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`