Semantica operazionale dello strumento da riga di comando `docxwatermarker`

Fabio F.G. Buono

2026

Versione 1

1 Introduzione

Queste note danno una semantica operazionale del sottocomando `stamp`. Dove il documento gemello *Semantica assiomatica di `docxwatermarker`* specifica la libreria attraverso triple di Hoare, queste note si rivolgono allo strumento da riga di comando che la chiama. Qui l'oggetto di studio è il flusso di controllo del comando stesso, l'ordine delle sue fasi e il codice d'uscita che ogni fallimento produce. La trattazione segue gli appunti di Barbuti, Mancarella e Turini [1], di cui usiamo direttamente i sistemi di transizioni, con il riferimento standard a Plotkin [2] per lo stile strutturale e a Winskel [3] per i fondamenti.

Lo strumento da riga di comando `docxwatermarker` avvolge la libreria omonima dietro due sottocomandi, `stamp` e `inspect`. Quindi la libreria espone funzioni pure mentre lo strumento è un processo. Legge gli argomenti, apre un template, costruisce un'immagine di sostituzione, esegue lo scambio, scrive il risultato e su richiesta produce un PDF, e a ogni passo può fermarsi e restituire un codice d'uscita numerico che registra fin dove è arrivato. Chi usa lo strumento da script legge quei codici, e il loro significato fa parte del contratto dello strumento.

La specifica assiomatica e quella operazionale descrivono lo stesso software da due angolazioni. La prima dice cosa garantisce ciascuna operazione della libreria, la seconda dice come il comando che chiama quelle operazioni avanza e termina. Dei due sottocomandi, questo documento si concentra su `stamp`, che porta l'intera catena di operazioni e i codici d'uscita differenziati. Il sottocomando `inspect`, un elenco in sola lettura, è trattato in breve alla fine.

2 Sintassi dei comandi e riconoscimento del formato

Prima delle regole di transizione, due parti dello strumento ammettono una descrizione nei termini classici dei linguaggi formali, ed enunciarle per prime rende più preciso il resto. Un linguaggio è un insieme di stringhe ammissibili su un alfabeto, e due strumenti standard ne descrivono uno. Una *grammatica* genera le stringhe che il linguaggio contiene, e un *automa* le riconosce. Usiamo una grammatica per la riga di comando che l'utente scrive, e un automa per un punto interno della libreria.

Una parola su cosa viene, e cosa non viene, modellato così. La libreria è una collezione di funzioni pure, ciascuna è caratterizzata dalle triple di Hoare del documento gemello. Una funzione pura non ha stati fra cui può muoversi, perciò un automa non la descriverebbe, inventerebbe una struttura che il codice non ha. Un punto interno è davvero un riconoscitore, la funzione che identifica il formato di un'immagine dai suoi byte iniziali. È questo che l'automa qui sotto modella, invece il flusso di controllo del comando, un processo che attraversa fasi, è l'oggetto del sistema di transizioni nelle sezioni che seguono.

2.1 Grammatica della riga di comando

La grammatica seguente genera le righe di comando che lo strumento accetta, nella notazione delle grammatiche libere da contesto. I terminali sono in `typewriter`, le metavariabili *path*, *text* e *name* stanno per un percorso del filesystem, una stringa arbitraria e un nome di preset. Le parentesi $[\,\cdot\,]$ segnano una parte opzionale e la barra | segna una scelta. Mentre la grammatica rispecchia il parser in `cli.py`, dove ogni alternativa qui sotto corrisponde a un argomento o a un gruppo mutuamente esclusivo dichiarato lì.

comando	::=	`stamp` *stamp-args* | `inspect` *path* [`--debug`]
stamp-args	::=	*path opzione**
opzione	::=	*sorgente* | *target* | *pdf* | *stile* | *modo*
sorgente	::=	`--image` *path* | `--watermark-text` *text* | `--preset` *name*
target	::=	`--use-marker` | `--target-filename` *path*
pdf	::=	`--pdf` | `--pdf-only`
stile	::=	`--size` *int* | `--rotation` *float*
modo	::=	`--output` *path* | `--interactive` | `--verbose` | `--debug`

Le opzioni lunghe `--output`, `--interactive` e `--verbose` hanno gli alias brevi `-o`, `-i` e `-v`, che il parser accetta in modo intercambiabile.

