteri disponibili tramite Unicode. Le altre opzioni prevedono la possibilità di usare alcuni tra i più diffusi metodi per visualizzare formule sul web tramite Javascript, in particolare MathJax (usato anche sul sito del GUIT), LaTeXMathML e jsMath. Oppure rendere le formule come immagini o, infine, usare MathML.[7]

Pandoc è anche in grado di interpretare delle semplici macro e espanderle in formati di output che non siano i due dialetti di TeX supportati. Questa funzionalità è comunque limitata alle sole formule matematiche.

Bibliografie

Pandoc può generare una bibliografia (e gestire le citazioni nel testo) a partire da un database bibliografico in uno dei formati riconosciuti dal programma (BibTeX, EndNote, ISI, ecc.). Se non si specificano opzioni (tranne il database bibliografico stesso, che deve essere obbligatoriamente indicato tramite l'opzione `--bibliography=FILE`), Pandoc inserisce i riferimenti e le voci bibliografiche come testo semplice, formattando le voci secondo lo stile bibliografico 'Chicago autore-data'. Oltre a poter scegliere un diverso formato per le voci bibliografiche, con l'opzione `--csl=FILE` il cui argomento deve essere un file di stile CSL,[8] l'utente può indicare al programma che vuole gestire la propria bibliografia, nel file `.tex`, mediante il pacchetto natbib o biblatex. In questo caso Pandoc non inserirà (nell'output LaTeX) citazioni e voci per esteso, ma i relativi comandi `\cite`, per i riferimenti nel testo, e quelli necessari a stampare la bibliografia. Le opzioni da passare per ottenere tale comportamento sono, rispettivamente, `--natbib` e `--biblatex`.

I riferimenti verranno inseriti nella forma `[@key1;@key2;...]` o `@key1` se si desidera che la citazione non sia racchiusa tra parentesi. Un trattino che precede l'etichetta fa sì che non venga stampato il nome dell'autore (quando il formato delle citazioni lo prevede). La bibliografia viene sempre inserita in fondo al documento.

Codice grezzo (HTML o TeX)

Tutte le implementazioni, canoniche e non, di Markdown accettano codice grezzo HTML, come è stato già segnalato nel paragrafo 2, che viene passato così com'è all'output HTML. Pandoc permette di inserire anche codice TeX grezzo che, anche in questo caso, verrà passato così com'è all'output LaTeX o ConTeXt e, naturalmente, ignorato negli altri output.

Template

Una delle funzionalità più interessanti di Pandoc è la possibilità di definirsi dei template persona-

7. Si veda rispettivamente `http://www.mathjax.org/`, `http://math.etsu.edu/LaTeXMathML/`, `http://www.math.union.edu/~dpvc/jsmath/` e `http://www.w3.org/Math/`.

8. CSL, (`http://citationstyles.org/`) *Citation Style Language*, è un formato aperto, basato su XML, per definire stili di citazione bibliografica. È impiegato in molti gestori di database bibliografici, come Zotero, Mendeley, Papers. Un elenco degli stili esistenti si trova a questo indirizzo web: `http://zotero.org/styles`.

lizzati per i vari formati di uscita. Per HTML e formati basati su TEX questo può avvenire in due modi diversi. Innanzitutto è sempre possibile generare solo il 'corpo' del documento e inserirlo in un 'master' tramite un apposito comando, che per LATEX sarà `\input` o `\include`. In questo modo il preambolo può essere costruito *ad hoc* per uno specifico progetto. Questo è, anzi, il comportamento di default di Pandoc, perché, per generare un documento HTML o TEX completo, bisogna specificare l'opzione `--standalone` o l'equivalente `-s`.

È anche possibile, però, costruire dei template più generici e flessibili, che possano tornare utili per più di un progetto, consentendo un moderato grado di personalizzazione. Come si può vedere nella figura 1, un template è sostanzialmente un file nel formato di uscita prescelto (nel nostro caso LATEX) al cui interno sono presenti variabili e strutture di controllo introdotte dal segno `$`. Queste espressioni verranno sostituite, al momento della generazione del file di uscita, con il testo appropriato. Per esempio, alla riga 111 del codice della figura 1 troviamo l'espressione `$body$` che verrà sostituita con il corpo del testo. Più sopra, alle righe 10–19 troviamo il codice che è responsabile dell'inserimento delle risorse necessarie a generare una bibliografia con natbib o biblatex. Questo codice verrà attivato solo nel caso in cui sia stata passata al compilatore l'opzione `--natbib` o `--biblatex` rispettivamente. Il codice per stampare materialmente la bibliografia si trova in fondo, alle righe 113–129.

Con questo sistema è possibile crearsi a piacimento variabili e relative opzioni da passare al compilatore. Possiamo quindi modificare il template predefinito per specificare, tra le opzioni da passare alla classe, non il solo corpo di riferimento del testo, ma una generica stringa che contenga tutte le opzioni che vogliamo.[9] Sostituiremo quindi la prima riga con la seguente:

```
\documentclass$if(clsopts)$[$clsopts$]$endif$%
  {article}
```

e compileremo, per esempio, così:

```
pandoc -s -t latex --template=mydefault \
  -V clsopts=a4paper,12pt -o test.tex test.md
```

avendo avuto cura di salvare nella cartella di lavoro il template modificato con il nome `mydefault.latex`.

4 Considerazioni finali

Pandoc possiede diverse funzionalità che lo rendono molto adatto a essere impiegato in un processo che prevede di ottenere uscite in diversi formati a partire da un unico sorgente. In particolare la possibilità di usare un database BIBTEX per generare automaticamente i riferimenti bibliografici e l'inserimento di espressioni matematiche 'come in LATEX', rappresentano due punti di forza notevoli per chi è abituato a usare LATEX.

Sfortunatamente, accanto a questi punti di forza sono presenti difetti e limitazioni che rendono il nostro programma meno attraente. Alcune di esse riguardano il tipo di linguaggio usato: Markdown, per esempio, non permette di usare il markup semantico.[10] Tali limitazioni possono essere anche aggirate tramite un ulteriore livello di astrazione, usando dei preprocessori, come `gpp` o `m4`. Una soluzione di questo tipo è proposta da Aditya Mahajan sul suo blog (MAHAJAN, 2012). Naturalmente ciò confligge un po' con la leggibilità del sorgente, oltre a introdurre maggiore complessità, sebbene l'uso di `m4` piuttosto che `gpp` consentirebbe di ridurre in parte la quantità di markup sovrabbondante.

Ma altri problemi sorgono al livello della conversione verso LATEX, in contesti in cui non ci si aspetterebbe di trovarli. Per esempio, il meccanismo di riferimenti incrociati è tarato sull'uscita in HTML e mostra tutte le sue lacune per quanto riguarda l'uscita LATEX. In sostanza, il riferimento viene generato mediante un'ancora ipertestuale e non attraverso il meccanismo usuale di `\label` e `\ref`. Per fare un esempio, considerando un titolo di sezione etichettato e richiamato in seguito nel testo, come nel seguente codice:

```
## Elementi di base ## {#Elbase}

[...]

Questo è un sottoparagrafo di
[Elementi di base](#Elbase)
```

il risultato è questo:

```
\hyperdef{}{Elbase}{%
  \subsection{Elementi di base}\label{Elbase}
}

[...]

Questo è un sottoparagrafo di
\hyperref[Elbase]{Elementi di base}
```

Non esattamente ciò che un utente di LATEX si aspetterebbe... Naturalmente è possibile usare direttamente i comandi `\label` e `\ref`, ma essi verranno ignorati in tutti i tipi di uscita non basati su TEX. Anche in questo caso si può usare

9. È evidente che questo può essere già fatto, 'abusando' della variabile `fontsize`.

10. Questa non è necessariamente una caratteristica inerente a tutti i linguaggi di markup leggero molti di essi forniscono un metodo per la definizione di oggetti che si comportano più o meno come una macro di LATEX. A volte (txt2tags) questo avviene tramite azioni di *pre-* o *post-processing*, a volte (Textile) è possibile 'abusare' di funzionalità pensate per scopi simili (la classe di uno `span` in HTML). In ogni caso l'impiego di un linguaggio di markup leggero tende a scoraggiare certe tecniche di etichettatura.

```
\documentclass$if(fontsize)$[$fontsize$]$endif${article}
\usepackage{amssymb,amsmath}
\usepackage{ifxetex}
\ifxetex
  \usepackage{fontspec,xltxtra,xunicode}
  \defaultfontfeatures{Mapping=tex-text,Scale=MatchLowercase}
\else
    \usepackage[utf8]{inputenc}
\fi
$if(natbib)$
\usepackage{natbib}
\bibliographystyle{plainnat}
$endif$
$if(biblatex)$
\usepackage{biblatex}
$if(biblio-files)$
\bibliography{$biblio-files$}
$endif$
$endif$
$if(lhs)$
\usepackage{listings}
\lstnewenvironment{code}{\lstset{language=Haskell,basicstyle=\small\ttfamily}}{}
$endif$
$if(verbatim-in-note)$
\usepackage{fancyvrb}
$endif$
```

[...]

```
$for(include-before)$
$include-before$

$endfor$
$if(toc)$
\tableofcontents

$endif$
$body$

$if(natbib)$
$if(biblio-files)$
$if(biblio-title)$
$if(book-class)$
\renewcommand\bibname{$biblio-title$}
$else$
\renewcommand\refname{$biblio-title$}
$endif$
$endif$
\bibliography{$biblio-files$}

$endif$
$endif$
$if(biblatex)$
\printbibliography$if(biblio-title)$[title=$biblio-title$]$endif$

$endif$
$for(include-after)$
$include-after$

$endfor$
\end{document}
```

Figura 1: Frammento del template predefinito di Pandoc per LaTeX.

un preprocessore per ottenere due distinti output intermedi a partire dai quali generare l'uscita in HTML e LaTeX (e magari un terzo formato intermedio per ODF/OOXML, ecc.). Ma, di nuovo, questo rende meno immediato l'uso di Pandoc per il tipo di contesto che abbiamo esaminato in questo articolo.

Anche le espressioni matematiche pongono dei problemi. Pandoc distingue solo tra formule nel testo e formule fuori testo. Queste ultime vengono sempre tradotte come ambienti `displaymath` generici. Non è possibile indicare ambienti alternativi, come `equation`, `gather`, ecc., se non attraverso i sistemi visti in precedenza in questo paragrafo, con tutti gli inconvenienti già indicati.

Va anche sottolineato che Pandoc è un programma in continuo sviluppo e che molte funzionalità presenti oggi non lo erano fino a poco tempo fa (per esempio la gestione delle bibliografie); non è quindi impensabile che tutte o alcune delle mancanze che un utente LaTeX può trovare in Pandoc in questo momento vengano risolte in un futuro più o meno prossimo. Entro certi limiti, inoltre, è possibile integrare o modificare il comportamento di base di Pandoc mediante degli script, come segnalato nella documentazione (`http://johnmacfarlane.net/pandoc/scripting.html`). Nel mio caso, un grosso impedimento all'implementazione di simili script è dato dal fatto che il linguaggio in cui Pandoc è scritto è Haskell.

A conclusione di questa panoramica, continuo personalmente a considerare Pandoc la scelta migliore per un generico progetto che preveda una molteplicità di formati in uscita. L'uso di un formato di testo originale 'neutro' rispetto ai formati di uscita rende più facile evitare le idiosincrasie di un particolare linguaggio, a cui si devono, di solito, eventuali problemi in fase di traduzione verso formati diversi. Tuttavia sarebbe ingenuo credere che tutte le difficoltà trovino in questo modo facile soluzione; limiti inerenti ai linguaggi di markup leggero e difetti di implementazione fanno sì che si debbano comunque inventare delle tecniche che aggirino tali inconvenienti. Queste tecniche, ovviamente, rendono più complessa l'operazione. Ciò non toglie che, se il progetto intrapreso consente di accettare tali limitazioni, Pandoc rende estremamente semplice ottenere molti formati di uscita diversi a partire da un singolo sorgente.

Riferimenti bibliografici

Gruber, J. (2013). «Markdown». `http://daringfireball.net/projects/markdown/`.

Kielhorn, A. (2011). «Multi-target publishing». *TUGboat*, **32** (3), pp. 272–277.

MacFarlane, J. (2013). «*Pandoc*: a universal document converter». `http://johnmacfarlane.net/pandoc/`.

Mahajan, A. (2012). «How I stopped worring and started using Markdown like TeX». `http://randomdeterminism.wordpress.com/2012/06/01/how-i-stopped-worring-and-started-using-markdown-like-tex/`.

Wikipedia (2013). «Lightweight markup language». `http://en.wikipedia.org/wiki/Lightweight_markup_language`.

▷ Massimiliano Dominici
`mlgdominici at gmail dot com`

LaTeX e le lingue romanze alpine: il friulano

Claudio Beccari

Sommario

LaTeX può gestire molte lingue, attualmente 74 più diverse varianti. Alcune delle lingue gestibili da LaTeX sono usate da poche persone, altre da milioni. Questo articolo vorrebbe essere un altro passo verso l'estensione di LaTeX alle lingue romanze usate nelle regioni alpine d'Europa. In questo secondo articolo sull'argomento si parla del friulano, parlato, letto e scritto da quasi 800 mila persone nel Nord Est d'Italia e nelle comunità friulane sparse per il mondo.

Abstract

LaTeX is used to typeset in a variety of languages: at the time of writing, it can handle 74 different languages (plus many variants), some of them used by very few people, some by millions. At the moment, there are no facilities for typesetting documents in the various more or less official Alpine Romance languages. This paper would be another step in order to extend the LaTeX language capabilities to the various romance languages that are being used by large communities in the European Alps. In this second article on the subject we deal with Friulan, spoken, read, an written by almost 800 thousand people in North-eastern Italy and within the Friulian communities around the world.

Somari

LaTeX a pues condusi cetantis lenghis, in dì di vuê 74 plui diversis variants. Cualchidune di chestis lenghis a son feveladis di pocjis personis, altris di milions.

Chest articul al volerès jessi un altri pas viers la estension de LaTeX a lis lenghis romanics in vore inta lis regions alpinis de Europe. In chest secont articul sul'argument si tabae dal furlan, fevelât, let e scrit di scuasit votcent mil personis intal Nord Est de Italie e inta lis comunitâts furlanis spandudis intal mont.

1 Introduzione

In un precedente articolo, (Beccari, 2012), si è parlato dell'estensione di LaTeX alla gestione del romancio svizzero. In questo secondo articolo sull'argomento si parlerà del friulano (furlan).

Il friulano è parlato da un numero di persone molto maggiore di coloro che si esprimono quotidianamente in romancio; ma a differenza della Svizzera, l'Italia non lo ha (ancora) riconosciuto come lingua ufficiale nazionale. Sia chiaro, non è vero che l'Italia non presti attenzione alle lingue di importanza regionale e alle lingue minoritarie in generale, ma le sole lingue ufficialmente riconosciute sono il francese e il tedesco, ma anche queste con una valenza prevalentemente regionale; la prima per la Valle d'Aosta, la seconda per la provincia autonoma di Bolzano.

Per l'intera Italia l'articolo 6 della Costituzione specifica che "la Repubblica tutela con apposite norme le minoranze linguistiche" e la legge 482/1999, intitolata "Norme in materia di tutela delle minoranze linguistiche storiche" prevede misure di tutela e valorizzazione che devono essere poi attuate da apposite disposizioni regionali o provinciali. L'Italia ha anche ratificato nel 2000 la "Carta Europea delle lingue regionali e minoritarie" che il Consiglio d'Europa ha prodotto nel 1992. Per quel che riguarda la Regione Autonoma Friuli-Venezia Giulia, questa ha emesso la legge regionale 15/1996 e con il decreto del presidente della Giunta Regionale del novembre 1996 è stata stabilita un Commissione regionale per la definizione di una "ortografia" standardizzata per tutte le varietà di friulano parlate nella regione; le conclusioni di questa Commissione sono state approvate nel 1998 e aggiunte alla legge 15/1996 sotto forma di emendamento.

In questo articolo, dunque, si esporranno i criteri con i quali si è esteso l'uso di LaTeX alla lingua friulana nella sua espressione scritta standardizzata secondo la legge citata. Si sottolinea qui che le indicazioni della Commissione sopra menzionata sono molto specifiche per l'ortografia, anche delle parole importate da lingue antiche (latino e greco) o dalle lingue dei popoli vicini o dalle lingue maggioritarie degli stati di cui la regione di lingua madre friulana ha fatto parte nell'arco dei secoli. Il rapporto della Commissione parla anche di alcune varianti ortografiche tollerate per la scrittura di certi suoni presenti soltanto in alcune varietà della lingua. Ma quel rapporto non dice una parola sulla divisione in sillabe; non è un caso che tutto quanto si trova scritto in documenti scaricabili dalla rete sia sempre privo di qualunque cesura in fin di riga.

Grazie al fatto che il friulano appartiene al gruppo delle lingue romanze, non è stato difficile definire delle regole di sillabazione; comunque è stato necessario rivolgersi a linguisti friulani per dirimere la questione della "s impura" che viene trattata in modo diverso dai vari standard nazionali o minoritari delle varie lingue romanze. Per l'italiano

esiste una specifica norma UNI; così per il francese, lo spagnolo e il portoghese; per il romancio l'argomento è stato trattato nella Grammatica Rumantscha, (Caduff *et al.*, 2009) con dovizia di particolari. Ma per le altre grafie ladine non si è trovato nulla in merito alla divisione *grammaticale*, anche se si è trovato qualcosa per la divisione *fonetica*, (Carrozzo, 2009; Roseano, 2010).

2 Fonemi e grafemi del friulano

Il friulano si scrive con l'alfabeto latino con l'aggiunta della lettera 'ç' e con l'uso degli accenti grave e circonflesso su alcune vocali. L'insieme di fonemi e grafemi è mostrato nella tabella 1, con i fonemi rappresentati mediante i caratteri IPA (International Phonetic Alphabet)[1].

Come si vede la grafia del friulano non è strettamente fonetica, diversi suoni sono rappresentati da un solo grafema, oppure lo stesso suono può essere rappresentato da grafemi diversi. La tastiera italiana è perfettamente adatta per rappresentare tutti i grafemi usati, tranne che per le vocali marcate con l'accento circonflesso; per questo scopo sarebbe opportuno astenersi dalla scrittura standard TEX `\^a, \^e,...` perché la presenza del comando per il circonflesso potrebbe mettere in crisi un eventuale correttore ortografico (ne esistono anche per il friulano); si segnala però che sono disponibili per le macchine Windows dei driver di tastiera che con pochi tasti permettono di scrivere direttamente le lettere segnate con l'accento circonflesso; con le macchine Linux e Mac, non dovrebbero esserci problemi di sorta.

Ma la presenza di accenti e di lettere non strettamente ASCII a 7 bit, richiede di usare una codifica di ingresso ben specificata, e si consiglia la codifica `utf8`; queste lettere marcate con diacritici, però, impongono assolutamente di usare per i font di uscita la codifica `T1`, quando si usa pdfLATEX; con XELATEX e i font OpenType non ci si deve preoccupare di nulla.

Piuttosto si noti che la tabella 1 è divisa in tre parti; nella prima sono mostrati i fonemi e i grafemi corrispondenti e vi si nota quanto detto sopra a proposito della mancanza di unicità fra pronuncia e scrittura; per altro questo succede anche con l'italiano, quindi non si tratta di una cosa così strana. Nella seconda parte sono elencati i grafemi e i fonemi biunivoci; nella terza parte sono indicati alcuni grafemi consentiti per esprimere certi suoni presenti in alcune varietà del friulano; il loro uso non fa parte della standardizzazione ufficiale, ma è suggerito per distinguere le pronunce delle varietà in quegli scritti dove si voglia evidenziare la natura locale della corrispondente parlata.

1. Con pdfLATEX i fonemi IPA sono accessibili mediante il pacchetto `tipa`, (Fukui, 2004) fornito di una discreta documentazione. Con XELATEX e LuaLATEX, che usano i caratteri OpenType, i grafemi sono tutti direttamente accessibili, sempre che il font di testo scelto come "main font" ne sia dotato.

Tabella 1: Corrispondenza fra fonemi e grafemi

Fonemi IPA	Grafemi
a, aː	a, â
e, eː	e, ê
i, iː	i, î
o, ɔ, oː, ɔː	o, ô
u, uː	u, û
aɪ	ai
aje, ae	aie
ajo, ao	aio
aju, au	aiu
oje, oe	oie
ja	ja, ia
je	je, ie
jo	jo, io
ju	ju, iu
aʊ, eʊ, iʊ, oʊ	au, eu, iu, ou
aj, ej, oj, uj	ai, ei, oi, ui
wa, we, wi, wo	ua, ue, ui, uo
k	c, ch
tʃ	c, ç
t͡ɕ	cj
kw	cu, qu
g	g, gh
dʒ, ts, dz	z, ç
d͡ʑ	gj
ɲ	gn
s	s, ss
z	ʼs, s
b	b
d	d
f	f
l	l
m	m
n	n
p	p
r	r
v	v
ʃ	sj, ssj
ʒ	sj
θ	th
ð	dh

3 Come LATEX gestisce le lingue

LATEX, come anche gli altri mark-up del sistema TEX, da XELATEX, a ConTEXt a LuaTEX, gestisce le lingue in due modi distinti. Se ne è già abbondantemente parlato nell'articolo precedente, (Beccari, 2012), e qui non si ripete altro che l'essenziale.

Ognuno di quei motori di composizione ha bisogno di tre informazioni essenziali e distinte: (*a*) dell'elenco delle parole da comporre in lingua, per

esempio nelle date o nei titoli; (*b*) delle regole di sillabazione; (*c*) di altre informazioni tipografiche connesse con la tipografia o le consuetudini tipografiche di ciascuna lingua o della stessa lingua in paesi diversi.

Il primo gruppo richiede l'uso corretto dei nomi dei mesi, e le giuste preposizioni, congiunzioni e punteggiatura per la scrittura delle date e le parole, per esempio "Capitolo", "Chapître", "Cjapitul", ..., da inserire nelle pagine iniziali dei capitoli prima del titolo.

La seconda classe di informazioni richiede che siano espresse in forma particolare le regole di sillabazione; Frank Liang, (Liang, 1983), collaboratore di Donald E. Knuth ai tempi della nascita di TEX, ha creato un algoritmo particolarmente veloce e versatile per eseguire la divisione in sillabe basandosi sui *pattern*; questi sono monconi di parole contenenti le indicazioni se fra certe due lettere, eventualmente inserite in un moncone di parola più lungo di due lettere, si può eseguire una cesura o se è vietato. Questa operazione può anche essere fatta a mano; l'algoritmo di Liang è utile solo se si dispone di un elenco di molte migliaia di parole già divisi e in sillabe; esso è adatto alle lingue per le quali le regole sono particolarmente complesse o non esistono affatto. La difficoltà è quella di disporre o di creare il suddetto elenco di parole già divise in sillabe, ma la creazione di un tale elenco è troppo complicata da fare a mano se si deve partire da zero. Io preferivo realizzare a mano l'elenco dei pattern, senza ricorrere all'algoritmo di Liang; l'intelligenza umana riesce, almeno per le lingue romanze, a creare insiemi di pattern un poco più semplici di quelli che può produrre l'algoritmo di Liang, anche se questo algoritmo è fatto molto bene.

Disponendo dei pattern si evita di disporre di un intero dizionario con tutte le parole possibili divise in sillabe, ma si concentrano le informazioni un pezzetto di parola alla volta. In italiano e in friulano occorrono 348 e 414 pattern rispettivamente; per l'inglese ne occorrono quasi 5000; per altre lingue si può arrivare anche oltre le 14 200 unità. Questi grossi numeri di pattern devono essere scritti da chi li prepara (una persona oppure l'algoritmo di Liang) in un modo ordinato facile da leggere per una persona, ma poi devono essere riordinati nella memoria della macchina in modo che siano accessibili con la massima velocità per non rallentare inutilmente il processo di composizione. Per questo è necessario che i pattern di tutte le lingue che si vogliono usare siano predigeriti dal motore di composizione durante la creazione del file di formato, un file speciale in linguaggio macchina che contiene oltre ai pattern tutte le macro definite nel nucleo di LATEX o di qualunque altro tipo di markup.

Le distribuzioni moderne del sistema TEX possono essere installate nelle macchine di ciascun utente con livelli di completezza diversi; se si installa una versione completa, non ci si deve preoccupare se i pattern della lingua che interessa usare siano stati già precaricati, perché la versione completa li carica tutti nel formato. Se un utente preferisce risparmiare spazio su disco e installa una versione incompleta, ma solo di base o di completezza media, può succedere che i pattern della lingua che si vuole usare non siano stati precaricati; questo implica che la sillabazione mancante venga sostituita con quella di default (quella dell'inglese americano) che molto probabilmente produrrà nel documento molte cesure in fin di riga completamente errate. Non si raccomanda mai abbastanza di installare la versione completa, anche perché, sebbene non sia difficile, caricare i pattern di una determinata lingua richiede di fare cose e usare programmi che abitualmente anche un utente esperto del sistema TEX non fa o non usa.

Il terzo gruppo di informazioni riguarda per esempio le penalità per le righe orfane (la prima riga di un capoverso dopo la quale si ha un salto di pagina) e le righe vedove (l'ultima riga di un capoverso prima della quale è avvenuto un salto di pagina); è chiaro che queste righe orfane o vedove sono antiestetiche in un testo ben composto e quindi sono da evitare. Nello stesso modo possono venire specificati dei *demeriti* per evitare che la penultima riga di un capoverso subisca la cesura in fin di riga, per evitare che l'ultima riga cominci con un frammento di parola e, a limite, possa contenere solo l'ultima sillaba dell'ultima parola del capoverso: estremamente sgradevole.

Va anche notato che le regole di sillabazione codificate con i pattern devono evidentemente rispettare la grammatica, ma in nessun caso devono eseguire la cesura in fin di riga lasciandosi alle spalle una sola sillaba troppo corta o mandando a capo una sola sillaba troppo corta; in generale la minima lunghezza del primo moncone non deve contenere meno di due lettere e quella dell'ultimo moncone meno di tre lettere. Per l'italiano e il friulano questi numeri sono invece entrambi uguali a due lettere; questo implica però che la parola 'idee', che grammaticalmente si potrebbe dividere in 'i-de-e', non può venire divisa da nessuna programma di composizione del sistema TEX, perché le sillabe iniziale e finale sono troppo corte[2].

Merita segnalare che con pdfLATEX il pacchetto che si occupa di gestire le lingue è babel, mentre con XELATEX e con LuaLATEX è polyglossia; me le tre funzioni precedentemente descritte sono pra-

2. La cesura impedita dalla prima sillaba di una sola lettera riappare, per esempio in italiano, quando la parola è preceduta dall'articolo o dalla preposizione articolata con l'elisione della vocale: 'del-l'i-dea'; quando si compone con lingue che usano l'elisione vocalica e impiegano l'apostrofo per marcarla, il programma viene istruito in modo da trattare l'apostrofo alla stessa stregua di una lettera dell'alfabeto.

ticamente le stesse; sono le altre funzioni offerte dei due pacchetti che possono essere assai diverse, ma è anche vero che il problema della scelta dei font con cui scrivere in lingue che non usano l'alfabeto latino sono gestite meglio con gli ultimi due programmi e con polyglossia.

4 File di descrizione della lingua e file di pattern

La funzione descrittiva della lingua esposta nella sezione precedente è svolta (generalmente) da un solo file con estensione `.ldf`. Invece i pattern possono richiedere diversi file: diversi perché devono essere usati solo con font aventi codifiche del tipo T1, oppure con codifiche UNICODE; c'è modo di costruire file che soddisfino entrambe le esigenze, ma la predisposizione dei pattern è piuttosto delicata.

4.1 Il file di descrizione della lingua

I file `.ldf` per babel hanno il nome formato con quello della lingua e con l'estensione indicata. I file per polyglossia hanno il nome formato agglutinando il nome della lingua con il prefisso `gloss-`; il nome della lingua è solitamente il nome in inglese e in lettere minuscole; per il friulano si hanno perciò i file `friulan.ldf` e `gloss-friulan.ldf`; per maggiore generalità si sono preparati anche i file `furlan.ldf` e `gloss-furlan.ldf`.

La funzione del file `.ldf` nei due casi è sostanzialmente la stessa, ma mentre con babel la lingua è una opzione per il pacchetto, con polyglossia questo è un file di estensione autonomo che permette di definire alcuni comandi con i quali precisare le singole lingue da usare nel documento specificandone, mediante opzioni, anche alcune particolarità che sono disponibili grazie all'uso dei font OpenType. Non si scenderà in questi dettagli; si descriverà il file `friulan.ldf` che contiene le definizioni importanti e assolutamente necessarie in ogni file di questo genere.

```
1  \LdfInit\CurrentOption{captions\CurrentOption}
2  \ifx\l@friulan\@undefined
3    \ifx\l@furlan\@undefined
4          \@nopatterns{friulan}
5          \adddialect\l@friulan\l@italian
6    \else
7          \let\l@friulian\l@furlan
8    \fi
9
10 \fi
11 \expandafter\ifx\csname l@\CurrentOption\endcsname
       \relax
12   \expandafter\let\csname l@\CurrentOption
       \endcsname\l@friulan
13 \fi
14 \@namedef{captions\CurrentOption}{%
15     \def\prefacename{Prefazion}%
16     \def\refname{Riferiments}%
17     \def\abstractname{Somari}%
18     \def\bibname{Bibliografie}%
19     \def\chaptername{Cjapitul}%
20     \def\appendixname{Zonte}%
21     \def\contentsname{Tabele gjener\^al}%
22     \def\listfigurename{Liste des figuris}%
23     \def\listtablename{Liste des tabelis}%
24     \def\indexname{Tabele analitiche}%
25     \def\figurename{Figure}%
26     \def\tablename{Tabele}%
27     \def\partname{Part}%
28     \def\enclname{Zonte(is)}%
29     \def\ccname{Cun copie a}%
30     \def\headtoname{Par}%
31     \def\pagename{Pagjine}%
32     \def\seename{cjale}%
33     \def\alsoname{cjale ancje}%
34     \def\proofname{Dimostrazion}%
35     \def\glossaryname{Glossari}%
36 }
37 \@namedef{date\CurrentOption}{%
38     \def\today{\number\day\space di\space\ifcase
       \month\or
39       Gen\^ar\or Fevr\^ar\or Mar\c{c}\or Avril\or
       Mai\or Jugn\or
40       Lui\or Avost\or Setembar\or Otobar\or
       Novembar\or Dicembar%
41       \fi\space dal\space\number\year}}
42 \providehyphenmins{\CurrentOption}{\tw@\tw@}
43 \expandafter\addto\csname extras\CurrentOption
       \endcsname{%
44   \babel@savevariable\clubpenalty
45   \babel@savevariable\widowpenalty
46   \babel@savevariable\@clubpenalty
47   \clubpenalty3000\widowpenalty3000\@clubpenalty
       \clubpenalty}%
48 \expandafter\addto\csname extras\CurrentOption
       \endcsname{%
49   \babel@savevariable\finalhyphendemerits
50   \finalhyphendemerits50000000}%
51 \expandafter\addto\csname extras\CurrentOption
       \endcsname{%
52   \lccode`'=`'}%
53 \expandafter\addto\csname noextras\CurrentOption
       \endcsname{%
54   \lccode`'=0}%
55 \ldf@finish\CurrentOption
56 \endinput
```

Alcuni commenti sono importanti. Innanzi tutto nell'intero file la lingua "friulan" non viene mai menzionata esplicitamente, ma è sostituita dalla macro `\CurrentOption` che contiene il nome "friulan" o "furlan" a seconda di quale nome si sia specificato come opzione per babel; nel seguito si eseguirà la sostituzione in modo da non complicare inutilmente la spiegazione; infatti la prima linea di codice serve per l'inizializzazione, ma fra le altre cose verifica che per "friulan" sia stato definito `\captionsfriulan`; ciò serve a babel tra l'altro per controllare se il file sia già stato caricato.

Le righe da 2 a 8 verificano se esista nel file di formato un insieme di pattern di sillabazione collegato alla lingua friulana; se un tale insieme non esistesse, sarebbe necessario comunque dare un significato al comando `\l@friulan` associandolo all'insieme di un'altra lingua come se ne fosse una varietà. Si è preferito associarlo all'insieme dell'italiano, perché io lavoro con distribuzioni complete del sistema TEX; se questo file fosse usato in una distribuzione

di base, non è detto che l'insieme collegato all'italiano sia stato generato, e quindi una associazione come quella che si è fatta qui potrebbe dare origine ad errori; per evitarli sarebbe necessario eseguire altri test; data la provvisorietà di questa descrizione della lingua friulana si è preferito omettere questo lievissimo perfezionamento.

Le successive righe dalla 14 alla 36 definiscono tutte le parole fisse che compaiono nei titolini o in certi ambienti; si sono presi questi nomi dal vocabolario bilingue friulano–italiano, (Cescje, 2011) messo a disposizione on line dal Centri Friûl Lenghe; e dal vocabolario monolingue (Vicario, 2010) reperibile nel sito della Società Filologica Friulana. Si noti che in queste parole, come nei successivi nomi dei mesi, il file non contiene altro che caratteri ascii a 7 bit, per cui esso non dipende da nessuna codifica particolare di input. Nei corrispondenti file `gloss-*.ldf`, invece, quelle parole sono state scritte con le lettere accentate in codifica utf8, perché quei file vengono letti solo con polyglossia da programmi che lavorano con la codifica utf8 per costruzione.

La data in friulano è definita nelle righe da 37 a 41; come si vede la data friulana non solo tratta i nomi dei mesi come nomi propri, ma viene anche formata con l'ausilio di preposizioni semplici e articolate.

Le righe da 42 a 50 impostano alcuni parametri per la sillabazione; la sillaba iniziale e quella finale non devono contenere meno di due caratteri; siccome moltissime parole friulane terminano con gruppi di consonanti, tocca ai pattern provvedere a che l'ultima sillaba contenga almeno una vocale.

Inoltre sono impostate le penalità per le righe vedove e le righe orfane; il valore 3000 è abbastanza alto ma non infinito, per cui le righe orfane e vedove potranno essere presenti ma *pdflatex* cercherà di evitarle con sufficiente determinazione. Il valore enorme assegnato a `\finalhyphendemerits` serve per evitare che la penultima riga di un capoverso subisca la cesura in fin di riga; in questo modo l'ultima riga di un capoverso non comincia (quasi) mai con una frazione di parola.

Infine le righe da 51 a 54 impostano e disimpostano la condizione per la quale l'apostrofo possa essere considerato come parte di una parola in friulano e non lo sia più quando si cambia lingua. Le ultime due righe completano le impostazioni per il friulano e finiscono il file; dopo `\endinput` potrebbe esserci scritta qualunque cosa, ma questa parte del file non verrebbe nemmeno letta dal programma di composizione.

4.2 I file per la sillabazione

I file per generare le strutture interne per la sillabazione sono tre: `loadhyph-fu.tex`, `hyph-quote-fu.tex` e `hyph-fu.tex`; `fu` *non* è la sigla del friulano secondo le norme iso, perché questa sigla è `fur` ma, nei limiti del possibile, il sistema TEX preferisce usare sigle di due sole lettere.

Il primo file carica gli altri due quando si devono creare o ricreare i file di formato, al momento di generare questi file, non durante la composizione di un particolare testo. Il secondo contiene le impostazioni per la codifica dell'apostrofo; infine il terzo file contiene i pattern veri e propri.

Il metodo che noi abbiamo seguito per creare i pattern non si affida al programma *patgen* creato da Frank Liang, (Liang, 1983), e distribuito contestualmente con il sistema TEX, ma si affida alla traduzione nello speciale linguaggio (descritto da Knuth nel suo manuale *The TEXbook*, (Knuth, 1996)) delle regole grammaticali valide per il friulano; queste regole, certamente simili a quelle dell'italiano, sono documentate nel testo di Carrozzo, (Carrozzo, 2009), nel Vocabolari furlan, (Vicario, 2010), e nella versione stampata del Grande Dizionario Bilingue, (Cescje, 2011). Nel testo di Carrozzo si sottolinea che una stessa parola può comportare un numero diverso di sillabe a seconda che sia intonata in modo "normale", oppure enfatico, oppure poetico oppure sia cantata; in quel testo l'autore tratta la divisione secondo la pronuncia che comporti il massimo numero di sillabe.

Per fortuna la sillabazione tipografica ha certe sue regole più restrittive sia delle regole grammaticali sia di quelle fonetiche, per cui non siamo stati sfiorati da questa questione sul differente numero di sillabe a seconda dell'intonazione: semplicemente abbiamo evitato le divisioni di vocali che *potrebbero formare* dittonghi o trittonghi. È noto, infatti, che il lettore si trova a disagio se, trovando una cesura in fin di riga, trova nella riga successiva un moncone di parola che comincia con una vocale[3]; perciò non non si sono mai divisi gli iati, sia quelli veri sia quelli derivanti dall'intonazione; l'approccio usato è: "meglio una divisione mancata che una divisione sbagliata".

Grazie all'aiuto del dott. Paolo Roseano abbiamo però chiarito la questione della divisione della 's impura', che in friulano appartiene sempre alla sillaba precedente, mentre la doppia 'ss' forma un digrafo e quindi non viene divisa fra due sillabe. In friulano non compaiono altre consonanti doppie tranne quando si compongono parole con prefissi terminanti con una consonante; apparentemente questo succede solo con la 'n'.

Per il resto le regole che abbiamo seguito sono le seguenti:

- Ogni sillaba contiene una vocale o un dittongo o un trittongo; in friulano le vocali possono essere accentate; i dittonghi e i trittonghi sono gli stessi che si trovano in italiano. Con lo stes-

3. Eccettuate le semivocali 'i' e 'u', che marcano l'inizio di un dittongo o di un trittongo; per esempio *maieutiche* viene divisa in 'ma-ieu-ti-che'.

so concetto dei pattern per l'italiano abbiamo scritto i pattern per il friulano evitando di esplicitare le vocali, cosicché, eventualmente, non vengono divisi alcuni iati, ma non vengono commessi errori dividendo gruppi vocalici dove non è consentito.

- Ogni consonante semplice e ogni digrafo seguito da una vocale o da un dittongo fa sillaba con la vocale che segue. I digrafi che corrispondono a "suoni semplici" sono: *ch, cj, gh, gj, gn, qu, ss.*

- Abbiamo deciso che due consonanti diverse *che non formino un digrafo*, si dividono fra le sillabe adiacenti se la coppia di consonanti non può trovarsi all'inizio di una parola friulana. Di fatto abbiamo seguito le indicazioni di Carrozzo, (Carrozzo, 2009, pag. 5). Questo implica tra l'altro che le consonanti liquide 'l' ed 'r' vengono separate dalle consonanti che le seguono, ma non da quelle che le precedono. La regola complessiva di questo punto è, *mutatis mutandis*, sostanzialmente uguale a quella per l'italiano; solo la 's impura' e la doppia 's' vengono trattate in modo opposto all'italiano.

- È vietato andare a capo dopo l'apostrofo, esattamente come in italiano.

Con queste regole si sono creati i pattern per il friulano semplicemente prendendo i pattern per l'italiano, che già facevano abbastanza bene il loro lavoro perché rispettano regole abbastanza simili a quelle del friulano, aggiungendo poi i pattern specifici per il friulano, correggendo eventualmente quelli dell'italiano, quando erano in contrasto con le regole enunciate sopra. I pattern totali sono saliti a 414, 66 in più di quelli per l'italiano. Rigenerati i file di formato, sulla mia macchina ora posso scrivere in friulano senza problemi. La lingua friulana, con i suoi pattern e i file di descrizione della lingua sono parte integrante della distribuzione TeX Live 2012 (e della distribuzione MacTeX 2012); al momento di scrivere non sono in grado di dire se sia già stata integrata nella distribuzione MiKTeX.

Ecco ora un breve testo in friulano tratto dalla presentazione del testo deliberato dalla Giunta Regionale del Friuli–Venezia Giulia, (ARLeF):

> Al è dal sigûr impuartant che i Furlans a conservin lis lôr fevelis locâls, lis lôr variantis, intai rapuarts informâi o pe espression poetiche, ma ancjemò plui impuartant al è che a imparin a doprâ la lenghe comune soredut intai rapuarts formâi, intal scrivi, pe prose e pai messaçs che a àn par distinataris ducj i Furlans.[4]

5 Conclusione

Ho imparato molto da questo progetto; ho approfondito la mia conoscenza delle lingue romanze e ho scoperto realtà che prima mi erano assolutamente sconosciute. Anche in questo caso, come in quello dell'articolo precedente, (Beccari, 2012), al di là delle questioni specifiche della lingua friulana o romancia, si vede bene quale sia il processo di gestione delle lingue gestito dai pacchetti babel e polyglossia.

6 Ringraziamenti

Voglio ringraziare la Società Filologica Friulana, che mi ha gentilmente messo in contatto con il dott. Paolo Roseano, che ringrazio con molto calore; egli mi ha chiarito diversi punti in merito alla sillabazione fonetica del friulano oltre a consentirmi di consultare diversi suoi lavori, in particolare (Roseano, 2010). Ringrazio anche Massimo Furlani che mi ha dato lo spunto per lavorare su questo tema. Ringrazio molto anche Tommaso Gordini che, grazie ad una sua amica friulana (carnica), mi ha corretto il sommario in friulano che avevo provato a tradurre io stesso (con diversi errori e scelte inadeguate di alcune parole) facendo l'esercizio di traduzione come ai tempi del liceo.

Riferimenti bibliografici

ARLeF (2002). *La grafie uficiâl de lenghe furlane*. Udine. URL `http://www.friul.net/lenghe/Grafie%20ULF%20-%202002.pdf`.

Beccari, C. (2012). «LaTeX e le lingue romanze alpine». *ArsTEXnica*.

Braams, J. (2011). *Babel, a multilingual package for use with LaTeX's standard document classes*. TUG. Leggibile con `texdoc babel` nella distribuzione TeX Live.

Caduff, R., Caprez, U. N. e Darms, G. (2009). *Grammatica per l'instrucziun dal rumantsch grischun*. Seminari da rumantsch da l'Universitad da Friburg. Dipartament da lingugatgs e litteraturas romanas, Friburg, CH. URL `http://www.unifr.ch/rheto/documents/Gramminstr.pdf`.