La grammatica descrive la forma che il parser accetta, non ogni vincolo che impone. Tre dei gruppi di opzioni, *sorgente*, *target* e *pdf*, sono mutuamente esclusivi, e il parser rifiuta una riga di comando che usa due alternative dello stesso gruppo. L'ulteriore regola per cui `stamp` richiede una sorgente, fornita da un'opzione *sorgente* o interattivamente sotto `-i`, è controllata dopo il parsing, ed è la fase `resolve` del sistema di transizioni a imporla.

2.2 Un automa per il riconoscimento del formato

Quando la libreria sostituisce un'immagine, identifica per prima cosa il formato dei byte di sostituzione, nella funzione `_detect_format` di `_imagedetect.py`. La funzione legge i byte iniziali e li accetta come uno dei formati noti, oppure li rifiuta. È un riconoscitore su un piccolo linguaggio di prefissi di magic byte, e un automa a stati finiti deterministico lo descrive.

L'automa legge l'input byte per byte a partire dallo stato iniziale q_0. Nella funzione di transizione qui sotto, una riga è lo stato corrente, l'intestazione di colonna è il byte letto, e la cella è lo stato successivo. Gli stati PNG, JPEG, GIF, BMP e TIFF sono accettanti e nominano il formato riconosciuto. Lo stato R è l'unica sottigliezza. Il prefisso `RIFF` è condiviso da più formati, perciò raggiungere R non basta, e l'automa deve leggere altri quattro byte e controllare `WEBP` prima di accettare. Ogni byte non mostrato porta allo stato di rifiuto $\bot$, da cui non si esce.

Stato	Input letto	Stato successivo
q_0	`0x89 50 4E 47 0D 0A 1A 0A`	PNG (accetta)
q_0	`0xFF D8 FF`	JPEG (accetta)
q_0	`GIF87a` *o* `GIF89a`	GIF (accetta)
q_0	`BM`	BMP (accetta)
q_0	`II*\0` *o* `MM\0*`	TIFF (accetta)
q_0	`RIFF`	R
R	`WEBP` (4 byte all'offset 8)	WEBP (accetta)
R	altri 4 byte qualsiasi	$\bot$ (rifiuta)
q_0	altro prefisso qualsiasi	$\bot$ (rifiuta)

L'automa riconosce il linguaggio delle intestazioni di immagine ammissibili, lo stesso insieme che la libreria tratta come input valido. Una difformità di formato in `replace_image`, il

codice d'uscita 6 nel sistema di transizioni, è esattamente il caso in cui i byte di sostituzione e l'immagine bersaglio finiscono in stati accettanti diversi di questo automa.

3 Sistemi di transizioni

Richiamiamo l'apparato che usiamo, nella forma data in [1]. Un sistema di transizioni è una tripla $\langle \Gamma, T, \to \rangle$ dove Γ è un insieme di configurazioni, $T \subseteq \Gamma$ è l'insieme delle configurazioni terminali, e $\to$ è la relazione di transizione. Una configurazione è uno stato in cui il sistema si può trovare, e una configurazione terminale è una in cui il sistema ha terminato. Scriviamo $\gamma \to \gamma'$ quando la coppia appartiene a $\to$, e $\gamma \to^* \gamma'$ quando esiste una derivazione finita $\gamma \to \gamma_1 \to \ldots \to \gamma'$. Una configurazione non terminale da cui non parte alcuna transizione è *bloccata*. La semantica che segue è costruita perché nessuna configurazione raggiungibile sia bloccata, ogni cammino termina in una terminale.

La relazione di transizione è data da regole condizionali nella forma a frazione

$$\frac{\pi_1 \quad \pi_2 \quad \cdots \quad \pi_n}{\gamma \to \gamma'}$$

dove le premesse π_i sono i prerequisiti alla transizione. Una premessa è un'uguaglianza $x = t$, una relazione $t \mathrel{rel} t'$, o un'altra transizione $\delta \to' \delta'$. Una regola con una premessa di transizione è ricorsiva quando $\to'$ è la relazione che si sta definendo. La lettura è quella di [1]. Una transizione vale quando una sostituzione di valori concreti per le variabili della regola rende vera ogni premessa e rende la conclusione uguale alla transizione in esame. Seguiamo la convenzione che ogni variabile in una regola sia coperta, il suo valore determinato dalla configurazione a cui la regola si applica, o attraverso le premesse.