Carrozzo, S. (2009). «Gramatiche fonetiche furlane». URL `http://www.cfl2000.net/download/Gramatiche_Fonetiche.pdf`.

4. È sicuramente importante che i Friulani conservino le loro parlate locali, le loro varianti, nei rapporti informali e nell'espressione poetica, ma ancor più importante è che imparino ad usare la lingua comune soprattutto nei rapporti formali, nello scrivere in prosa e per i messaggi destinati a tutti i Friulani.

Cescje, A. (a cura di) (2011). *Grant dizionari bilengâl Talian – Furlan*. Centri Friûl Lenghe 2000. URL `http://www.cfl2000.net/`.

Charette, F. (2011). *Polyglossia: a Babel replacement for XeLATEX*. TUG. Leggibile con `texdoc polyglossia` nella distribuzione TeX Live. L'attuale versione è affidata alla manutenzione di Arthur Reutenauer.

Fukui, R. (2004). *TIPA manual*. Graduate School of Humanities and Sociology, Tokyo University, Tokyo. Leggibile con `texdoc tipa` nella distribuzione TeX Live.

Knuth, D. E. (1996). *The TEXbook*. Addison Wesley, Reading, Mass., 16ª edizione.

Liang, F. M. (1983). *Word Hy-phen-a-tion by Com-puter*. Tesi di Dottorato, Department of Computer Science, Stanford, CA. URL `www.tug.org/docs/liang/liang-thesis.pdf`.

Roseano, P. (2010). «La division in silabis dai sufis -sion e -zion tal Grant Dizionari Bilengâl Talian-Furlan». *Sot la nape*, **LXI** (4), pp. 51–60.

Vicario, F. (a cura di) (2010). *Vocabolari furlan*. Società Filologica Friulana. URL `http://www.filologicafriulana.it/easynet/Archivi/SFF/Files/Vocabolari_Furlan_1_2010.pdf`.

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

Le filigrane e le figure di sfondo

Claudio Beccari

Sommario

Il sistema TEX offre diversi pacchetti per inserire nei documenti filigrane o immagini di sfondo, ma ognuno di questi pacchetti ha le sue caratteristiche particolari che lo rendono utile in certe applicazioni più che in altre. In questo articolo si vuole esaminare a fondo il modo di introdurre queste decorazioni per comprenderne meglio il meccanismo.

Abstract

Our TEX system has several packages available for inserting watermarks or background images. But each one of these packages has specific features that make it more suitable in certain applications rather than others. In this article the mechanism for inserting such "decorations" is thoroughly examined so as to fully understand how it works.

1 Introduzione

Le filigrane e le figure di sfondo hanno qualcosa in comune, sono "disegnate" sotto il testo; in altre parole, ci si scrive sopra. È vero che le immagini di sfondo solitamente impegnano tutta la pagina, mentre le filigrane di solito rimangono all'interno della griglia di stampa, ma queste sono differenze non sostanziali, mitigate dal "di solito" che fa sì che esistano immagini di sfondo che non occupano l'intera pagina oppure filigrane che escono dalla griglia di stampa.

La differenza sostanziale è che le figure di sfondo sono spesso, per non dire sempre, costituite da immagini a colori vivi, più o meno saturi; sopra i quali bisogna scrivere mediante tinte contrastanti con font colorati affinché il testo sia distinguibile dallo sfondo. Le filigrane sono immagini o parole composte con colori poco saturi ossia con una elevata componente di bianco e sulle quali il testo nero risalta senza interferenze.

Le filigrane sono adatte sia per mettere un'immagine di tonalità fortemente attenuata al di sotto del testo di una presentazione, per esempio composta con la classe beamer, sia per porre trasversalmente alla pagina scritte di fondo del tipo "BOZZA", "DRAFT", "RISERVATO", "CLASSIFIED". Queste appariranno sotto il testo in modo da fornire informazioni relative alla natura del documento e rendendo la loro eliminazione impossibile per semplice fotocopiatura.

Le immagini di sfondo vanno bene per le copertine dei documenti; le filigrane possono andare bene per qualsiasi pagina interna di qualsiasi documento.

In ogni caso le une o le altre non debbono venire usate solo perché LATEX consente di farlo.

In questo articolo parlerò essenzialmente di filigrane, anche se molte delle cose che esporrò potranno essere valide anche per le immagini di sfondo.

Ho deciso di scrivere questo articolo stimolato da una discussione sul Forum GUIT; chiedeva infatti un partecipante: "Ciao, vorrei inserire la scritta DRAFT in diagonale su ciascun foglio del documento." Questa discussione si è poi sviluppata con diversi interventi.

Ora, senza scendere nell'analisi del codice di questo o quel pacchetto, vorrei esporre che cosa c'è dietro a queste operazioni.

2 Breve rassegna dei pacchetti esistenti

Ovviamente la soluzione più semplice consiste nell'impiegare i pacchetti già predisposti per questo compito, come eso-pic (Niepraschk, 2010) e watermark (Rozhenko, 2004). Anzi, watermark è fatto proprio per questo scopo, tanto che il suo uso è talmente facile quanto disarmante:

```
\usepackage{watermark}
\watermak{DRAFT}
```

e da qui in poi il pacchetto fa tutto da solo. Il pacchetto eso-pic richiede forse un paio di righe in più. Secondo la mia impressione eso-pic è più adatto per le immagini di sfondo, mentre watermark è esplicitamente dedicato alle filigrane.

Il risultato può essere ottenuto anche con l'uso del pacchetto TikZ (Tantau, 2010), diventato di fatto il software di prima scelta per eseguire qualsiasi operazione grafica su qualunque documento. Claudio Fiandrino[1] ha fornito la seguente soluzione[2]:

```
\documentclass{article}
\usepackage{tikzpagenodes}
\usetikzlibrary{calc}
\usepackage[contents={}]{background}

\usepackage{lipsum}% for dummy text
```

1. Collaboratore molto competente del Forum e autore di alcune Guide Tematiche sull'impiego di programmi che utilizzano pesantemente la grafica

2. Il colore e l'opacità della filigrana sono stati modificati per poter disporre di un'immagine in tonalità di grigio, come così pure il comando per l'inserimento in calce del nome dell'autore (estraneo al tema di questo articolo).

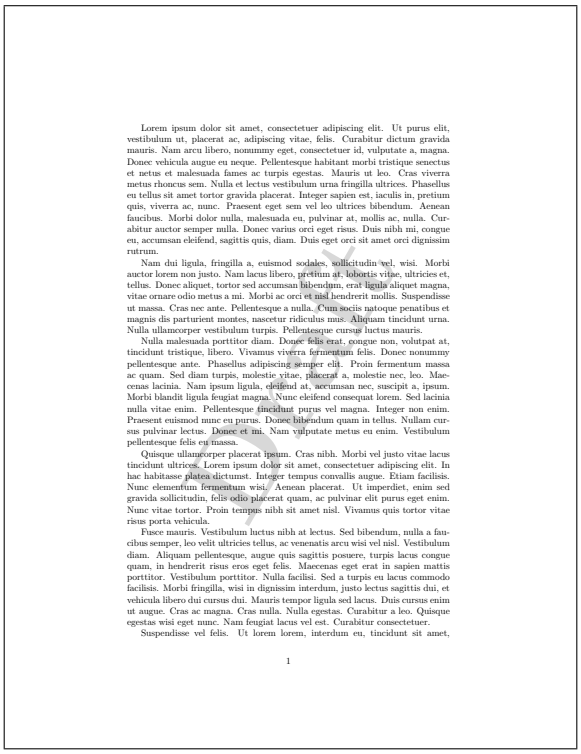


Figura 1: Risultato del codice di Claudio Fiandrino mediante l'uso del pacchetto TikZ

```
7
8  \newcommand{\draftlogo}[1][darkgray,rotate=60]{%
9  \tikz[remember picture,overlay]{%
10    \node[scale=5,font=\bfseries\Huge,
11      opacity=0.2,#1] at (current page.center)
12     {Draft};
13   }%
14 }
15
16 \AddEverypageHook{%
17 \draftlogo
18 \BgMaterial}
19
20 \begin{document}
21 \lipsum[1-8]
22 \end{document}
```

Per ottenere il risultato, mostrato nella figura 1, il codice dell'esempio deve essere compilato due volte.

Leggendo il codice ed osservando la figura si comprende come la filigrana venga centrata nella pagina e non nel centro della griglia di stampa, e come venga ottenuta l'inclinazione di 60° della filigrana stessa. Inoltre, avendo impiegato la classe article, lo specchio di stampa è quasi centrato rispetto al foglio di carta (in questo caso di formato *letter*) facendo apparire la filigrana sostanzialmente al centro dello specchio di stampa. Componendo fronte retro la cosa potrebbe non funzionare al meglio, ma comunque le modifiche da apportare sarebbero poche. Non c'è dubbio che TikZ consente di fare operazioni grafiche molto elaborate con pochi comandi relativamente semplici, ma l'impostazione del risultato non è proprio banale; a confronto, l'uso del pacchetto watermark è un giochetto da ragazzi.

Concretamente, mi riferisco qui alle condizioni per un "concorso" scherzoso che ho bandito sul forum GUIT:

- creare il codice per inserire una filigrana usando solo le potenzialità del codice di LaTeX e del motore di composizione con cui gira il programma *pdflatex*;
- è consentito usare (ma non è obbligatorio farlo) il pacchetto pict2e, che estende le funzionalità del linguaggio originale di LaTeX secondo le linee indicate da Leslie Lamport stesso nella seconda edizione del suo manuale (Lamport, 1994);
- è consentito usare (ma non è obbligatorio farlo) il pacchetto graphicx (Carlisle, 2005).

Nello sviluppare il codice che soddisfi ai requisiti del "concorso" ci si rende conto dei problemi da affrontare e dei "trucchi" a cui bisogna ricorrere. Ma, più importante di tutti, ci si rende conto del funzionamento del meccanismo per inserire le filigrane e, *mutatis mutandis*, per inserire le immagini di sfondo.

Aggiungerò alcune altre condizioni:

- la filigrana deve essere collocata al centro dello specchio di stampa, non al centro della pagina;
- l'inclinazione della filigrana deve essere pari a quella della diagonale dello specchio di stampa; è consentita sia una soluzione manuale sia una soluzione automatica;
- deve essere possibile specificare a piacere la grandezza e il colore della filigrana; per questo scopo è accettabile usare almeno il pacchetto color (Carlisle, 2005) o, meglio ancora, il pacchetto xcolor (Kern, 2007); supponendo che il codice da scrivere venga conservato in un file `filigrana.sty`, diventa possibile ricorrere a comandi interni di LaTeX che sono formati con nomi che contengono il carattere `@`;
- ogni personalizzazione deve poter essere eseguita dopo il caricamento del file `filigrana.sty`.

3 Premessa operativa

La filigrana deve essere inserita sul supporto di stampa, sia questo lo schermo o un foglio di carta, prima della collocazione del testo sulla medesima pagina. La filigrana altrimenti coprirebbe il testo rendendolo, in mancanza di colori trasparenti, illeggibile.

Quindi per essere sicuri dove la filigrana va inserita bisogna conoscere almeno l'essenziale del funzionamento della routine di uscita di LaTeX, quella

che spedisce una pagina completamente composta accodandola al file di uscita.

Questa routine è poco documentata nelle distribuzioni del sistema TEX ma l'essenza di questa procedura, al fine di quello che ci interessa qui, è sapere che tale routine pone come prima cosa sulla pagina in costruzione la testatina, poi il blocco del testo, accompagnato dalle eventuali note marginali e dagli altri oggetti mobili (come figure e tabelle) che trovano posto nella pagina, ed infine mette il piedino. Completata questa costruzione la pagina viene accodata al file di uscita. Possiamo dunque immaginare che ciò che va sulla pagina sia costituito da tre strati, il più profondo dei quali contiene la testatina, quello intermedio contiene il corpo del testo e quello più in superficie contiene il piedino.

Ogni filigrana deve stare quindi sotto lo strato del testo, quindi deve essere inserita insieme alla testatina. Questo è un buon posto, anche perché la testatina ha una posizione ben fissa sulla pagina e di solito non subisce alterazioni se la classe del documento e le personalizzazioni inserite dall'utente non violano qualche regola.

Tuttavia, anche se collocata al medesimo livello di profondità, la filigrana non fa parte della testatina nel senso che non giace sulla riga dove solitamente si trovano le informazioni che compaiono nella testatina. Inoltre esistono degli stili di pagina privi di testatina (per esempio gli stili "empty" e "plain").

In effetti in quest'ultimo caso le testatine sono ancora presenti ma vuote. Il problema iniziale sembra invece più difficoltoso: apparentemente ogni comando di marcatura di LATEX che produce del testo occupa spazio sul supporto di stampa. In realtà ci sono due comandi `\rlap` (*right overlap*, ricopri a destra) e `\llap` (*left overlap*, ricopri a sinistra) raramente usati ma disponibili. Esiste anche il comando `\makebox` a livello utente (Beccari, 2012) che produce una scatola di larghezza specificata, eventualmente nulla, dove il testo può uscire tranquillamente dalla scatola se più largo della larghezza specificata. Tuttavia questi comandi, pur non occupando spazio orizzontale, o una parte inferiore a quella richiesta dalla larghezza naturale del loro contenuto, occupano sempre un'altezza e una profondità corrispondenti a quelle del loro contenuto.

Tuttavia i comandi originali di marcatura di LATEX contenevano l'ambiente picture il quale nella versione originale del 1985 (LATEX 209) era veramente molto limitato, ma ha perso virtualmente ogni limitazione (nel senso che Leslie Lamport stesso ha definito nella seconda edizione del suo manuale come avrebbe dovuto estendersi) con la versione del 1994 (LATEX). L'ambiente è stato "esteso" solo nel 2003 mediante il pacchetto pict2e (Gässlein *et al.*, 2011) e questo pacchetto ha poi subito successive estensioni, l'ultima delle quali risale al 2011. Dopo tali estensioni, l'ambiente picture non ha virtualmente nessuna limitazione, anche se molte delle cose che si possono fare semplicemente con TikZ richiedono invece un grosso lavoro con picture.

Per questo motivo l'ambiente picture, esteso con il suo pacchetto pict2e, non è molto conosciuto ma per cose semplici come le filigrane è facilissimo da usare; anzi, non è nemmeno necessaria l'estensione pict2e. I pacchetti eso-pic e watermark si appoggiano infatti all'ambiente picture, e così farò anche qui.

Il pregio fondamentale è che l'ambiente picture consente di definire un disegno eseguito su una tela di dimensioni nulle e grazie ai suoi comandi di posizionamento consente di collocare qualunque cosa fuori dall'area del disegno senza bisogno di ricorrere ad alcun artificio specifico.

Ecco dunque che abbiamo trovato come mettere nelle testatine un oggetto di dimensioni nulle, dal quale "fuoriescono" altri oggetti, immagini, scritte o filigrane, collocabili ovunque senza alterare il testo.

4 Soluzione manuale

Creiamo un file `filigrana.sty` che contenga tutte le definizioni che occorrono. Cominciamo dunque a descrivere una prima soluzione manuale al problema della nostra filigrana. Abbiamo bisogno di macro per l'utente per specificare dimensioni e colori della filigrana e per specificare l'angolo di inclinazione della retta sulla quale essa sembra "infilzata":

```
% Pacchetti su cui appoggiarsi
\usepackage{graphicx}
\usepackage{xcolor}
\usepackage{pict2e}[2009/06/01]
% Impostazioni predefinite:
\def\Draftname{Draft}\def\Draftangle{55}
\newlength{\Draftheight}\setlength\Draftheight{43
    mm}
\definecolor{Draftcolor}{gray}{0.8}
% Comandi per l'utente
\newcommand\AltezzaBOZZA[1]{\setlength\Draftheight
    {#1}}
\newcommand\BOZZA[1]{\gdef\Draftname{#1}}
\newcommand\AngoloBOZZA[1]{\xdef\Draftangle{#1}}
\newcommand\ColoreBOZZA[3]{\definecolor
    {#1}{#2}{#3}}
```

Poi occorre una macro per comporre la filigrana nell'ambiente picture di dimensioni nulle:

```
\newcommand\B@ZZA{%
\begin{picture}(0,0)\unitlength\p@
\put(0,-\strip@pt\headsep){%
\put(-\strip@pt\dimexpr\textwidth/2\relax,
-\strip@pt\dimexpr\textheight/2\relax){%
\makebox(0,0){\normalfont
\fontsize{\Draftheight}{\Draftheight}\selectfont
\rotatebox{\Draftangle}{\color{Draftcolor}
    \Draftname}}}
}\end{picture}}
```

Questa macro contiene un certo numero di "sporchi trucchi", tipo quelli descritti da Knuth nel suo

TEXbook, (Knuth, 1996). In realtà l'unico trucco che non è descritto nel manuale di Lamport è l'uso del comando `\strip@pt` definito nel nucleo di LaTeX; esso agisce sul valore esplicito di una dimensione e gli "strappa" via le unità di misura 'pt'. La 'dimensione' può anche essere una espressione dimensionale e il moderno *pdftex* (il motore di composizione di LaTeX) dispone di questi comandi dimensionali dal 2005, quindi se non sono tanto conosciuti è solo dovuto all'inerzia di molti utenti, che preferiscono usare il pacchetto calc (Krab Thorup *et al.*, 2007) che fa esattamente le stesse cose ricorrendo a medesimi comandi. Di solito usare calc o `\dimexpr` non comporta alcun risparmio di tempo o di spazio nello scrivere il file di macro. Usare il comando primitivo del linguaggio ε-TeX (Breitenlohner, 1998) consente un risparmio di tempo rispetto a medesimi calcoli eseguiti con calc ma stiamo parlando di microsecondi, quindi di limitata importanza. La sequenza di controllo `\p@` indica una lunghezza di 1 pt mentre il comando `\fontsize` è disponibile anche per l'utente per specificare un corpo particolare del font e il suo corrispondente avanzamento di riga (LaTeX3 Project Team, 2005). In questo caso scrivendo una sola parola non dobbiamo preoccuparci dell'interlinea.

Tuttavia questo modo di procedere nasconde un'esigenza ben particolare. La famiglia di font su cui operare deve essere continuamente scalabile, altrimenti il gioco non funziona. Tra i font predefiniti distribuiti con il sistema TEX ci sono moltissimi font di tipo Type 1 continuamente scalabili[3] ma fra quelli che si rifanno ai font originali del sistema TEX, come i Computer Modern, i CM-super e altri, nessuno è continuamente scalabile a meno che non venga caricato l'uno o l'altro dei pacchetti type1cm (codifica OT1) oppure type1ec (codifica T1) che, eliminando la scelta dei corpi ottici[4], consentono lo scalamento continuo. Agire in questo modo sarebbe poco funzionale perché esistono i font Latin Modern, molto simili ai font Computer Modern, che, a differenza di questi, sono continuamente scalabili pur conservando i corpi ottici.

Il codice precedente mostra come il comando `\B@ZZA`, che contiene il disegno di dimensioni nulle sia in larghezza sia in altezza e profondità, può essere messo all'estremità destra delle testatine sia di destra che di sinistra. Il primo comando `\put` sposta in basso il riferimento esattamente di `\headsep`, quindi lo abbassa allo spigolo superiore destro dello specchio di stampa; il successivo `\put` pone il suo contenuto in basso e a sinistra di mezza altezza e di mezza larghezza dello specchio di stampa, quindi esattamente al centro dello specchio. Il contenuto di questo comando `\put`, a sua volta, è una scatola di dimensioni nulle il cui contenuto viene esattamente centrato in altezza e in larghezza. Questo contenuto, la lettura è chiara, è costituito dal testo specificato con `\Draftname`, colorato come specificato dal colore `Draftcolor`, composto con il font di corpo pari a `\Draftheight` e ruotato dell'angolo `\Draftangle`.

Quindi ora non abbiamo più problemi con la filigrana, ma dobbiamo mettere il comando che la produce dentro le testatine con i vari stili di pagina. Se l'utente usasse il pacchetto fancyhdr, l'impostazione delle testatine sarebbe più semplice, ma pensando ad un file di macro personali va benissimo ricorrere a comandi di basso livello:

```
\let\ori@ps@empty\ps@empty
\let\ori@ps@plain\ps@plain
\let\ori@ps@headings\ps@headings

\renewcommand\ps@plain{\ori@ps@plain
\def\@oddhead{\null\hfill\B@ZZA}%
\let\@evenhead\@oddfoot}

\renewcommand\ps@empty{\ori@ps@empty
\def\@oddhead{\null\hfill\B@ZZA}%
\let\@evenhead\@oddfoot}

\renewcommand\ps@headings{\ori@ps@headings
\toks@=\expandafter{\@oddhead}%
\edef\@oddhead{\the\toks@\noexpand\B@ZZA}%
\toks@=\expandafter{\@evenhead}
\edef\@evenhead{\the\toks@\noexpand\B@ZZA}}
```

Ho esplicitato solo i tre stili di pagina più frequentemente usati; dentro il nucleo di LaTeX e nei file di classe questi stili sono identificati dal loro nome aggiunto al prefisso `\ps@`. Basta quindi memorizzare gli stili originali mediante dei comandi "sinonimi" (gli stessi nomi con il prefisso `\ori@ps@`) e poi ridefinirli in modo che contengano le testatine con l'aggiunta del comando `\B@ZZA` all'estremità destra.