4 Stato del comando `stamp`

La libreria è l'oracolo per tutto ciò che accade dentro un'operazione. Quel che la semantica operazionale traccia è il dato che il comando porta fra un'operazione e l'altra, non i byte che la libreria muove. Raccogliamo quel dato in uno stato.

Uno stato σ è una funzione finita da nomi a valori, scritta come un insieme di legami. Lo stato del comando lega gli argomenti parsati e i risultati intermedi calcolati fin lì. Scriviamo $\sigma(x)$ per il valore legato a x, indefinito quando x non è legato, e $\sigma[v/x]$ per lo stato che concorda con σ ovunque tranne che in x, dove assume il valore v. Sono lo stato e l'operatore di modifica di [1], sezione 4.

I nomi che il comando `stamp` usa sono questi. I nomi degli argomenti *src*, *out*, *pdf*, *interactive* contengono la riga di comando parsata, ossia la sorgente del watermark richiesta dall'utente, il percorso di output se dato, se è stato chiesto un PDF, e se è stato chiesto l'inserimento interattivo. I nomi di lavoro t, img, t', o contengono il template aperto, l'immagine di sostituzione, il template dopo lo scambio, e il percorso di output risolto. Un nome è legato solo quando la sua fase l'ha prodotto, perciò $\sigma(t')$ è indefinito finché lo scambio non è stato eseguito.

La sorgente *src* è una delle forme che il comando accetta, un percorso di immagine $\mathsf{image}(p)$, una lista di righe di watermark $\mathsf{text}(\ell)$, un nome di preset $\mathsf{preset}(k)$, o l'assenza di tutte queste, $\bot$. Astraiamo su quale sia tranne dove una regola deve distinguerle.

5 Configurazioni e codici d'uscita

Una configurazione del comando `stamp` accoppia una fase a uno stato,

$$\langle \mathsf{ph}, \sigma \rangle \in \Gamma,$$

dove ph nomina il punto che il comando ha raggiunto. Le fasi, nell'ordine in cui il comando le visita, sono $\mathsf{resolve}$ (fissa la sorgente, chiedendola se richiesto), open (apre il template), build (costruisce l'immagine di sostituzione), swap (sostituisce l'immagine nel template), save (scrive il risultato), e pdf (produce un PDF se richiesto). La configurazione iniziale di un'esecuzione su argomenti parsati σ_0 è $\langle \mathsf{resolve}, \sigma_0 \rangle$.

Le configurazioni terminali sono i codici d'uscita,

$$T = \{\, \mathsf{esci}\, n \mid n \in \{0, 1, 3, 4, 5, 6, 7\} \,\}.$$

Un'esecuzione termina quando ne raggiunge uno. I codici non sono etichette arbitrarie. Ciascuno segna un modo distinto in cui il comando si può fermare, e chi lo invoca dirama su di essi. Il codice 0 è successo. Il codice 1 è errore d'uso, nessuna sorgente data o una malformata. Il codice 3 è un template illeggibile. I codici 4, 5, 6 vengono direttamente dalle eccezioni della libreria che lo scambio può sollevare, immagine non trovata, corrispondenza ambigua, e formato non combaciante. Il codice 7 è una conversione PDF fallita dopo che un documento è già stato scritto. La tabella alla fine della Sezione 6 li raccoglie.

6 Regole di transizione per `stamp`

Definiamo $\to_{\mathrm{st}}$, la relazione di transizione del comando `stamp`, una fase per volta. Ogni fase contribuisce una regola per il caso in cui procede e una o più regole per i casi in cui si ferma con un codice. Le premesse che menzionano la libreria usano gli stessi nomi della specifica assiomatica, e una premessa della forma "*op* solleva E" abbrevia la condizione sotto cui l'operazione di libreria segnala l'eccezione E.