Per gli stili `empty` e `plain` basta aggiungere le definizioni opportune. Per lo stile `headings` (e lo stesso bisognerebbe fare per ogni altro stile che contenga qualcosa nelle testatine) bisogna ricorrere allo "sporco trucco" di memorizzare la definizione (il testo sostitutivo) delle definizioni delle testatine in un registro "token", il registro numero 'zero', che dispone di un nome particolare `\toks@`. Quindi ridefinire ciascun comando per le testatine pari o dispari espandendo il registro token ma senza espandere il comando `\B@ZZA`. Lo "sporco trucco" può essere semplicemente evitato usando il pacchetto etoolbox che definisce i comandi `\preto` e `\addto` per aggiungere del testo a quello contenuto nella definizione di una macro rispettivamente

3. I font di tipo Type 1 sono font vettoriali ed hanno il vantaggio che possono venire ingranditi, distorti, stirati in mille modi, ma comunque resi in modo impeccabile sia su schermo, sia a stampa.

4. Uno dei vantaggi dei fonti originali del sistema TEX è che sono fra le poche collezioni di font che dispongono di una vasta serie di corpi ottici, cioè di font affini nello stile grafico, ma disegnate in modo tale che i corpi più piccoli non perdano i loro dettagli più minuti mentre i corpi più grandi non risultino troppo neri. Questo è un enorme vantaggio, specialmente quando si devono usare molti apici e pedici come in matematica.

prima o dopo il testo preesistente. Ma qui non lo facciamo perché le condizioni che ci siamo dati non consentono di usare comandi di altri pacchetti, se non di quelli specificamente elencati.

Infine, affinché le modifiche agli stili di pagina abbiano luogo, bisogna eseguire almeno il comando per lo stile di default:

```
\pagestyle{headings}
```

Ora basta creare un file di prova che carichi il file `filigrana.sty` oltre agli altri pacchetti necessari.

```
\documentclass[a4paper,12pt]{book}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage{lmodern}
\usepackage[italian]{babel}
%\usepackage{canoniclayout}
%\usepackage{layaureo}
\usepackage{filigrana}
\usepackage{lipsum}
\begin{document}\Huge

\chapter{Prima pagina in stile plain}
    \thispagestyle{plain}
Le pagine successive sono in stile headings.

\lipsum[1-5]

\lipsum[6-20]

\chapter{Pagine classificate}
\BOZZA{CLASSIFIED}

\lipsum[5-25]


\end{document}
```

Salvato questo file, ad esempio come `BOZZA.tex`, si può vedere che cosa succede con il layout di pagina della classe book oppure con il layout che si ottiene con il pacchetto canoniclayout oppure con il pacchetto layaureo. Si noterà ad occhio nudo che la filigrana ha bisogno di un aggiustamento per l'inclinazione perché le diagonali degli specchi di stampa nei tre casi sono inclinate in modo diverso (vedi più avanti la figura 7).

L'inclinazione non è tuttavia un problema, basta misurare l'altezza e la larghezza di uno specchio di stampa in una pagina sullo schermo o stampata su carta. L'angolo da introdurre mediante il comando `\AngoloBOZZA` sarà l'arcotangente in gradi sessagesimali del quoziente altezza diviso larghezza. Per molti lettori è dai tempi delle scuole secondarie che non capita di calcolare un'arcotangente, ma oggi ogni calcolatrice elettronica tascabile ha il tasto per eseguire questo calcolo. In mancanza di tale strumento c'è sempre l'applicazione/programma *Calculator* in ogni computer, che consente di eseguire il calcolo senza alzarsi dalla scrivania. D'altra parte questo calcoletto è da fare una sola volta per l'intero documento, non è una cosa da ripetere in ogni pagina, per cui il "disturbo" di doverlo calcolare a mano è veramente trascurabile. Il risultato di questo codice, con l'angolo corretto è mostrato nella figura 3.

5 Prima soluzione automatica

È possibile lasciare fare i calcoli al programma *pdftex*? Ebbene sì, è possibile ma con qualche acrobazia. Infatti, tranne il modulo per eseguire calcoli del pacchetto TikZ, non ho trovato nessun altro pacchetto che sia in grado di calcolare l'arcotangente precedentemente eseguita a mano.

Non possiamo chiamare il modulo dei calcoli di TikZ, perché sarebbe vietato dalle condizioni del concorso. D'altra parte non abbiamo bisogno di fare una grande elaborazione per calcolare l'arcotangente di un angolo che sicuramente giace nel primo quadrante; al massimo potremmo calcolare l'arcocotangente (cioè l'arcotangente dell'angolo complementare) per poter mantenere i calcoli con una precisione decente. Inoltre la procedura di TikZ fa riferimento ad una tabella di un migliaio di valori per calcolare l'angolo di cui è data la tangente; usa metodi di interpolazione che ritengo siano inutilmente complicati per replicarli qui in forma semplificata.

Il concorso ha messo in luce l'inventiva dei partecipanti. In particolare Roberto Giacomelli ha fornito una soluzione basata sull'uso delle funzioni disponibili per eseguire calcoli in virgola mobile contenuti nella libreria `l3fp.sty` che accompagna la distribuzione del linguaggio sperimentale messo a punto dal LaTeX 3 Team. Questa libreria sperimentale, già stabile, fa parte di ogni distribuzione aggiornata del sistema TeX in quanto parte integrante di XeLaTeX e ad essa molti pacchetti si appoggiano sistematicamente; non sempre fanno parte di pdfLaTeX, ma ora cominciano ad essere presenti diversi pacchetti basati su questo linguaggio anche per pdfLaTeX.

Non riporto il codice della soluzione trovata da Giacomelli, che comunque è reperibile nel filone di discussione citato a proposito del concorso, ma voglio esporne i principi, molto interessanti ed efficaci.

Premetto che la libreria `l3fp.sty` definisce già una serie di funzioni tra le quali quelle trigonometriche seno, coseno e tangente, oltre alla radice quadrata di cui parleremo più avanti. Non contiene le definizioni delle funzioni trigonometriche inverse, quindi anche con quella libreria bisogna arrangiarsi.

Giacomelli ha usato il metodo delle tangenti, è un semplice metodo iterativo che viene usato spessissimo per risolvere equazioni per le quali non è disponibile una formula risolutiva esplicita. Si parte da un valore approssimato della soluzione e con un formula iterativa, che illustrerò fra poco, si migliora l'approssimazione. Ovviamente la procedura iterativa dell'algoritmo può essere applicata solo un numero finito di volte. I motori di composi-

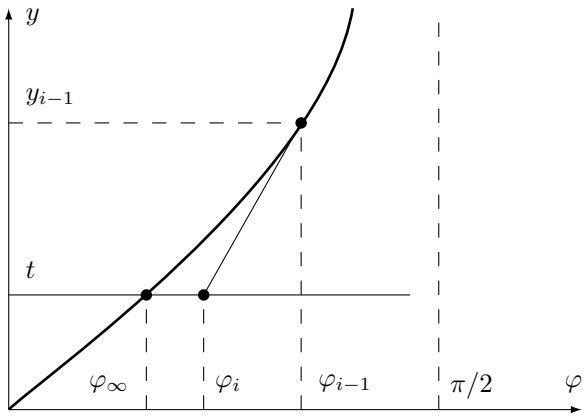


Figura 2: Metodo delle tangenti

zione del TEX lavorano a virgola fissa e dispongono di 16 cifre binarie fratte, corrispondenti a circa 5 cifre decimali fratte. Quindi è perfettamente inutile iterare l'algoritmo delle tangenti oltre a questo limite per cercare di ottenere valori approssimati migliori: sarebbe impossibile. Anzi, con una simile macchina dalle prestazioni di calcolo modeste è già un buon risultato raggiungere una precisione di tre o quattro cifre decimali esatte.

Il metodo delle tangenti è basato sul disegno della figura 2 e mostra quanto lavoro sarebbe richiesto per determinare l'angolo φ_∞ la cui tangente vale t. Nel nostro problema questa tangente è data dal quoziente fra l'altezza e la larghezza dello specchio di stampa, quindi l'assumiamo come quantità nota. Dobbiamo risolvere allora l'equazione:

$$\varphi_\infty = \arctan t$$

ovvero:

$$\tan \varphi_\infty = t \tag{1}$$

Facendo riferimento alla figura 2 dobbiamo trovare l'intersezione della curva della tangente con la retta orizzontale $y = t$. Supponiamo di essere partiti da un'approssimazione dell'ascissa del punto di intersezione φ_0 e dopo un certo numero di iterazioni $i-1$ di essere arrivati all'ascissa φ_{i-1} dove però la funzione tangente vale y_{i-1} e non vale ancora t, come richiesto dall'equazione (1). Possiamo tuttavia determinare la pendenza della funzione tangente nel punto di coordinate (φ_{i-1}, y_{i-1}) impiegando la trigonometria "elementare"[5] per cui la pendenza della retta tangente alla curva in quel punto vale:

$$y'_{i-1} = y'(\varphi_{i-1}) = \frac{1}{\cos^2 \varphi_{i-1}} \tag{2}$$

5. La parola "elementare" è racchiusa fra virgolette e molti lettori potrebbero non conoscere nemmeno la trigonometria. Coloro che hanno fatto il liceo scientifico o alcuni indirizzi degli istituti tecnici potrebbero confermare che si tratta di un concetto elementare, ma gli altri? Non è il caso di spaventarsi: chi non conosce la trigonometria e le basi del calcolo differenziale prenda quanto segue con leggerezza e scoprirà che non è difficile da seguire anche senza le basi teoriche.

Possiamo allora intersecare questa retta tangente, di cui conosciamo un punto e l'inclinazione in quel punto, con la retta orizzontale $y = t$ che è il nostro obiettivo e determinare la nuova ascissa φ_i che risulta più vicina a φ_∞ di quanto non lo fosse φ_{i-1}:

$$\varphi_i = \varphi_{i-1} - \cos^2 \varphi_{i-1}(\tan \varphi_{i-1} - t) \tag{3}$$

Possiamo quindi ripetere la procedura iterativamente, incrementando i ad ogni iterazione, fino a quando, a causa delle limitazioni di calcolo della macchina, il valore aggiornato φ_i ottenuto smette di avvicinarsi a φ_∞ oppure quando la precisione del valore aggiornato è ritenuta soddisfacente, per esempio:

$$|\varphi_i - \varphi_{i-1}| < 0.001$$

In definitiva l'algoritmo potrebbe essere:

Inizializza $i = 1$ e $\varphi_0 = \pi/4$

finché $|\varphi_i - \varphi_{i-1}| > 0.001$

ripeti

$\varphi_i = \varphi_{i-1} - \cos^2 \varphi_{i-1}(\tan \varphi_{i-1} - t)$

$i \leftarrow i + 1$

fine

Giacomelli assicura che bastano tre o quattro iterazioni per ottenere l'angolo la cui tangente è assegnata e, pur non avendole conteggiate, non ho dubbi sulla correttezza dell'affermazione in quanto tipicamente il metodo delle tangenti converge quadraticamente e quindi ad ogni iterazione le cifre esatte possono raddoppiare.

Mi pare giusto descrivere questo algoritmo senza riportarne il codice perché il linguaggio di LATEX 3 è sperimentale, ancora non molto conosciuto e, comunque, non fa ancora parte integrale del nucleo di pdfLATEX, ma solo di alcuni pacchetti. Inoltre i concetti elementari di trigonometria che ho esposto non sono familiari a tutti i lettori, quindi anche il codice per realizzare l'algoritmo presenterebbe per loro delle difficoltà in più.

Mi sono rivolto invece all'algoritmo delle frazioni continue che serve per eseguire certi calcoli numerici in modo molto semplice usando solo numeri interi. Ho trovato che il valore di $\pi/4$ è calcolabile con la precisione consentita dai mezzi di *pdftex*

mediante la seguente frazione continua:

$$\begin{aligned}\frac{\pi}{4} &= 0.7853981634\\ &= \cfrac{1}{1+0,27323954474}\\ &= \cfrac{1}{1+\cfrac{1}{3+0,65979236633}}\\ &= \cfrac{1}{1+\cfrac{1}{3+\cfrac{1}{1+0,51562832648}}}\\ &= \dots\end{aligned}$$

che va proseguita fino ad avere tante frazioni quante sono le cifre esatte desiderate.

Ne segue che:

$$\frac{\pi}{4} = \arctan 1 = \cfrac{1}{1+\cfrac{1}{3+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{15+\cfrac{1}{2+\cfrac{1}{73}}}}}}}} \tag{4}$$

Ho allora pensato di ottenere un'approssimazione dell'arcotangente sostituendo gli '1' nei vari numeratori con il valore della tangente di cui calcolare l'arcotangente, in modo tale che l'approssimazione ottenuta sia ancora una funzione dispari come lo è la funzione arcotangente. Pertanto, siccome $[\arctan(t)]/t$ deve essere una funzione pari, l'equazione (4) porta a:

$$\arctan t \approx \cfrac{t}{1+\cfrac{t^2}{3+\cfrac{t^2}{1+\cfrac{t^2}{1+\cfrac{t^2}{1+\cfrac{t^2}{15+\cfrac{t^2}{2+\cfrac{t^2}{73}}}}}}}} \tag{5}$$

Il procedimento che ho seguito è tutt'altro che rigoroso ed è molto empirico tuttavia nell'intervallo $0 \le t \le 1$ l'errore di approssimazione è rappresentato nella figura 4. Come si vede il massimo errore di approssimazione è pari a 22,2 millesimi di grado, corrispondenti a circa 0,4 millesimi di radiante – un errore perfettamente accettabile.

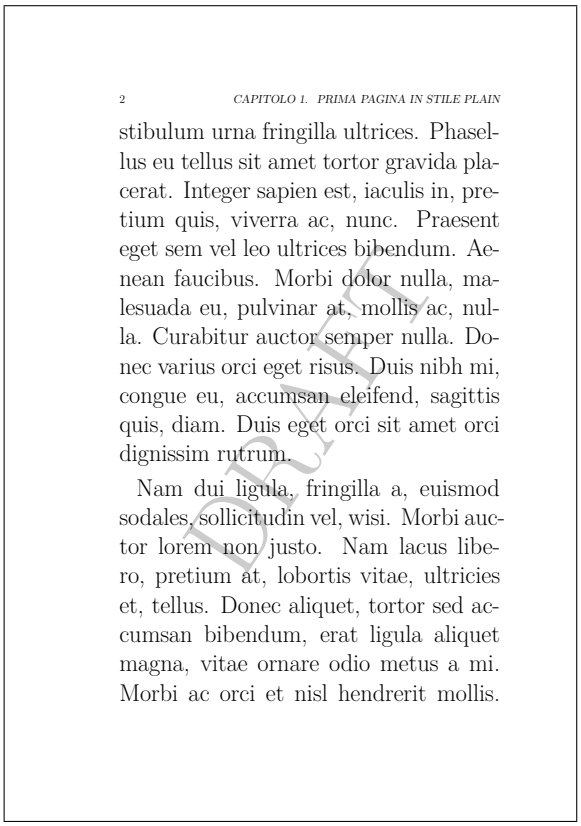


Figura 3: Filigrana inserita con la procedura manuale descritta nel paragrafo 4

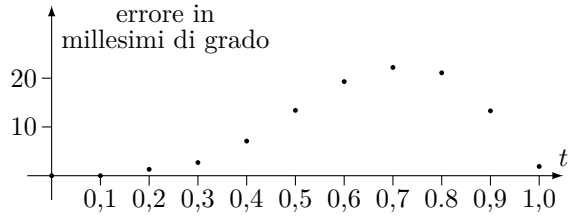


Figura 4: Errore fra la funzione approssimante e la funzione reale

Il procedimento è empirico ma è sufficientemente preciso ed è inutile spingersi oltre. Quando $t = 1$ si dovrebbe ritrovare il valore di 45° ma invece c'è un errore di 1,9 millesimi di grado, da dove arriva? Dall'aritmetica interna di *pdftex* che è a "virgola" fissa con 16 cifre binarie fratte e sulle quali si accumulano gli errori di troncamento e di arrotondamento. Raramente si può pensare o sperare che calcoli eseguiti con l'aritmetica interna di *pdftex* possano produrre errori inferiori. Nella fattispecie l'errore indicato deriva dall'accumulazione degli errori sulle ultime sette cifre binarie fratte e ce ne sono ancora nove su cui contare, per cui un errore dell'ordine del millesimo è del tutto accettabile. Ragionando in radianti il massimo errore di 22,2

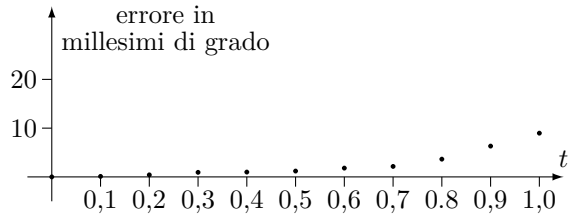


Figura 5: Errore assoluto fra la forma di Padé di grado 5 e la funzione reale

millesimi di grado corrisponde ad un errore inferiore a 0,4 millesimi di radiante, dello stesso ordine di grandezza di quello accettabile con il metodo iterativo descritto in precedenza.

Al di là di queste considerazioni di calcolo numerico, è ovvio che un errore di pochi millesimi di grado sull'inclinazione della filigrana passa del tutto inosservato per l'applicazione proposta.

Ma per chi non si accontentasse della semplice evidenza empirica è disponibile una teoria più solida basata sulle forme di Padé (Carreri, 2005). Queste sono approssimazioni razionali di serie di potenze troncate a quozienti polinomiali che generalmente hanno numeratore e denominatore formati da polinomi di grado diverso; ma quest'ultima, però, non è una condizione essenziale. Chi usa queste approssimazioni può scegliere liberamente i gradi del numeratore e del denominatore; ad esempio con polinomi pari di grado 4 l'approssimazione della funzione dispari arcotangente può essere ricavata nella forma:

$$\frac{\arctan t}{t} \approx \frac{1 + \frac{7}{9}t^2 + \frac{64}{945}t^4}{1 + \frac{10}{9}t^2 + \frac{5}{21}t^4}$$

Tenuto conto che l'aritmetica interna di *pdftex* (e anche di *xelatex*) non è molto precisa, è meglio riscrivere l'espressione in una forma tale da minimizzare il numero di operazioni da eseguire:

$$\arctan t \approx t\frac{945 + 735t^2 + 64t^4}{945 + 1050t^2 + 225t^4} \qquad (6)$$

La precisione di questo algoritmo è riportata nella figura 5. Come si vede l'errore assoluto della differenza tra la funzione approssimante e la funzione reale è molto minore dell'errore ottenuto con la frazione continua (5). Questo prova, se mai ci fosse qualche dubbio, che un problema matematico risolto con i metodi della matematica dà risultati migliori che non con metodi empirici. Non trascuriamo la possibilità di usare anche la frazione continua (5), ma in questo caso sarebbe stupido farlo.

Ciò premesso, il problema si riduce a creare il software per calcolare automaticamente l'angolo della diagonale rispetto all'orizzontale. Poiché l'errore dell'approssimazione tende ad aumentare con il valore della tangente, riduciamo gli angoli al primo ottante e quindi a valori della tangente non superiori all'unità. Le formule trigonometriche ci aiutano nello svolgere questo compito e ne teniamo conto nel codice, infatti se la tangente è maggiore dell'unità ne prendiamo il reciproco, calcoliamo l'angolo che risulta inferiore a 45°, poi ne calcoliamo il valore cercato come complemento a 90°:

```
% Definizione di alcuni registri lunghezza e di costanti
\newlength{\tang}
\newlength{\tanquad}
\newcommand{\gradrad}{57.2957795130823}% gradi al radiante
\newlength{\arctang}
\ifx\undefined\@tdG \newdimen\@tdG \fi % numeratore
\ifx\undefined\@tdH \newdimen\@tdH \fi % denominatore
\ifx\undefined\I \newcount\I \fi% contatore booleano
% Inizializzazione
\setlength\tang{\dimexpr\p@*\textwidth/\textheight
   \relax}% tangente
\setlength\tanquad{\dimexpr\tang*\tang/\p@
   \relax}% t^2
% Calcolo della approssimante di Pade'
\ifdim\tang>\p@
   \I=\@ne % arcotangente > 1
   \tang\dimexpr\p@*\p@/\tang\relax% cotangente
\else
   \I=\z@ % arcotangente <= 1
\fi
\tanquad=\dimexpr\tang*\tang/\p@\relax
%
\@tdG = \dimexpr 64\tanquad+735\p@\relax
\@tdG = \dimexpr \strip@pt\@tdG \tanquad + 945\p@ \relax % numeratore senza t
\@tdH = \dimexpr 225 \tanquad + 1050\p@ \relax
\@tdH = \dimexpr \strip@pt\@tdH \tanquad + 945\p@ \relax % denominatore
\arctan = \gradrad\dimexpr\tang*\@tdG/\@tdH\relax
\ifnum\I>0
  \arctan=\dimexpr90\p@ - \arctan\relax% complemento a 90 gradi
\fi
% Inizializzazione dell'angolo di rotazione
\AngoloBOZZA{\strip@pt\arctan}%
```

Con l'ultimo comando viene inizializzato l'angolo di rotazione della filigrana e quindi, oltre all'aggiunta di queste poche righe di codice, il contenuto del file `filigrana.sty` diventa capace di ruotare la filigrana dell'angolo corretto. La figura 3 di fatto continua a presentare la filigrana collocata correttamente in posizione e queste poche righe di codice risparmiano all'utente dal doversi calcolare a mano l'arcotangente dell'angolo di rotazione. Non solo, il codice tratta correttamente sia il caso della gabbia di stampa in verticale (*portrait*), sia quello della gabbia panoramica (*landscape*).

6 Seconda soluzione avanzata

Le soluzioni presentate sopra richiedono l'uso del comando `\rotatebox` del pacchetto graphicx. Se ne può fare a meno?

La risposta è sì, ma bisogna "sporcarsi le mani" con comandi di bassissimo livello del linguaggio di descrizione della pagina specificato dal formato PDF (Thành *et al.*, 2010).

In questo formato vengono eseguite delle operazioni grafiche che vengono svolte dai comandi del linguaggio PDF inseriti nel file finale PDF dai driver d'interfaccia che l'utente usa quando compone con *pdflatex*. Il motore di resa grafica per visualizzare o per stampare il documento comprende solo il linguaggio PDF.

La dilatazione e la contrazione, eventualmente diverse nelle due direzioni, la rotazione e la traslazione costituiscono delle trasformazioni affini e vengono controllate da una matrice di trasformazione che ora verrà brevemente illustrata. Naturalmente bisogna anche essere certi che definendo una certa matrice di trasformazione, questa non rimanga per sempre in vigore ma piuttosto esaurisca il suo compito appena eseguita la trasformazione dell'oggetto. In altre parole, la matrice di trasformazione predefinita corrisponde alla matrice identità e il vettore di traslazione è nullo; al termine di ogni diversa trasformazione è necessario ripristinare la matrice predefinita.

Qui siamo interessati ed eseguire solo la rotazione di un angolo φ e, geometricamente parlando, nel nostro caso la 'matrice' di trasformazione che usa il linguaggio PDF è semplicemente data da

$$\begin{pmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \end{pmatrix} \tag{7}$$

dove l'ultima colonna rappresenta il vettore di traslazione.

Il comando `\rotatebox` genera i due valori del seno e del coseno di φ. Se conosciamo i valori del seno e del coseno di tale angolo, conservati nelle macro `\seno` e `\coseno`, basta far precedere l'oggetto da ruotare dai comandi in linguaggio PDF introdotti mediante il comando `\pdfliteral` in questo modo[6]:

```
\pdfliteral{ q \coseno\space \seno\space
             -\seno\space \coseno\space
             0 0 cm}
```

e poi far seguire l'oggetto da ruotare dal comando di ripristino della matrice di trasformazione preesistente:

```
\pdfliteral{ Q}
```

In entrambi gli argomenti gli spazi indicati come tali o esplicitati con il token implicito `\space` sono essenziali per il linguaggio PDF. Il comando del linguaggio base `q` serve per memorizzare l'attuale matrice di trasformazione, i sei elementi successivi sono la 'matrice' scritta per colonne; il comando `cm` serve per dichiarare la precedente sequenza di valori come la "current matrix', la matrice corrente di trasformazione; il comando di ripristino `Q` ripristina la matrice di trasformazione precedentemente memorizzata.

6. I comandi gestiti dal programma di composizione *pdftex* includono anche i comandi `\pdfsave`, `\pdfsetmatrix` e `\pdfrestore` che separano le tre azioni qui descritte e impongono solo la matrice di rotazione senza dover preoccuparsi della matrice di traslazione. Dovendo scrivere un codice conviene usare i comandi più compatti forniti dall'unico comando `\pdfliteral` e con gli argomenti appositi.

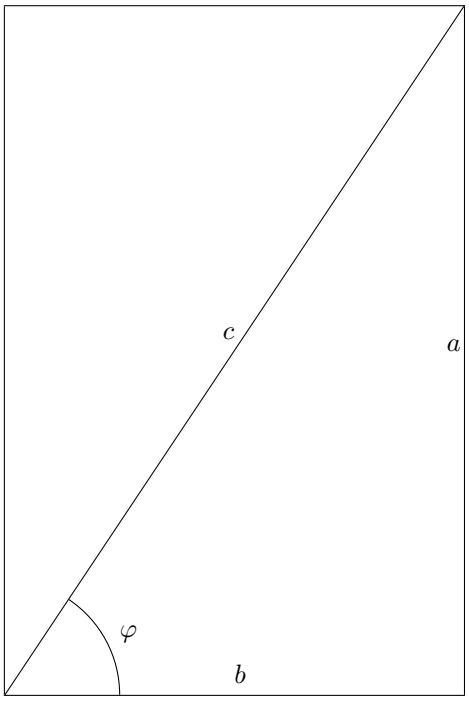


Figura 6: Lo specchio di stampa, la sua diagonale e l'angolo fra la diagonale e la base

Perciò, non usando il comando `\rotatebox`, dobbiamo ridefinire la macro che costruisce la filigrana. La definizione non è molto diversa da quella che abbiamo già usato con `\rotatebox`, solo che ora dobbiamo inserire prima e dopo la scatola che contiene la filigrana i comandi in linguaggio PDF che definiscono la matrice di rotazione:

```
\newcommand\B@ZZA{%
\begin{picture}(0,0)\unitlength\p@
\put(0,-\strip@pt\headsep){%
\put(-\strip@pt\dimexpr\textwidth/2\relax,
  -\strip@pt\dimexpr\textheight/2\relax){%
  \pdfliteral{ q \coseno\space \seno\space
  -\seno\space \coseno\space 0 0 cm}%
  \setbox\z@\hbox{\makebox(0,0){\normalfont
  \fontsize{\Draftheight}{\Draftheight}\selectfont
  \color{Draftcolor}\Draftname}}\box\z@
  \pdfliteral{ Q}}}
\end{picture}}
```

Nel paragrafo precedente sono dovuto ricorrere a calcoli non semplici per determinare il valore dell'angolo φ a partire dall'altezza e dalla larghezza dello specchio di stampa; non si può evitare tutto ciò? Visto che poi `\rotatebox` deve determinare il seno e il coseno dell'angolo di rotazione, non sarebbe preferibile calcolare subito tali valori a partire dalle dimensioni dello specchio di stampa, invece di tribolare con la formula approssimata

dell'arcotangente? Certamente. Infatti detti a e b rispettivamente l'altezza e la base dello specchio di stampa, come si vede nella figura 6, basta calcolare le grandezze seguenti:

$$a = \texttt{\textheight} \tag{8}$$

$$b = \texttt{\textwidth} \tag{9}$$

$$c = \sqrt{a^2 + b^2} \tag{10}$$

$$\cos\varphi = \frac{b}{c} \tag{11}$$

$$\sin\varphi = \frac{a}{c} \tag{12}$$

Ora però siamo di fronte ad un altro piccolo problema, bisogna far calcolare a *pdftex* una radice quadrata senza usare altri pacchetti.

Ho già ricordato che la libreria `l3fp.sty` contiene le macro in linguaggio LaTeX 3 per eseguire calcoli con valori numerici a virgola mobile compreso il calcolo per la radice quadrata e ho già spiegato perché non vorrei ricorrere a quella libreria mentre uso pdfLaTeX. Ma per il calcolo della radice quadrata l'algoritmo è così semplice che programmarlo con le espressioni di ε-TeX, facenti parte di pdfLaTeX, non è assolutamente difficile.

Infatti la radice quadrata può venire calcolata con l'algoritmo di Newton che ha una convergenza quadratica, quindi è molto veloce e preciso; esso si basa sul calcolo di una sequenza di valori sempre meglio approssimati ricavati mediante una formula ricorsiva: sia R il radicando e sia r_i l'i-esima approssimazione della radice di questa sequenza; sia r_0 il primo elemento con cui iniziare i calcoli. La formula ricorsiva diventa la seguente:

$$r_{i+1} = \frac{r_i + R/r_i}{2} \qquad \text{per } i = 0, 1, 2, 3, \ldots \tag{13}$$

Se il valore iniziale è scelto con accuratezza, il procedimento ricorsivo (13) raddoppia le cifre esatte ad ogni iterazione. A questo scopo, visto che l'altezza dello specchio di stampa è (quasi) sempre maggiore della base, allora (quasi) sempre è $x = b/a \leq 1$. Possiamo perciò riscrivere l'espressione (10) nella forma

$$c = a\sqrt{1 + x^2} \tag{14}$$

e scegliere:

$$r_0 = 1 + \frac{x^2}{2} \tag{15}$$

per iniziare le iterazioni per il calcolo della radice.

Facciamo due conti di massima. Per uno specchio di stampa piuttosto tozzo come quello ottenibile con uno specchio di stampa dalle proporzioni simili a quelle della carta in formato "letter", il rapporto x varrebbe circa 0,773, poco più di $3/4 = 0,75$; quindi il valore esatto di $\sqrt{1 + 0,75^2}$ sarebbe 1,25. Il valore r_0 con cui iniziare le iterazioni sarebbe $1,28125$, come si vede molto vicino al valore finale della successione. Invece, con uno specchio di stampa molto allungato con un rapporto $x = 0,5$, la radice esatta di $\sqrt{1 + 0,5^2}$ sarebbe $1,1180\ldots$ mentre il valore iniziale di r_0 sarebbe 1,125; anche in questo caso molto vicino al valore finale della successione.

Ecco allora che posso stendere il codice per determinare il seno e il coseno dell'angolo di rotazione in questo modo[7]:

```
% Definizione di alcuni registri lunghezza
\newlength\rappcateti
\newlength\radicando\newlength\radice
\ifx\I\undefined\newcount\I\fi
% Macro per il calcolo del seno e del coseno
\newcommand*\sincos{%
\rappcateti=\dimexpr\textwidth*\p@/\textheight
   \relax % b/h
% Calcolo del radicando e di di r_0
\radicando=\dimexpr\rappcateti*\rappcateti/\p@
   + \p@\relax
\radice=
   \dimexpr\p@+(\rappcateti*\rappcateti/\p@)/2
   \relax
% Formula di Newton con 6 iterazioni
\I=0
\@whilenum\I<6\do{%
\radice=
   \dimexpr(\radice+\radicando*\p@/\radice)/2
     \relax
   \advance\I\@ne}%
% Calcolo dell'ipotenusa
\radice=\dimexpr\radice*\textheight/\p@\relax
% Trovata l'ipotenusa ora si calcolano seno e
    coseno
\edef\seno{%
   \strip@pt\dimexpr\textheight*\p@/\radice}%
\edef\coseno{%
   \strip@pt\dimexpr\textwidth*\p@/\radice}%
}
% Si esegue ora la macro in modo da definire
% effettivamente i valori del seno e coseno
\sincos
```

Ma se volessi fissare a mano il valore dell'angolo di rotazione per avere una filigrana centrata, sì, nel centro dello specchio di stampa, ma ruotata di un angolo diverso rispetto alla sua diagonale? Avrei bisogno di un comando `\AngoloBOZZA` simile a quello definito in precedenza ma che *non* conservi soltanto il valore di questo angolo in una opportuna macro, ma che provveda a calcolare anche il seno e il coseno necessari alla rotazione. Ho dunque bisogno di un comando che accetti un argomento facoltativo. Se questo manca, allora il seno e il coseno necessari vengono determinati in base alle dimensioni dello specchio di stampa, con l'algoritmo che ho appena descritto. Se invece l'argomento facoltativo viene specificato, allora il seno e il coseno necessari vengono calcolati a partire da questo valore. Per comodità dell'utente, l'an-

7. Nella macro qui riportata le iterazioni sono 6; probabilmente è eccessivo, e un valore come 3, o meglio 4, sarebbe più che sufficiente; tuttavia i calcoli eseguiti con `\dimexpr` sono velocissimi perché vengono svolti nel "processore" del calcolatore e non impegnano la scrittura sul disco, se non per i risultati finali di ogni iterazione; quindi un paio di iterazioni in più non danno nessun fastidio.

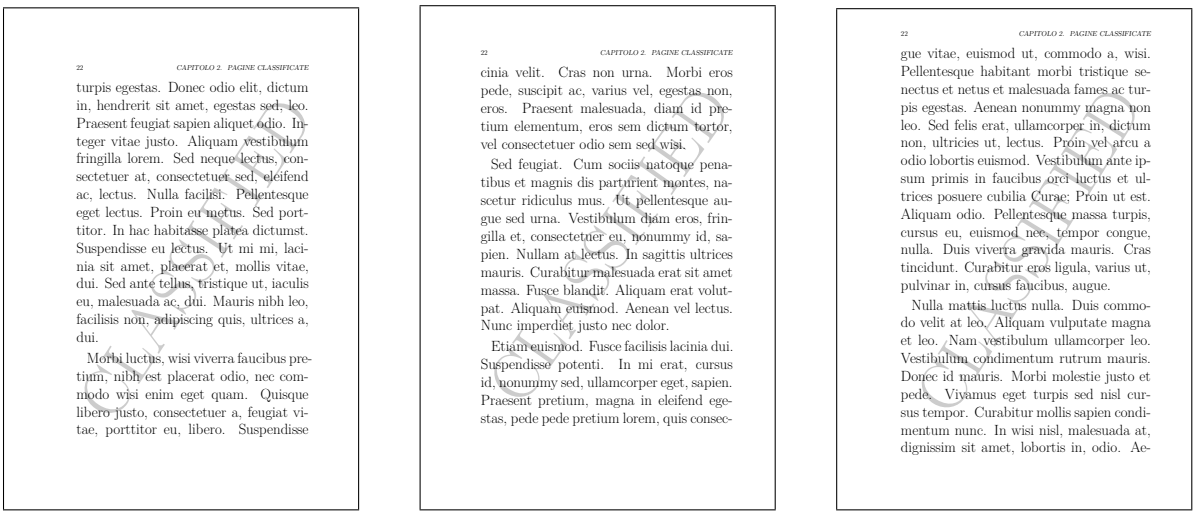


FIGURA 7: Tre layout di pagina diversi con le relative filigrane

golo facoltativo deve venire specificato in gradi sessagesimali.

Il comando in questione va definito con la sintassi LaTeX per i comandi con argomenti facoltativi in questo modo:

```
\newcommand\AngoloBOZZA[1][]{\def\@tempA{#1}%
\ifx\@tempA\@empty\sincos\else\FIL@senocoseno{#1}
    \fi}
```

Il valore di default dell'argomento facoltativo è predefinito come "vuoto" e lo sviluppo del comando prevede inizialmente la verifica che tale argomento sia effettivamente vuoto. Se lo è, esegue il comando `\sincos` che determina il seno e il coseno necessari per la rotazione a partire dai lati dello specchio di stampa. Se non lo è, provvede a determinare il seno e il coseno richiesti mediante un algoritmo abbastanza semplice: determina numericamente il (reciproco del) valore in radianti della metà dell'angolo specificato, poi calcola la tangente, la cotangente e infine determina il seno e il coseno mediante le formule parametriche:

$$\sin\varphi = \frac{2}{\cot\varphi/2 + \tan\varphi/2} \tag{16}$$

$$\cos\varphi = \frac{\cot\varphi/2 - \tan\varphi/2}{\cot\varphi/2 + \tan\varphi/2} \tag{17}$$

I calcoli per determinare il seno e il coseno richiesti sono semplici, ma tutto sta a vedere come si calcola la tangente del semiangolo (in radianti). Posto:

$$x = \frac{57,2957795}{\varphi/2} \tag{18}$$

la frazione continua che dà la tangente richiesta è data da:

$$\tan\varphi/2 = \cfrac{1}{x - \cfrac{1}{3x - \cfrac{1}{5x - \cfrac{1}{7x - \cfrac{1}{9x - \cfrac{1}{11x - \cdots}}}}}} \tag{19}$$

Per i nostri calcoli con l'aritmetica disponibile con *pdflatex* ci basta troncare la frazione continua alla parte esplicitata, eliminando i puntini. Siccome $x \to \infty$ quando $\varphi \to 0$, risulta evidente che in casi in cui si voglia usare questo algoritmo con angoli qualsiasi è meglio eseguire dei test per sostituire il calcolo per valori di $|\varphi| \lessapprox 0,1$ (circa 6°) con altri sviluppi approssimati, per esempio con la serie di Maclaurin troncata al secondo termine. Un test simile potrebbe essere eseguito per valori di φ prossimi a 180°; qui li ometto, perché mi sembrerebbe piuttosto insolito richiedere una filigrana ruotata di 180°, cioè capovolta, mentre non escluderei una filigrana orizzontale, non ruotata affatto.

Ciò premesso, la realizzazione degli algoritmi indicati nelle equazioni (16)–(19) avviene mediante il codice seguente:

```
\def\FIL@senocoseno#1{\bgroup%
% Trasforma gradi in radianti, divide per due,
% e calcola la tangente di fi/2
\@tdE=#1\p@% angolo in gradi
\ifdim\dimexpr\@tdE*\@tdE/\p@ < 36\p@
    \@tdE=0.01745329252393\@tdE% angolo in
     radianti
        \xdef\seno{\strip@pt\dimexpr\@tdE\relax}%
        \xdef\coseno{\strip@pt\dimexpr\p@-
                    0.5\@tdE*\@tdE/\p@\relax}%
\else
        \@tdB=\dimexpr 114.591559\p@*\p@/\@tdE
     \relax % 2/fi
```

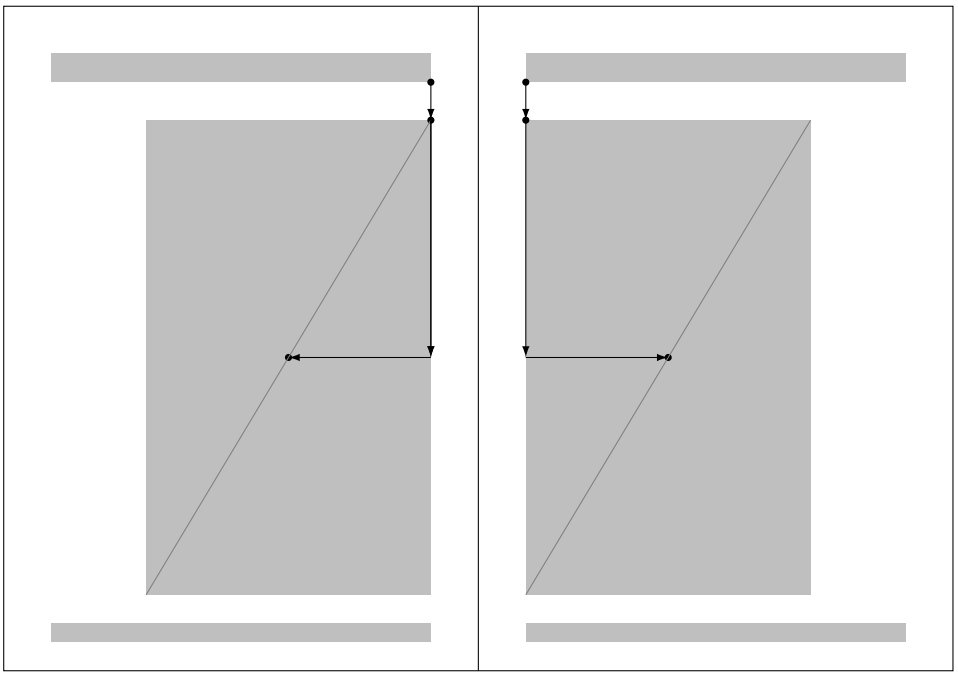


FIGURA 8: Punti di riferimento per collocare le filigrane in uno *spread* con testatine sporgenti. Le frecce indicano i successivi spostamenti dei punti di riferimento dalle estremità interne delle testatine, fino al centro degli specchi di stampa. Le diagonali dello specchio hanno la stessa direzione nelle due pagine affacciate.

```
12          \countdef\I=254\I=11\relax
13          \let\Tan\@tdA \let\Cot\@tdC
14          \Tan=\z@
15          \@whilenum\I>\z@\do{%
16             \@tdD=\dimexpr\I\@tdB-\Tan\relax
17             \Tan=\dimexpr \p@*\p@/\@tdD\relax
18             \advance\I-2\relax}%
19          \Cot=\dimexpr\p@*\p@/\Tan\relax
20 % Calcola \seno e \coseno con le formule
       parametriche
21 \xdef\seno{\strip@pt\dimexpr2\p@*\p@/(\Cot+\Tan)
       \relax}%
22 \xdef\coseno{\strip@pt\dimexpr(\Cot-\Tan)*\p@/(
       \Cot+\Tan) \relax}%
23 \fi\egroup\ignorespaces}%
```

Nella figura 7 sono riportate alcune pagine filigranate ricavate da documenti composti con la classe book e con tre diversi specchi di stampa, con quello di default oppure con lo specchio generato mediante il pacchetto canoniclayout oppure con quello generato con il pacchetto layaureo. Come si vede, le diverse proporzioni dello specchio di stampa richiedono diverse inclinazioni della filigrana ma questa appare sempre e comunque centrata sulla diagonale dello specchio di stampa.

7 Conclusione

Questa lunga spiegazione, con tre soluzioni diverse per inserire la filigrana, mostra in modo abbastanza approfondito quello che fanno dietro le quinte i vari pacchetti per inserire le filigrane o le immagini di sfondo. Naturalmente le macro che abbiamo sviluppato, volendo, potrebbero diventare il contenuto di un file di estensione personale, per esempio `filigrana.sty`. Ma sarebbe anche il caso di estendere le macro per gestire il caso di composizione fronte retro creato con testatine che sporgono nel margine esterno, dove quindi l'estremità destra delle testatine delle pagine pari e di quelle dispari non hanno la stessa posizione rispetto allo specchio di stampa. In quel caso sarebbe meglio inserire la macro che disegna la filigrana all'estremo interno delle testatine, quindi i comandi di posizionamento sarebbero diversi per le pagine pari rispetto a quelle dispari: figura 8.

Potrebbe essere anche una buona idea quella di non aggiungere le filigrane alle testatine degli stili di pagina standard, ma di creare dei nuovi stili di pagina con nomi diversi, ma che contengano le macro per le filigrane. In questo modo la decisione di mettere o meno la filigrana consiste semplicemente nel cambiare lo stile delle pagine.

Invece, come sottoprodotto di questa esposizione, ho usato spesso le espressioni dimensionali dimostrando che si possono fare calcoli abbastanza complessi anche con la modesta aritmetica dei motori di composizione del sistema TEX(esclusi quelli che si basano sull'interprete *luatex*, naturalmente, ma qui mi sono rivolto principalmente se non esclusivamente al programma *pdflatex*). Si possono calcolare anche le funzioni trascendenti e devo dire che le approssimazioni con frazioni continue che ho usato non sono impiegate spesso negli altri pacchetti di calcolo del sistema TEX. Sono particolarmente soddisfatto delle funzioni trigonometriche calcolate mediante le formule parametriche, che permettono di evitare tanti test per cercare di riportare il calcolo nel primo quadrante o addirittura nel primo ottante. Talvolta può sembrare che abbia voluto

complicare apposta i calcoli ma per mia fortuna ho usato a lungo il calcolo numerico e ho sviluppato una certa sensibilità per evitare alcune trappole che si manifestano sovente con il calcolo automatico[8]. Posso capire che i lettori, che non hanno questa esperienza, trovino gli sviluppi numerici che ho fatto un po' strani. Non si spaventino, per comporre tipograficamente non è necessario essere buoni matematici (anche se non guasta: Gutenberg a suo tempo conosceva molto bene la geometria), infatti basta usare pacchetti già creati da altri. Ma comprendendo quello che c'è nascosto dentro il pacchetto lo si usa certamente meglio.

8 Ringraziamenti

Ringrazio Claudio Fiandrino che ha messo a disposizione sul sito del Forum GuIt la sua soluzione al problema posto dal primo richiedente che si è firmato con lo pseudonimo `@virgo`; ringrazio molto anche questa persona che mi ha dato lo spunto per illustrare il problema "da dentro". Ringrazio Roberto Giacomelli per aver voluto sportivamente partecipare al "concorso" e fornire una soluzione per il calcolo dell'arcotangente che applica in modo creativo concetti imparati nel primo corso di Analisi Matematica seguito all'Università. Ringrazio Francesco Endrici, un altro partecipante sportivo al "concorso"; si è cimentato con il problema trovando la soluzione giusta per rendere la filigrana indipendente dal testo.

Riferimenti bibliografici

LaTeX3 Project Team (2005). «LaTeX 2ε font selection». Leggibile con `texdoc fntguide` con la distribuzione TeX Live.

Beccari, C. (2012). «Il LaTeX Reference Manual commentato». Scaricabile dal sito `http://www.guitex.org/home/images/doc/GuideGuIT/latexhandbookcommentato.pdf`.

Breitenlohner, P. (1998). «The ε-TeX manual». Leggibile con `texdoc etex` con la distribuzione TeX Live.

Carlisle, D. (2005). «Packages in the 'graphics' bundle». Leggibile con `texdoc graphicx` con la distribuzione TeX Live.

Carreri, A. (2005). *Approssimante di Padé*. Tesi di Laurea, Università degli Studi della Calabria. Questa tesi è in rete all'indirizzo `http://lan.unical.it/Persone/Dellaccio/careri.pdf`. Probabilmente si tratta di un monografia di laurea triennale; il testo è ben focalizzato sul tema e, a parte qualche refuso, è sostanzialmente corretto.

Gäßlein, H., Niepraschk, R. e Tkadlec, J. (2011). «The pict2e package». Leggibile con `texdoc pict2e` con la distribuzione TeX Live.

Kern, U. (2007). «Extending LaTeX's color facilities: the xcolor package». Leggibile con `texdoc xcolor` con la distribuzione TeX Live.

Knuth, D. E. (1996). *The TeXbook*. Addison Wesley, Reading, Mass., 16ª edizione.

Krab Thorup, K., Jensen, F. e Rowley, C. (2007). «The calc package – Infix notation arithmetic in LaTeX». Leggibile con `texdoc calc` con la distribuzione TeX Live.

Lamport, L. (1994). *LaTeX: A document preparation system – User guide and reference manual*. Addison Wesley, 2ª edizione.

Niepraschk, R. (2010). «The eso-pic package». Leggibile con `texdoc eso-pic` con la distribuzione TeX Live.

Rozhenko, A. I. (2004). «The watermark package». Leggibile con `texdoc watermark` con la distribuzione TeX Live.

Tantau, T. (2010). «The TikZ & PGF packages». Leggibile con `texdoc tikz` con la distribuzione TeX Live.

Thành, H., Ratz, S., Hagen, H., Jakowski, P. e Schröder, M. (2010). «The pdfTeX manual». Leggibile con `texdoc pdftex` con la distribuzione TeX Live.

8. Per esempio, mi piace molto l'artificio sviluppato nel realizzare il comando `\dimexpr` di eseguire lo "scalamento" in due mosse: lo scalamento consiste nel moltiplicare una lunghezza per un quoziente fra due lunghezze. I registri lunghezza e le lunghezze in generale contengono al massimo 32 bit, un paio di servizio e gli ultimi 16 da considerare come parte fratta della misura in punti di quella lunghezza. Prima viene eseguito il prodotto in un registro a 64 bit interno al processore, il che non comporta nessuna perdita di informazione dovuta a troncamenti o arrotondamenti, poi viene eseguita la divisione con il taglio del risultato mediante arrotondamento. In questo modo mediamente i calcoli sono più affidabili ma scegliendo adeguatamente le lunghezze su cui operare si possono eseguire moltiplicazioni fra grandezze, divisioni fra grandezze, inversioni e simili. Si tratta sempre di usare correttamente i tre ingredienti dell'espressione di "scalamento". Nel codice che ho scritto ho usato spessissimo questa operazione di scalamento ottenendo risultati migliori di quelli che avrei ottenuto con i "vecchi" comandi indicati nel TeXbook.

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

LuajitTEX

Luigi Scarso

Sommario

LuajitTEX è una variante di LUATEX che utilizza LuaJIT, un compilatore just-in-time per Lua. Vengono illustrati alcuni aspetti degli interpreti TEX, Lua e LuaJIT e vengono brevemente riferiti i primi risultati che evidenziano un guadagno medio del 25% del tempo di esecuzione.

Abstract

LuajitTEX is a LUATEX version that uses LuaJIT, a just-in-time Lua compiler. This paper shows some aspects of TEX, Lua and LuaJIT interpreters and shortly reports early results that highlight how execution time speeds up by 25% on average.

1 Introduzione

Quest'anno ricorre il ventesimo anno di Lua, la cui origine è descritta in IERUSALIMSCHY *et al.* (2007) (disponibile anche presso `http://www.lua.org/doc/hopl.pdf`) e che a partire dal 2005 è parte integrante del motore di impaginazione LUATEX, giunto oggi alla versione `0.77` — cioè, in modo un po' impreciso, al 77% della versione finale. Quasi a voler celebrare questa data, il team di sviluppo ha pubblicato nel Gennaio 2013 una nuova edizione del testo *Programming in Lua* (IERUSALIMSCHY, 2013) che fa da riferimento per la versione `5.2` del linguaggio pubblicata nel Dicembre 2011. Dato che il team di LUATEX tende ad usare l'ultima versione standard di Lua, nel Novembre 2012 è cominciato l'adattamento del codice base per integrare la versione `5.2` (lavoro terminato con la versione `0.75`), e questo è sembrato il momento opportuno per sperimentare alcune modifiche. Una di queste era verificare la possibilità di utilizzare un interprete Lua alternativo, LuaJIT di Mike Pall (`http://luajit.org`), giunto alla versione `2.0.0` proprio nel Novembre 2012 dopo un periodo di beta release di quasi tre anni. L'utilizzazione di LuaJIT porterebbe un considerevole guadagno di velocità (speed up) rispetto al tradizionale interprete: almeno due volte più veloce (speed up 2×) ed in particolari situazioni addirittura 110× (cfr. `http://luajit.org/ext_ffi.html`). Tuttavia bisogna considerare che LUATEX non usa *di per sé* programmi Lua: solo formati come LUALATEX o ConTEXt-MKIV hanno una consistente base di script Lua e quindi beneficierebbero in teoria dello speed up. Inoltre, sebbene LuaJIT e Lua siano API-compatibili (ovvero offrano la stessa interfaccia di programmazione) ed entrambi implementino un interprete Lua, internamente sono completamente differenti e non è chiaro a priori se anche LUATEX possa effettivamente trarre vantaggio o meno dalla diversa implementazione. Il lavoro su LuajitTEX è stata anche l'occasione per riconsiderare TEX e Lua nella loro specificità, ossia quella di essere interpreti: nelle prossime sezioni descriveremo TEX, Lua e LuaJIT considerandoli come programmi che processano un linguaggio. Vedremo la differenza tra un compilatore ed un interpretere e come questa differenza sfumi negli interpreti altamente efficienti come LuaJIT.

2 TEX, Lua, LuaJIT

2.1 Compilatori ed interpreti

Un compilatore è un programma che traduce un programma *sorgente* in un secondo programma *oggetto*. Questa definizione è piuttosto generica: nella pratica il sorgente è scritto in un linguaggio ad alto livello che permette cioè di esprimere concetti come cicli, condizioni, assegnazioni, operazioni aritmetiche senza far riferimento ad un preciso agente di calcolo utilizzando un linguaggio simile a quello naturale, mentre il codice oggetto è eseguibile da un agente di calcolo meccanico, come ad esempio i moderni microprocessori. Nel seguito con sorgente intenderemo un programma in un linguaggio ad alto livello, mentre con codice oggetto un programma adatto per microprocessori. Codice nativo e microcodice sono talvolta usati come sinonimi di codice oggetto.

La costruzione di un compilatore dipende sia dalle specifiche del linguaggio del sorgente che dalle specifiche del linguaggio oggetto: le prime cambiano lentamente, ma sono complicate da implementare, le seconde invece sono più facili da implementare ma cambiamo piuttosto frequentemente, in quanto cambiano i microprocessori disponibili sul mercato. Consideriamo ad esempio il compilatore GCC: accetta come sorgente programmi scritti in Ada, C, C++, Fortran, Go, Java, Objective-C, Objective-C++ e può produrre codice oggetto per svariate decine di famiglie di microprocessori — quindi circa un centinaio di microprocessori. Implementando separatamente un compilatore per ciascuna coppia (sorgente, oggetto) si avrebbero 10 linguaggi × 100 microprocessori = 1000 compilatori: chiaramente la gestione diventa un compito praticamente impossibile.

Una soluzione consiste nel suddividere il compilatore in uno stadio iniziale (*front end*) che si

occupa di un solo linguaggio sorgente ed opera come traduttore verso una forma intermedia IR (*Intermediate Representation*). Un secondo stadio (*middle end*) si occupa della ottimizzazione del formato IR ed un terzo stadio (*back end*) traduce la forma IR ottimizzata in codice assembly il quale viene a sua volta tradotto in codice oggetto da un assemblatore. Seguendo l'esempio precedente, in questo modo si hanno 10 linguaggi $\rightarrow$ 10 traduttori verso IR, 1 ottimizzatore di IR ed 1 backend $\times$ 100 microprocessori $\rightarrow$ 100 assemblatori per un totale di 111 componenti del compilatore — quindi rispetto a prima la complessità è ridotta di un ordine di grandezza.

Con riferimento alla fig. 1, il front end è suddiviso in tre fasi:

analizzatore lessicale (*Lexical Analyzer — lexer* per brevità — o *scanner*) suddivide il sorgente in *token*;

analizzatore sintattico (*Syntax Analyzer* o *parser*) raggruppa i token in unità sintattiche in modo da rispettare la grammatica del linguaggio. Questa fase produce una struttura dati chiamata albero della sintassi (*syntax tree*) del sorgente;

analizzatore semantico (*Semantic Analyzer*) controlla la correttezza semantica del syntax tree secondo le regole della grammatica — per esempio sommare un numero ed una stringa è un errore se il linguaggio non prevedere una conversione automatica tra i due tipi `numero` e `stringa`; non lo è se, per esempio, `numero` può essere trasformato in `stringa` e la somma di due stringhe (unione) è lecita.

Possiamo vedere il middle end come una sorta di minicompilatore suddiviso in due stadi:

generatore di codice intermedio (*Intermediate Code Generation*) ha in ingresso il syntax tree (il "sorgente" del minicompilatore) e produce il corrispondente IR, il "codice oggetto" di un microprocessore virtuale (ad esempio, ha un numero illimitato di registri);

ottimizzatore codice intermedio (*Code optimization*), probabilmente è lo stadio più complesso di tutto il compilatore.

Il middle end è una sorta di "camera di compensazione" tra due entità molto differenti: il linguaggio ad alto livello, comprensibile ma astratto, ed una miriade di microcodici (uno per ogni microprocessore) concreti sì, ma ciascuno con le proprie caratteristiche. Il formato IR deve essere facile da produrre e da tradurre in assembly, quindi la CPU virtuale deve astrarre le caratteristiche di ogni CPU concreta, ed è anche il formato utilizzato per la maggior parte delle ottimizzazioni. Alcune di queste sono semplici: per esempio nel seguente codice `C`

```
static void relax(){;}
int main(){return 0;}
```

la funzione `relax()` è inutile in quanto non sarà mai invocata e quindi può essere eliminata (*dead code*). È evidente che è meglio affrontare questa ottimizzazione (*Dead Code Elimination*) nel middle end, dato che nel front end dipende dalla grammatica del linguaggio sorgente e nel contempo si riduce la dimensione del codice assembly. Ma altre ottimizzazioni sono decisamente più complesse: ad esempio la maggior parte dei linguaggi high level non mette limiti al numero di variabili utilizzabile ed il compilatore assegna ad una variabile una distinta area di memoria. I registri di una CPU sono le unità di memoria più veloci, quindi è naturale cercare di usarli il più possibile: essendo il loro numero limitato sorge il problema della allocazione dei registri disponibili (*register allocation*). Purtroppo questo è un problema NP-completo: non si conosce una soluzione "buona" (per esempio in tempo polinomiale rispetto alla dimensione dei dati in ingresso), per cui si devono adottare tecniche euristiche che impattano in modo negativo sia sulla velocità del compilatore sia sulle sue dimensioni.

Infine nel back end abbiamo:

generatore codice assembly (*Target Code Generator*) che traduce il codice IR in assembly;

ottimizzatore codice assembly (*Target Code Optimizer*) in questo stadio è possibile effettuare una ulteriore riduzione del codice assembly sostituendo, quando possibile, un gruppo di istruzioni con uno più breve (*peephole optimization*). Non sempre presente nel back end, alcuni compilatori spostano l'ottimizzatore nel middle end;

assemblatore (*Assembler*) che traduce il codice assembly in codice oggetto. Tecnicamente è un programma distinto dal compilatore vero e proprio, ma è pratica consolidata che il compilatore di default esegua l'assemblatore al termine della fase di ottimizzazione.

Una ulteriore complicazione nasce dalla fondamentale necessità di eseguire passo dopo passo un programma compilato, tipicamente per verificare il motivo di un malfunzionamento e correggere il codice sorgente nel punto corretto (*debugging*). In questo caso è necessario mappare il codice oggetto verso il codice sorgente e questa "mappa" è immersa nel codice oggetto stesso e non è raro che questo comporti una aumento delle dimensioni anche di 4 volte; inoltre le ottimizzazioni come la Dead Code Elimination possono interferire con il flusso di esecuzione descritto dal sorgente e possono quindi confondere il programmatore, per cui vanno disabilitate — in sostanza il debugging richiede di progettare il compilatore *anche* per poter produrre un codice altamente inefficiente sia per le dimensioni che per la velocità.

Un compilatore *produce* codice oggetto, ma non lo esegue. Da questo punto di vista è differente

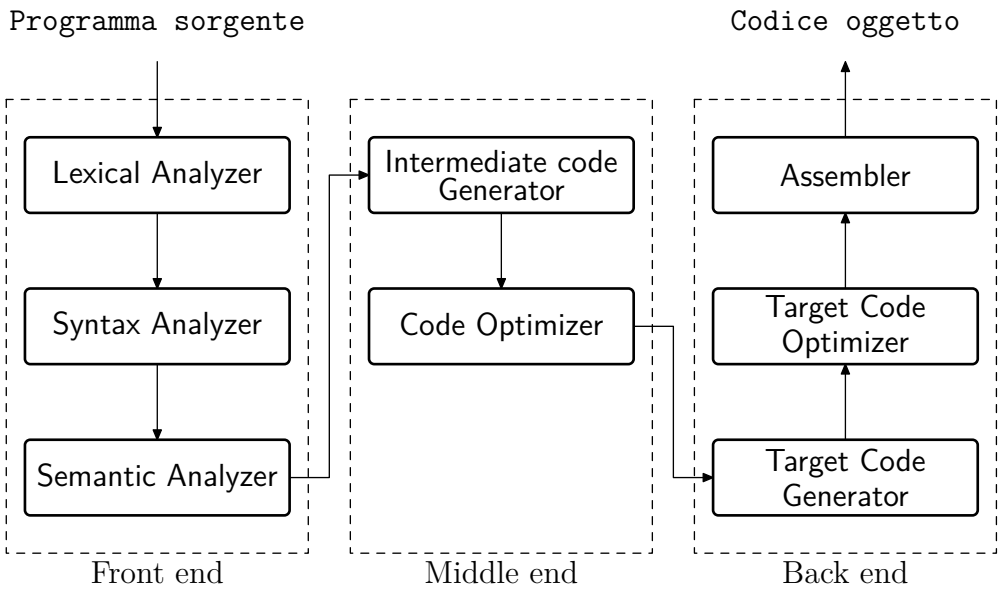


Figura 1: Stadi di un compilatore.

da un *interprete* di un linguaggio, il quale *esegue* un programma sorgente *una istruzione alla volta* fino a quando non vi sono più istruzioni da eseguire (termine dell'input, oppure `halt` dell'esecuzione per errore irreversibile, oppure uscita anticipata). Questo processo può anche essere interattivo: si legge una istruzione da terminale, la si valuta, si stampa l'eventuale risultato e si ripete il ciclo (*read-eval-print-loop* o REPL). Solitamente i linguaggi interpretati prevedono un programma REPL che viene distribuito come parte integrante dell'interprete.

Ogni CPU è un interprete del proprio microcodice: infatti ad intervalli di tempo scanditi dal clock del sistema, la CPU preleva una istruzione dalla memoria, la decodifica per trovare l'operazione da eseguire ed i relativi operandi, la esegue e ripete il ciclo (*fetch-decode-execute cycle*). Scrivere in assembly un interprete di un linguaggio high level (o very high level) non è un compito precisamente semplice, ma la teoria dei compilatori ci soccorre anche in questo caso, e con riferimento alla fig. 2 e seguendo (Parr, 2009), distinguiamo diversi tipi di interpreti identificandoli con le diverse fasi del compilatore (sebbene non vi sia una corrispondenza uno a uno):

Syntax-Directed Interpreter: appena il lexer produce un token, viene eseguita l'azione corrispondente e si ripete quindi il ciclo tornando al lexer. Semplice da costruire se il linguaggio è elementare, un SDI diventa rapidamente complicato non appena si introducono le istruzioni condizionali, i cicli, le funzioni: nel caso di una funzione, ad esempio, il codice sorgente viene memorizzato integralmente e non è possibile invocare subito l'azione corrispondente, ma solo al momento della sua chiamata (*forward reference*); inoltre ad ogni chiamata lo scanner usa sempre lo stesso codice sorgente con un impatto negativo sul tempo di esecuzione;

Tree Interpreter: costruisce il syntax tree dell'input e lo utilizza come input per l'interprete. Il vantaggio principale è la separazione tra dichiarazione di un simbolo (variabile, funzione) e risoluzione del suo attuale valore, permettendo così le forward references ed evitando il reparsing del codice sorgente. Un syntax tree inoltre può essere ottimizzato per l'esecuzione, ma per attivare l'azione l'interprete deve prima visitare un sottoalbero completo, operazione più complessa rispetto ad un SDI. In ogni caso sia un SDI che un Tree Interpreter sono indicati per linguaggi specifici per area di applicazione (*Domain Specific Language* o DSL) dove spesso la grammatica non è complicata ed i programmi si riducono a flussi di comandi; programmi come lex o yacc permettono anche di automatizzare la costruzione del lexer e del parser partendo da una specifica formale, tipicamente in formato EBNF (cfr. Levine *et al.* (1992) e Levine (2009));

Bytecode Interpreter: il codice sorgente viene tradotto in un linguaggio macchina (detto *bytecode*, in quanto vi sono al massimo 256 tipi — *opcodes* — di istruzioni e quindi un opcode è codificato in un byte) di una CPU virtuale e l'interprete (chiamato, di conseguenza, *Virtual Machine* o *VM*) emula questa CPU. È simile ad un formato IR, ma mentre quest'ultimo è pensato per l'ottimizzazione, il bytecode separa il problema della complessità del parsing (astratto e "lontano" dall'hardware) da quello dell'efficienza (ottimizzazione della memoria e della CPU reale). Ciò agevola l'implementazione della VM su diverse architetture mantenendo inalterata la grammatica del linguaggio. Per que-

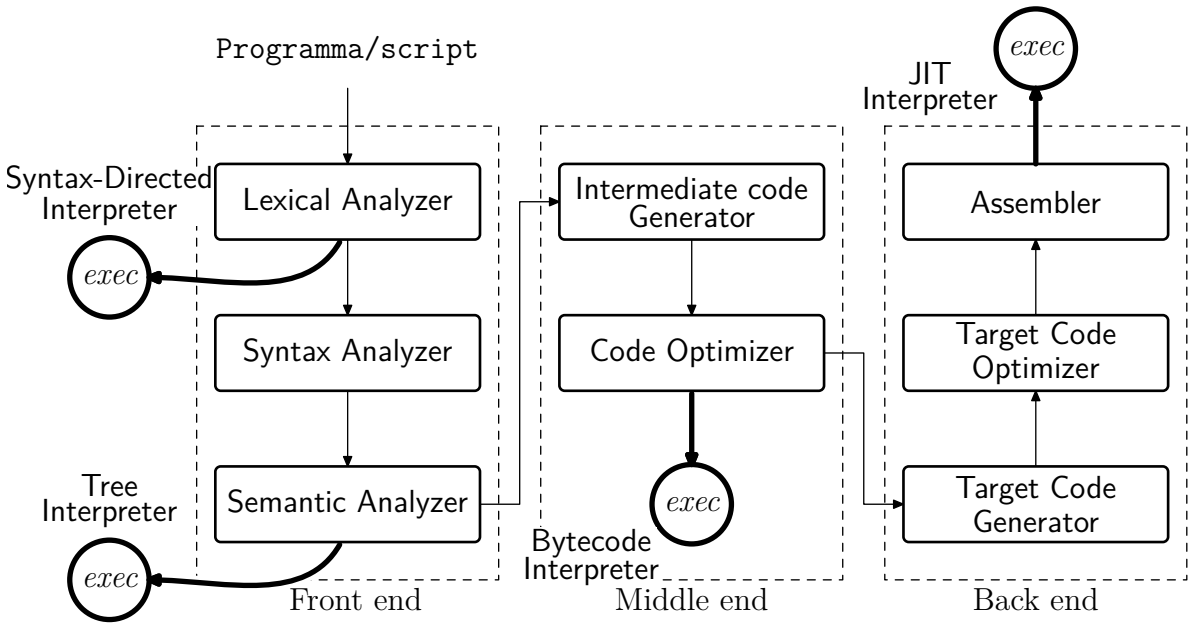


FIGURA 2: I diversi tipi di interpreti vs gli stadi di un compilatore.

sto motivo è il tipo di interprete più usato per i linguaggi ad uso generale (*general purpose*).

Analogamente ad una CPU reale, anche una VM ha tre fasi (SHI *et al.*, 2008): prelievo del bytecode dalla memoria e salto alla corrispondente azione (*dispatching*), individuazione degli operandi ed esecuzione. Normalmente il dispatching è realizzato in C/C++ come un ciclo `while` che contiene uno `switch` con tanti `case` quanti sono le opcodes del linguaggio, mentre, a seconda della struttura dati per gestire le variabili temporanee, vi sono due tipi di VM: 1) *stack based*, dove viene usato una pila (*stack*), di semplice realizzazione ma che tende a produrre bytecode più articolato; 2) *register based* dove vengono usati i registri in modo del tutto analogo ad una CPU reale, anche se in questo caso non vi sono limiti al numero di registri possibili. Il Bytecode Interpreter produce bytecode più compatto, anche se è di implementazione più complessa.

Un Bytecode Interpreter ha la possibilità di salvare il sorgente in bytecode, eliminando così la necessità di parsing ed aumentando la velocità di esecuzione dell'interprete — naturalmente se il sorgente cambia bisogna ricompilarlo, in modo analogo ad un compilatore. Gli interpreti efficienti sono in grado di ricompilare solo i sorgenti modificati e questo è di enorme aiuto nel caso di progetti con molti moduli interdipendenti, perché evita di ricompilare tutti i moduli indistintamente.

JIT Interpreter: è una specializzazione del Bytecode Interpreter e nasce dalla constatazione che alcune parti di bytecode generato non cambiano e quindi possono essere tradotte a tempo di esecuzione (*run time*) in codice macchina della CPU reale "al momento" (*just in time* or *JIT*). Un JIT Interpreter deve quindi saper riconoscere quali parti di bytecode sono *hot* (e quindi mantere una statistica del loro uso), valutare se è possibile tradurle in codice macchina e, nel caso, produrre il relativo codice, eventualmente ottimizzarlo ed infine eseguirlo. È evidente che si tratta di un interprete decisamente più complesso dei precedenti ma le prestazioni possono essere simili ad un codice compilato o, in alcuni casi dove l'ottimizzazione dinamica è più efficiente di quella statica del compilatore, addirittura superiori — ed è questo che giustifica l'investimento di risorse per la costruzione di un JIT Interpreter. La presenza di codice macchina implica ovviamente che questo interprete è specifico per una coppia CPU/Sistema Operativo ma apre anche un problema di sicurezza. Molte CPU infatti hanno un flag che impedisce ad un processo di eseguire codice macchina salvato nel proprio segmento dati e settare questo flag *può* richiedere di privilegi di amministratore. Questo significa che un JIT interpreter potrebbe non essere eseguito se privo di un certificato di idoneità, magari rilasciato da un ente indipendente dall'amministratore del sistema.

Sia un Bytecode che un JIT Interpreter prevedono di ottimizzare il bytecode, ma non bisogna comunque dimenticare l'influenza del run time: ad esempio nel programma Lua `function relax() end return 0` simile al codice C visto precedentemente, è possibile eliminare `relax()` solo quando si esegue `return 0` — ossia quanto è troppo tardi — dato che solo a quel momento l'interprete sa che `relax()` non verrà più chiamata.

È interessante osservare come Java riassume quanto visto fino ad ora. Java è un linguaggio com-

pilato high level e general purpose, ispirato al C++ ed ObjectiveC e quindi Object Oriented, creato da SUN Microsystems (successivamente acquisita da Oracle) agli inizi degli anni 90. Il suo compilatore `javac` produce esclusivamente bytecode (i *class files*) per la macchina virtuale `java` (la Java VM o JVM), originariamente un Bytecode Interpreter stack based successivamente evolutosi in un JIT Interpreter (*HotSpot*^{TM}, attualmente mantenuto da Oracle). HotSpot^{TM} è probabilmente meglio descrivibile come un trans-compilatore: il bytecode in ingresso viene tradotto in una IR ad alto livello (*HIR*) la quale viene ottimizzata e tradotta in una IR a basso livello (*LIR*) ulteriormente ottimizzata con una tecnica peephole e resa disponibile per la traduzione in codice macchina. Il termine Hotspot[1], come visto precedentemente, indica che le ottimizzazioni riguardano le parti maggiormente eseguite del flusso bytecode e quindi la VM mantiene una statistica del flusso.

Una fondamentale caratteristica di Java è che un class file è eseguibile ovunque sia presente una JVM, indipendentemente dal CPU/Sistema Operativo. `Dalvik`, sviluppata da Google per il proprio SO Android pensato per dispositivi mobili, è un altro JIT Interpreter register based per Java, ma il bytecode non è compatibile con la JVM. Sebbene in Shi *et al.* (2008) sia presentata una JVM register based più performante rispetto ad una stack based, presso `https://blogs.oracle.com/javaseembedded/entry/how_does_android_22s_performance_stack_up_against_java_se_embedded` sono disponibili dei test che puntano in direzione contraria, per cui la questione rimane ancora controversa.

2.2 TEX

Per la maggior parte degli utenti TEX, `pdflatex`, `luatex` o `xetex` sono programmi che trasformano un testo in un file `dvi` o `ps` o `pdf` (possibilmente mediante programmi come `dvips` o `dvipdf`). Superficialmente (ma non troppo, visto che gli sviluppatori parlano di back end dvi o back end pdf) la somiglianza con un compilatore è evidente: il testo è un programma in un linguaggio ad alto livello e il file di uscita ha un formato binario (almeno fino a quando ci si si limita a `dvi` o `pdf`). Visto che il formato `pdf` probabilmente è il più usato, non è improprio vedere `tex` come un *compilatore di documenti*: in questo senso è simile a `javac`, con il `pdf` al posto del bytecode. Sebbene questo punto di vista sia accettabile (e nella pratica consolidato) tuttavia si focalizza sulla relazione input-output e nasconde importanti parti che rimangono comunque interessanti da vedere.

Innanzitutto TEX è un linguaggio di programmazione ad alto livello dedicato alla tipografia digitale (quindi un DSL); è un linguaggio a marcatori di tipo procedurale, ed i marcatori sono macro definibili dall'utente. Esistono diversi programmi che sono in grado di leggere sorgenti TEX e la maggior parte di questi (o perlomeno quelli noti all'autore) sono interpreti con REPL. Analizzando il codice sorgente di `pdftex` e `luatex` si nota che la funzione `main_control` implementa il dispatching delle istruzioni: il codice sorgente viene letto sequenzialmente in spezzoni, che vengono a loro volta decomposti in coppie `(character-code, category-code)` — i `token`. Ad esempio `(ch('a'),11)` e `(ch('b'),11)` sono due token diversi perché il character code (indicato dalla funzione `ch()`) è differente, ma entrambe sono della stessa categoria sintattica `11` (lettere) e generano quindi lo stesso tipo di token. La coppia `(ch('\'),0)` invece è carattere di tipo escape per cui il resto dell'input va trattato in modo particolare: ad esempio se l'input contiene `\relax␣foo` il carattere \ segnala a TEX di produrre un token di tipo macro identificata dai caratteri `'r' 'e' 'l' 'a' 'x'` (e delimitata da ␣) che vengono quindi consumati e non trattati come lettere.
Qui sotto viene riportata la funzione relativa a `luatex`, ma quella di `pdftex` è simile:

```
void main_control(void){
  main_control_state = goto_next;
  init_main_control();
  if (equiv(every_job_loc) != null)
     begin_token_list(equiv(every_job_loc),
                      every_job_text);
  while (1) {
     if (main_control_state==goto_skip_token)
       main_control_state = goto_next;
     else
       get_x_token();
     if (interrupt != 0 && OK_to_interrupt) {
       back_input();
       check_interrupt();
       continue;
     }
     if (int_par(tracing_commands_code)> 0)
       show_cur_cmd_chr();

     (jump_table[(abs(mode)+cur_cmd)])();
     if (main_control_state == goto_return) {
       return;
     }
  }
  return;
}
```

Il punto chiave è
`jump_table[(abs(mode)+cur_cmd)])()`:
`jump_table` è un array che memorizza le funzioni (o più precisamente i puntatori alle funzioni) da eseguire per ciascun tipo di token in relazione allo stato corrente. È una variante del classico `while` con `switch-case`, il quale infatti è usato nel codice di `pdftex` e di cui si vede

1. In `http://www.ntg.nl/maps/16/14.pdf` D. Knuth lo indica con "jump trace"

ancora traccia nei tre `if-then` presenti all'interno del ciclo `while(1)`. Usare una jump table è una soluzione vantaggiosa perché mantiene il codice più corto e quindi gestibile. Un interprete TEX è quindi un SDI anche se, da questo punto di vista, è un caso limite: è inusuale per un interprete DSL un così fine controllo sull'input. Ad esempio è possibile, usando `catcode`, ridefinire la categoria dei caratteri in modo tale da trasformare TEX in un altro interprete per i records di bibTEX:

```
@ARTICLE{Ertl03thestructure,
    author = "M. Anton Ertl and David Gregg",
    title = "The Structure and Performance of
             Efficient Interpreters",
    journal = "Journal of Instruction-Level
               Parallelism",
    year = "2003",
    volume = "5",
    pages = "2003"
}
```

L'utente ha quindi il controllo del lexer e questo, unito al fatto che TEX è Turing-complete (cfr. `http://it.wikipedia.org/wiki/Turing_equivalenza`) significa che in realtà TEX è un meta-DSL — permette cioè di definire DSL. Non solo: è possibile salvare un set di macro in formato compresso per essere rapidamente caricato in un secondo momento e questo formato è portabile (entro certi limiti: cfr. `http://www.tug.org/texinfohtml/web2c.html#Memory-dumps`) — una sorta di bytecode, quindi. Tuttavia in questo caso entrano in gioco considerazioni di efficienza, e linguaggi come il C sono a tutt'oggi le scelte preferibili, anche considerando che i compilatori ANSI C (quelli conformi allo standard ANSI X3.159-1989, successivamente ratificato in ISO/IEC 9899:1990) sono molto diffusi.

Esiste però l'altra faccia della medaglia: la complessità del linguaggio TEX può essere controproducente. Consideriamo ad esempio ancora il C, la cui grammatica è fissata e non può essere cambiata in corso di esecuzione. È quindi possibile costruire un lexer per C, integrarlo in un editor ed usarlo come evidenziatore di sintassi, ad esempio per evidenziare i commenti dal resto del codice. Praticamente tutti gli editor per sviluppatori hanno un evidenziatore di sintassi per i più comuni linguaggi di programmazione, ma per il TEX non esiste a tutt'oggi una soluzione definitiva. Ad esempio, dato che è possibile ridefinire il significato di caratteri è anche possible ridefinire % come un carattere *non* di commento, ma un lexer tradizionale non cambierebbe la sua assegnazione originale e continuerebbe a vedere % come un commento, con il risultato che il codice verrebbe evidenziato in modo sbagliato. Un evidenziatore di sintassi per il TEX in pratica coinciderebbe con il programma `etex`, back end a parte.

Un secondo aspetto critico negli SDI, e quindi in TEX, è costituito proprio dallo `switch-case`: in breve, la CPU non riesce a fare una buona previsione di dove sarà trasferito il controllo dell'esecuzione e questo con le moderne architetture pipeline ha un impatto negativo sulla performance.

La soluzione ideale sarebbe eliminare del tutto lo `switch` ed utilizzare un `goto` per saltare direttamente alla subroutine che esegue l'azione, come nel seguente pseudocodice in cui la label del `goto` è una variabile (*label as value*) e quindi può essere calcolata a tempo di esecuzione (*computed goto*), e dove gli indirizzi delle subroutine sono memorizzati nell'array `dispatch_table[]` ed il flusso dei token in `instruction[]`. Il token da eseguire è indicizzato in `instruction` dalla variabile `pc`, una sorta di program counter che può essere modificato dalle subroutine `goiftrue`, `goiffalse`...:

```
static void* dispatch_table[] = {
        ...
        &&OPR_AND,
        &&OPR_OR,
        &&OPR_CONCAT,
        &&OPR_ADD,
        ...};
#define DISPATCH() \
  goto *dispatch_table[instruction[pc++]]
int pc = 0;
while (1) {
  ...
  OPR_AND:
    goiftrue(fs, v);
    DISPATCH();
  }
  OPR_OR: {
    goiffalse(fs, v);
    DISPATCH();
  }
  OPR_CONCAT: {
    exp2nextreg(fs, v);
    DISPATCH();
  }
  OPR_ADD: {
    if (!isnumeral(v)) exp2RK(fs, v);
    DISPATCH();
  }
  ...
}/* end while */
```

Notiamo che sebbene assomigli alla `jump_table` vista precedentemente, di fatto incide sulle prestazioni perché a run time la chiamata di funzione tramite puntatore è più onerosa rispetto alla invocazione della stessa subroutine definita a tempo di compilazione tramite un computed goto. Inoltre, come detto precedentemente, le moderne CPU hanno una architettura a pipeline dove (semplificando) un modulo si occupa del fetch, un secondo del decode ed un terzo di execute dell'istruzione assembly ed è concettualmente identica ad una catena di montaggio, in quanto il primo modulo dopo aver terminato il fetch dell'istruzione corrente passa il

compito al modulo successivo ed inizia il fetch della prossima istruzione. A regime, supponendo che ogni modulo impieghi lo stesso numero di cicli di clock u, ad ogni u "esce" una istruzione completa *se la pipeline è sempre "piena"* (l'approccio naïve, quello di attendere che ogni istruzione completi il ciclo richiederebbe $3u$). L'effetto di un jump si vede solo al modulo execute, dove la CPU scopre che le istruzioni nei moduli fetch e decode non sono valide perché bisogna prelevare un'istruzione diversa e quindi bisogna svuotare la pipeline e ricominciare, operazione onerosa con conseguente perdita di performance. Dato che non è possibile evitare i jump, un modulo hardware (*branch predictor*) cerca di prevedere, partendo dall'indirizzo dell'istruzione corrente, l'indirizzo della prossima istruzione supponendo valido il *principio di località spaziale*. Questo afferma che quando si accede ad un indirizzo i è molto probabile che i successivi accessi siano vicini ad i (Hennessy e Patterson, 2012, pag. 45). Nel caso di uno `switch/case` l'indirizzo corrente è l'indirizzo della variabile dello `switch`, indirizzo *che non cambia a run time*, mentre quello di destinazione è dato dal valore della suddetta variabile *che cambia a run time* e può essere una delle n label del `case`: assumendo equiprobabilità, la probabilità di caricare l'indirizzo giusto è $1/n$ e quindi non si sfrutta il principio di località spaziale. Nel caso di un computed goto l'indirizzo corrente è l'istruzione della subroutine appena eseguita *e cambia a run time*: il successivo indirizzo, relativo alla prossima subroutine, ha maggior probabilità di essere caricato dal branch predictor proprio per il principio di località spaziale (Ertl e Gregg, 2003, pag. 14). Questo comporta anche che le istruzioni da prelevare probabilmente risiederanno nella cache, il cui tempo di accesso è dalle 20 alle 160 volte minore di quello della memoria del sistema. Purtroppo labels-as-values è una estensione C non standard: alcuni compilatori come GCC la implementano, altri no, come il compilatore di Microsoft. Al contrario i puntatori a funzioni e `switch/case` sono standard, quindi è naturale che un interprete TEX che mira alla portabilità lo preferisca ad altri costrutti anche a prezzo di una minore performance (è anche possibile riscrivere `main_control` in assembly, dove le label-as-first-class-value sono ammesse, ma è evidente che si hanno problemi di portabilità ancora maggiori).

2.3 Lua

In Ierusalimschy *et al.* (2005) il team di sviluppo dichiara gli obiettivi di Lua: linguaggio general purpose, semplicità di sintassi e di realizzazione, efficienza dell'interprete in termini di memoria e velocità di esecuzione, portabilità su diverse architetture, facilità di integrazione in progetti software esistenti.

Con riferimento alle sezioni precedenti, possiamo riassumere così le sue caratteristiche:

- Lua è un linguaggio high level general purpose (a differenza di TEX). Ha la gestione automatica delle variabili non più indicizzate (*garbage collector*), un meccanismo di coroutine per esprimere la concorrenza, e le funzioni sono assegnabili come le variabili (*function as first class value*);
- l'interprete Lua è un Bytecode Interpreter, ma il bytecode non è portabile come nel caso di Java (il suo principale obiettivo infatti è solo velocizzare il caricamento di codice precompilato e non di renderlo multipiattaforma);
- la VM è register based e per design (per esempio per non limitare la portabilità del codice C) il dispatching utilizza lo `switch/case` e non i computed goto;

Lua è progettato sia per essere estensibile sia per estendere una applicazione. Nel primo caso questo significa sia che è possibile caricare moduli scritti in Lua (caratteristica comune a tutti gli interpreti general purpose) ma soprattutto che per un programmatore C è relativamente semplice scrivere moduli in C, con l'evidente vantaggio di poter usare un linguaggio compilato. Facciamo un esempio: consideriamo la funzione "esterna" `func_sin_I` che viene invocata 10^5 volte e supponiamo che sia implementata in Lua:

```
local function Lua_func_sin_I(a)
 local y=0;
 local sin = math.sin
 for i=1,1000 do
     y=sin(y+a);
 end
 return y;
end
func_sin_I = Lua_func_sin_I
--Test code
local n=1e5
local a,jfloat=0.0,0.0
local t =os.clock()
for j=1, n  do
 jfloat=j*1.0
 a=func_sin_I(jfloat)+a
end
print("done",a,n,os.clock()-t)
```

In Linux 64 bit, con Intel(R) Core(TM) i7-3610QM CPU @ 2.30GHz, il codice impiega circa 7.9 secondi:

```
$ luatex --luaonly lua-native.lua
done 2.644855993055 100000 7.93
```

Ora consideriamo la sua equivalente in C

```
double func_sin_I(double a){
 double y=0;
 int i;
 for(i=0;i<1000;i++)
     y=sin(y+a);
 return y;
}
```

ed il relativo codice, supponendo per il momento che la funzione sia disponibile nel modulo Lua `shlib`:

```
local _c, error = package.loadlib(
          "./swiglibsh.so",
          "luaopen_shlib")
if not(_c) then
  print("Errore:",error) return 1 end
local shlib = _c()
local func_sin_I = shlib.func_sin_I
-- The test code follows
```

Il tempo impiegato è ora di circa 3 secondi:

```
$ luatex --luaonly lua-swig.lua
done 2.644855993055 100000 3.01
```

con uno speed up di $7.9/3 \approx 2.6\times$. Per caricare il codice C compilato in Lua bisogna però preparare e compilare un *wrapper* (`swiglibsh.so` nell'esempio), ossia un programma in C che segua la Application Program Interface (*API*) di Lua — in breve, le API (Ierusalimschy, 2013) sono le regole da seguire affinché l'interprete Lua sia in grado di usare una funzione che risiede in una libreria esterna non necessariamente scritta in C (e per questo chiamate *foreign functions*). È interessante notare che Lua utilizza per questo scopo un meccanismo stack based: tutte le comunicazioni tra la libreria e l'interprete avvengono tramite uno stack. Le API sono progettate proprio per facilitare questa comunicazione e non sorprende che esistano programmi in grado di automatizzare il compito di creare wrappers; uno di questi è SWIG (`http://www.swig.org`). Con SWIG si descrivono in un file interfaccia i simboli della libreria da esportare e come importarli in Lua: nel nostro caso l'interfaccia è semplicemente

```
%module shlib
%{
/* C functions exported */
extern double func_sin_I(double a);
%}
/* Lua functions */
double func_sin_I(double a);
```

Successivamente si lancia il programma che produce il codice del wrapper, lo si compila e quindi lo si collega con codice della funzione in C. Il seguente script mostra (per Linux) l'intero processo:

```
swig -lua shlib.i
O=-O2
gcc="gcc -msse4.2 "
## libreria
$gcc  $O -fPIC -I./ -o shlib.o -c shlib.c
$gcc  $O -shared -I./ shlib.o -lm -o libsh.so
## wrapper: swiglibsh.so è il modulo Lua
$gcc $O -I./ -I/usr/include/lua5.2 -fPIC \
   -o shlib_wrap.o -c shlib_wrap.c
$gcc $O  -I./ -shared shlib_wrap.o \
   -l:./libsh.so -llua5.2 -lm -o swiglibsh.so
```

È da notare come script, file interfaccia e codice C siano in tutto meno di 500 bytes. SWIG è così affidabile che raramente è necessario modificare il wrapper C prodotto, lavorando normalmente solo sul file di interfaccia ad alto livello[2].

LuaTEX è invece un esempio di una applicazione, pdfTEX in questo caso, estesa con Lua. L'idea è semplice: scelta una funzione target (come `line_break` che in pdfTEX si occupa di calcolare i breakpoints della linea) si aggiunge nel codice sorgente una funzione (*callback*, come `lua_linebreak_callback` in questo caso) che chiama l'interprete Lua e riceve il risultato:

```
void line_break(boolean d,
                int line_break_context){
...
callback_id =
 callback_defined(linebreak_filter_callback);
if (callback_id > 0) {
 callback_id =
  lua_linebreak_callback(d, temp_head,
             addressof(cur_list.tail_field));
...
}
```

il callback viene invocato con opportuni parametri ed è il punto in cui si interagisce con lo stack:

```
int
lua_linebreak_callback(int is_broken,
    halfword head_node, halfword * new_head){
int a;
register halfword *p;
int ret = 0;                /* failure */
lua_State *L = Luas;
int s_top = lua_gettop(L);
int callback_id =
  callback_defined(linebreak_filter_callback);
if (head_node == null ||
    vlink(head_node) == null ||
     callback_id <= 0) {
    lua_settop(L, s_top);
    return ret;
}
if (!get_callback(L, callback_id)) {
   lua_settop(L, s_top);
    return ret;
}
nodelist_to_lua(L, vlink(head_node));
lua_pushboolean(L, is_broken);
if (lua_pcall(L, 2, 1, 0) != 0) {
    fprintf(stdout, "error: %s\n",
             lua_tostring(L, -1));
    lua_settop(L, s_top);
    error();
    return ret;
```

2. Il progetto SwigLib (`http://www.luatex.org/swiglib.html`) intende esplorare proprio l'area di sviluppo di moduli C per LuaTEX con SWIG. Alcuni sono già disponibili presso `https://foundry.supelec.fr/scm/viewvc.php/trunk/?root=swiglib`. SwigLib è in parte sponsorizzato dal TEX User Group, cfr. `http://tug.org/tc/devfund/grants.html`

```
}
p = lua_touserdata(L, -1);
if (p != NULL) {
    a = nodelist_from_lua(L);
    try_couple_nodes(*new_head,a);
    ret = 1;
}
lua_settop(L, s_top);
return ret;
}
```

Con `get_callback(L, callback_id)` si pone nello stack la funzione Lua identificata da `callback_id` (`linebreak_filter` in questo caso); con `nodelist_to_lua(L, vlink(head_node))` e `lua_pushboolean(L, is_broken)` si aggiungono due argomenti, sempre nello stack, ed infine con `lua_pcall` si chiama la funzione; `p = lua_touserdata(L, -1)` è il risultato, e se è non nullo viene convertito in una lista di nodi e passato all'interprete TEX tramite `try_couple_nodes(*new_head,a)`. In questo caso è possibile quindi rimpiazzare l'algoritmo `line_break` con uno proprio, senza dover ricompilare il codice. LUATEX ha qualche centinaio di callbacks (oltre ad una versione embedded di METAPOST) — TEX è un programma piuttosto complicato.

Infine un ultimo cenno ad un'altra caratteristica di Lua che è molto importante per il TEX: l'uso di *self-describing data format* che sono costrutti Lua. In questo modo è più facile adattare Lua al linguaggio che si vuole estendere: per esempio ecco come potrebbe apparire la versione Lua del record BibTEX visto precedentemente:

```
current_key=nil
current_type=nil
bibtex_db={}
function insert(...)
 local _t=bibtex_db[current_type] or {}
 _t[current_key] = (...)
 bibtex_db[current_type]=_t
end
article=function(key)
  current_type,current_key='article',key
  return insert
end
ARTICLE,Article=article,article
ARTICLE "Ertl03thestructure" {
    author = "M. Anton Ertl and David Gregg",
    title = "The Structure and Performance of
             Efficient Interpreters",
    journal = "Journal of Instruction-Level
               Parallelism",
    year = "2003",
    volume = "5",
    pages = "2003"
}
print(bibtex_db.
      article["Ertl03thestructure"].author)
print(bibtex_db.
      article["Ertl03thestructure"].title)
```

`ARTICLE "Ertl03thestructure"` è una chiamata alla funzione `ARTICLE(key)` (alias di `article(key)`) la quale ritorna a sua volta la funzione `insert(...)` che viene chiamata con la tabella `{author=...,pages="2003"}`. Come si vede la somiglianza con il record di BibTEX è notevole.

Notiamo che quanto detto fino a qui si applica direttamente a LUATEX, anche se attualmente non è presente un REPL Lua nativo: per eseguire script Lua a riga di comando bisogna usare la sintassi `$ luatex --luaonly <scritp-lua>`, mentre con `\directlua <16-bit number> <general text>` si possono eseguire script Lua da TEX.

2.4 LuaJIT

Si è visto precedentemente come dalla coppia Domain Specific Language/Direct Syntax Interpreter di TEX si sia passati alla coppia general purpose/-Bytecode Interpreter di Lua: vediamo ora come LuaJIT si inserisce in questa linea evolutiva.
LuaJIT è un interprete Lua 5.1, meno focalizzato sulla portabilità e più sulla velocità e sulla memoria. Vi sono tre importanti novità da evidenziare:

1. la VM è in assembly: il codice è minore, implementa la tecnica dei computed goto vista in precedenza per cui ha migliori performance in termini di occupazione di cache e branch prediction ed è più veloce di una VM in C;

2. LuaJIT è un JIT-Interpreter *tracing just-in-time*, simile a HotSpot™ di Oracle; diverse funzioni delle librerie standard sono compilate just in time, ma un buon numero ancora no (*NYI*, Not Yet Implemented cfr. `http://wiki.luajit.org/NYI`);

3. Il modulo `ffi` è integrato nella VM (*foreign function interface*) e offre funzionalità simili a SWIG, ma *evitando* il wrapper intermedio.

Non è difficile vedere le performance di `func_sin_I` in LuaJIT utilizzando `ffi`, la funzione `math.sin` (jit-compiled) ed infine il wrapper SWIG, il cui codice sorgente è valido ma deve essere collegato con la libreria `libjit` e non `liblua`:

```
-- ----------------------------------------
-- test.lua
-- ----------------------------------------
--
-- ffi interface
local ffi = require("ffi")
ffi.cdef[[
double func_sin_I(double a);
]]
local shlib = ffi.load("./libsh.so")
--
-- SWIG interface
local _c, error =
  package.loadlib("./swiglibsh_jit.so",
                  "luaopen_shlib")
if not(_c) then print("Errore:" ,error)
```

```
    return 1 end
local SWIG_shlib = _c()
--
-- LuaJIT native
local function Lua_func_sin_I(a)
 local y=0;
 for i=1,1000 do
     y=math.sin(y+a);
 end
 return y;
end
local f1 = shlib.func_sin_I
local Lua_f1 = Lua_func_sin_I
local SWIG_f1 = SWIG_shlib.func_sin_I
--
--Test code
local n=1e5
local a=0.0
local jfloat = ffi.new('double',0)
local t=os.clock()
for j=1, n  do
 jfloat=j*1.0
 a=shlib.func_sin_I(jfloat)+a
end
print("jit/ffi",a,n,os.clock()-t)
local t=os.clock()
a=0;jfloat=0
for j=1, n  do
 jfloat=j*1.0
 a=Lua_f1(jfloat)+a
end
print("jit/native",a,n,os.clock()-t)
local t=os.clock()
a=0;jfloat=0
for j=1, n  do
 jfloat=j*1.0
 a=SWIG_f1(jfloat)+a
end
print("jit/SWIG",a,n,os.clock()-t)
```

Eseguendo il programma con
`$>luajittex --jiton --luaonly test.lua`
abbiamo:

```
jit/ffi    2.644855993055 100000 2.98
jit/native 2.6448559930926 100000 3.69
jit/SWIG   2.644855993055 100000 2.99
```

Confrontata con la versione nativa precedente lo speed up ora è di $7.9/3.7 \approx 2.1\times$. È interessante notare come in questo caso le prestazioni si equivalgano se si delega al C la funzione `func_sin_I`: tuttavia in `http://luajit.org/ext_ffi.html` è mostrato un esempio dove la sostituzione di una `table` di Lua con un `struct` in C ha come effetto uno speed up del $110\times$ e una riduzione di memoria occupata di 64 volte (su processore `x86_64`).

Non è difficile ora capire cos'è LuajitTEX: è LuaTEX dove Lua è stato sostituito con LuaJIT, cosa possibile grazie al fatto che le API sono sostanzialmente invariate. A parte il nome `luajitex`, l'invocazione dell'interprete Lua a riga di comando o da TEX è analoga a quanto visto per LuaTEX.

3 Primi risultati

LuajitTEX è un progetto relativamente giovane: la prima release ufficiale è di fine Dicembre 2012 e per il momento l'obiettivo principale è verificare la corretta implementazione e misurare speed-up, rimanendo in sincronia con le release di LuaTEX e LuaJIT. Dato che ConTEXt-MKIV attualmente ha il più esteso codice base in Lua, alcuni test sono stati condotti da H. Hagen (Hagen, 2013) ed i risultati si possono così riassumere:

1. la VM in assembly introduce in media uno speed up $2\times$;
2. il compilatore JIT tende a degradare leggermente le prestazioni.

Questi risultati sono coerenti con quanto visto precedentemente, ma sono relativi *solo* al codice Lua mentre ConTEXt (LuaTEX) usa anche l'interprete TEX che non è influenzato da LuaJIT: i test condotti mostrano che il codice TEX occupa una quantità dal 30% al 70% del tempo totale per cui lo speed up complessivo varia da $1.15\times$ a $1.35\times$ — in media cioè si ha un guadagno del 25% del tempo, comunque significativo. È ancora in corso di studio invece il degrado dovuto al JIT, che per questo motivo è disattivato per default: la spiegazione potrebbe essere l'utilizzo delle funzioni NYI e delle chiamate C verso Lua che possono interferire con il tracing del bytecode, impedendone la compilazione.
Infine il consumo di memoria non mostra apprezzabili differenze.

4 Conclusioni

È interessante notare come LuajitTEX riassuma l'evoluzione dell'interprete, partendo dal più semplice (TEX) per arrivare all'implementazione odierna dove sfuma il confine tra compilatore ed interprete.

L'incremento di performance di LuajitTEX non è trascurabile e, da questo punto di vista, lo qualifica come una valida alternativa a LuaTEX, ma la performance non è l'unico parametro e forse neanche il più importante. Un nuovo interprete TEX deve essere disponibile per le stesse piattaforme di LuaTEX, ma in questo caso si tratta di un difficile obiettivo per LuajitTEX dato che la VM è implementata in codice assembly per piattaforme specifiche. Inoltre anche la scelta del compilatore è vincolante: è richiesto almeno `gcc 3.4` ed attualmente non è il compilatore usato per la TEX Live. Per il momento l'eseguibile è disponibile per diverse piattaforme Unix/Mac presso `http://svn.contextgarden.net/suite-bin/tex/` (grazie a Mojca Miklavec): una versione per windows 32 bit si trova presso `http://w32tex.org/` (grazie al prof. A. Kakuto).
Tuttavia il compilatore non è il solo problema: il team di sviluppo di Lua e quello di LuaJIT (essenzialmente una persona) sono separati e non

condividono una comune *roadmap*. Ad esempio la versione di luaTEX per la TEX Live 2013 si baserà su Lua 5.2.1, ma LuaJIT si basa su Lua 5.1 con una integrazione per Lua 5.2 non a livello di API, ma solo di linguaggio, e non pare ci sia l'intenzione a breve di portarsi ad una piena compatibilità con Lua 5.2. Quindi allo stato attuale Lua appare un progetto più solido se non altro perché LuaJIT è un *one-man* project, ma non si deve trascurare che la prima release è del Settembre 2005 e che con la versione 2.0 la comunità è notevolmente cresciuta. Un ambito in cui LuajitTEX può competere con luaTEX è il *data publishing* in cui vi siano requisiti sia di velocità che di interfacciamento verso altri sistemi (ad esempio DBMS o server WEB) ed in questo contesto il modulo `ffi` è una funzionalità interessante, soprattutto se è possibile sfruttare il self-describing data con il compilatore jit evitando le funzioni NYI. Anche in questo caso, però, la differenza tra la complessità di un'interfaccia `ffi.cdef` di LuaJIT e la medesima in SWIG per standard Lua non è così rilevante — e SWIG è un pre-processore C più completo e può produrre wrappers per altri linguaggi oltre a Lua. Nel caso, è eventualmente possibile scrive un modulo SWIG che emuli `ffi`, i.e. un wrapper alla libreria libffi (cfr. `https://foundry.supelec.fr/scm/viewvc.php/trunk/experimental/libffi/?root=swiglib`). Si tratta in sostanza di una caratteristica affascinante sì, ma che deve essere valutata in base al progetto.

Riferimenti bibliografici

Ertl, M. A. e Gregg, D. (2003). «The structure and performance of efficient interpreters». *Journal of Instruction-Level Parallelism*, **5**, p. 2003.

Hagen, H. (2013). «ConTEXt: Just in Time LuaTEX». *TUGboat*, **34** (1), pp. 72–78.

Hennessy, J. L. e Patterson, D. A. (2012). *Computer Architecture — A Quantitative Approach*. Morgan Kaufmann, 5ª edizione.

Ierusalimschy, R. (2013). *Programming in Lua, Third Edition*. Lua.org.

Ierusalimschy, R., de Figueiredo, L. H. e Celes, W. (2005). «The implementation of lua 5.0». *Journal of Universal Computer Science*, **11** (7), pp. 1159–1176. `http://www.jucs.org/jucs_11_7/the_implementation_of_lua`.

— (2007). «The evolution of lua». In *Proceedings of the third ACM SIGPLAN conference on History of programming languages*. ACM, New York, NY, USA, HOPL III, pp. 2–1–2–26. URL `http://doi.acm.org/10.1145/1238844.1238846`.

Levine, J. R. (2009). *flex and bison - Unix text processing tools*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

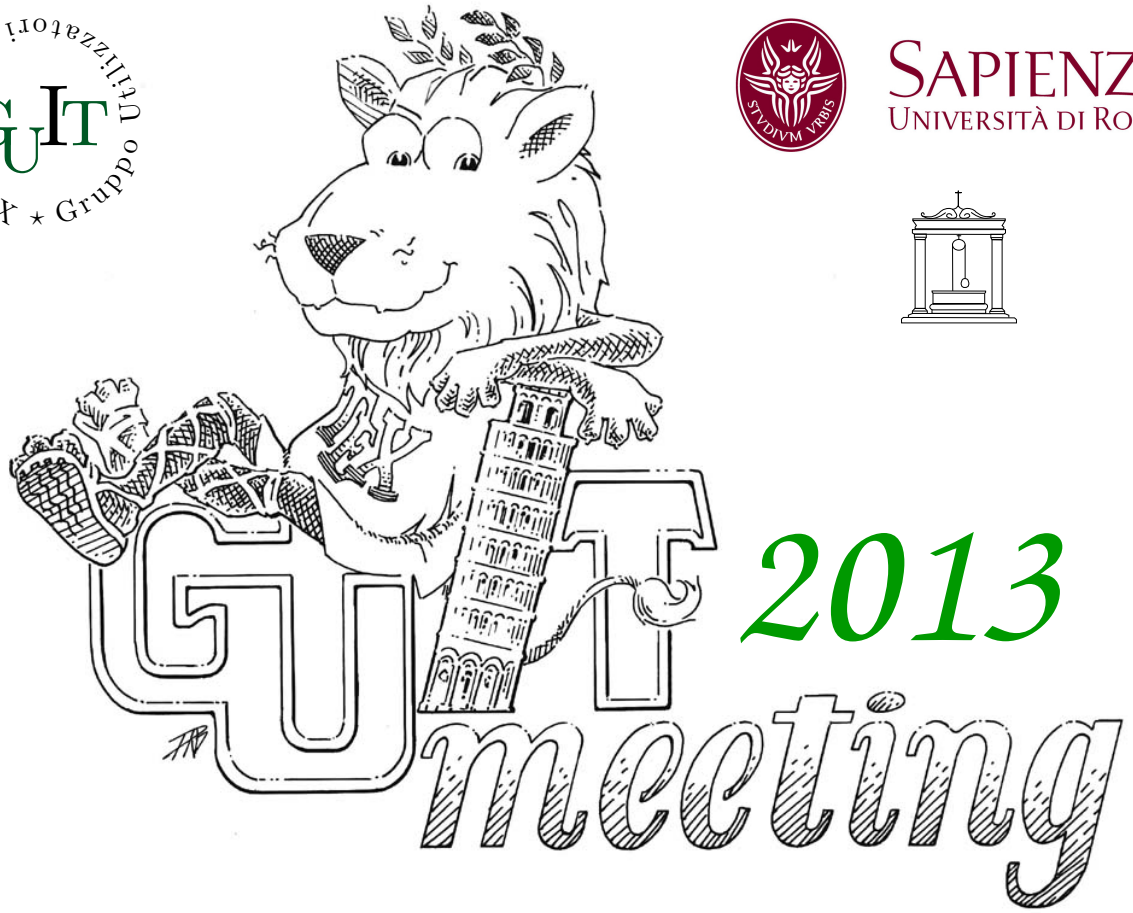
Levine, J. R., Mason, T. e Brown, D. (1992). *lex & yacc (2nd ed.)*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Parr, T. (2009). *Language Implementation Patterns: Create Your Own Domain-Specific and General Programming Languages*. Pragmatic Bookshelf, 1ª edizione.

Shi, Y., Casey, K., Ertl, M. A. e Gregg, D. (2008). «Virtual machine showdown: Stack versus registers». *ACM Trans. Archit. Code Optim.*, **4** (4), pp. 2:1–2:36. URL `http://doi.acm.org/10.1145/1328195.1328197`.

▷ Luigi Scarso
`luigi dot scarso at gmail dot com`

Questa rivista è stata stampata
presso Editrice Rotas Barletta
su carta Fedrigoni Freelife Vellum white 80 g



Decimo convegno nazionale su TEX, LATEX e tipografia digitale

Roma, 26 ottobre 2013

Sala del chiostro
Facoltà di Ingegneria
Sapienza Università di Roma

Call for Papers

Sabato 26 ottobre 2013 presso la sala del chiostro della facoltà di Ingegneria della Sapienza Università di Roma si terrà il decimo Convegno annuale su TEX, LATEX e tipografia digitale organizzato dal Gruppo Utilizzatori Italiani di TEX. Il Convegno sarà un momento di ritrovo e di confronto per la comunità LATEX italiana, tramite una serie di interventi atti sia a contribuire all'arricchimento sia a supportarne lo sviluppo.
Maggiori informazioni sul Convegno e sulle modalità di presentazione degli interventi sono disponibili all'indirizzo:

`http://www.guitex.org/guitmeeting/2013/`

ArsTEXnica

Rivista italiana di TEX e LATEX

Numero 15, Aprile 2013

30015
9 771828 236001