Resolve. Il comando fissa per prima cosa la sorgente del watermark, poi quando è stato richiesto l'inserimento interattivo e nessuna sorgente è stata data sulla riga di comando, il comando legge righe dal terminale. La lettura produce una lista ℓ, e sulla lista vuota il comando si ferma con un errore d'uso. Scriviamo $\mathsf{prompt} \to \ell$ per la lettura interattiva che produce ℓ.

$$\frac{\sigma(\mathit{interactive}) = \mathrm{tt} \qquad \sigma(\mathit{src}) = \bot \qquad \mathsf{prompt} \to \ell \qquad \ell \neq []}{\langle \mathsf{resolve}, \sigma \rangle \to_{\mathrm{st}} \langle \mathsf{open}, \sigma[\mathsf{text}(\ell)/\mathit{src}] \rangle}$$

$$\frac{\sigma(\mathit{interactive}) = \mathrm{tt} \qquad \sigma(\mathit{src}) = \bot \qquad \mathsf{prompt} \to []}{\langle \mathsf{resolve}, \sigma \rangle \to_{\mathrm{st}} \mathsf{esci}\, 1}$$

Quando una sorgente è presente, il comando prosegue invariato. Quando nessuna è presente e l'inserimento interattivo non è stato richiesto, si ferma con lo stesso codice d'uso.

$$\frac{\sigma(\mathit{src}) \neq \bot}{\langle \mathsf{resolve}, \sigma \rangle \to_{\mathrm{st}} \langle \mathsf{open}, \sigma \rangle} \qquad \frac{\sigma(\mathit{src}) = \bot \qquad \sigma(\mathit{interactive}) = \mathrm{ff}}{\langle \mathsf{resolve}, \sigma \rangle \to_{\mathrm{st}} \mathsf{esci}\, 1}$$

Open. Il comando apre il template al percorso che gli è stato dato. Il successo lega il template aperto a t e risolve il percorso di output o, dall'argomento esplicito quando presente e altrimenti derivato dal percorso del template tramite la funzione *derive*. Il fallimento dell'apertura, sia che il file manchi sia che non sia un archivio valido, ferma il comando con codice 3.

$$\frac{\mathsf{open}(\sigma(\mathit{template})) \to t \qquad o = \mathit{outpath}(\sigma)}{\langle \mathsf{open}, \sigma \rangle \to_{\mathrm{st}} \langle \mathsf{build}, \sigma[t/t,\ o/o] \rangle} \qquad \frac{\mathsf{open}(\sigma(\mathit{template})) \text{ solleva } \mathsf{BadZip} \vee \mathsf{OSError}}{\langle \mathsf{open}, \sigma \rangle \to_{\mathrm{st}} \mathsf{esci}\, 3}$$

dove $\mathit{outpath}(\sigma) = \sigma(\mathit{out})$ quando out è legato e $\mathit{derive}(\sigma(\mathit{template}))$ altrimenti.

Build. Il comando costruisce l'immagine di sostituzione dalla sorgente. Una sorgente di percorso è letta dal disco, una sorgente di testo è resa in un'immagine di watermark, un preset si espande in un testo fisso ed è poi reso. La costruzione può fallire su una sorgente malformata, per esempio un percorso `--image` che non esiste, e quel fallimento è un errore d'uso.

$$\frac{\mathsf{build}(\sigma(src)) \to img}{\langle \mathsf{build}, \sigma\rangle \to_{\mathrm{st}} \langle \mathsf{swap}, \sigma[img/img]\rangle} \qquad \frac{\mathsf{build}(\sigma(src)) \text{ solleva } \mathsf{ValueError} \vee \mathsf{FileNotFound}}{\langle \mathsf{build}, \sigma\rangle \to_{\mathrm{st}} \mathsf{esci}\,1}$$

Swap. Il comando sostituisce l'immagine combaciante nel template, producendo un nuovo template t'. È la fase che eredita i fallimenti differenziati della libreria. Uno scambio riuscito passa a save.

$$\frac{\mathsf{replace}(\sigma(t), \sigma(img)) \to t'}{\langle \mathsf{swap}, \sigma\rangle \to_{\mathrm{st}} \langle \mathsf{save}, \sigma[t'/t']\rangle}$$

Le tre eccezioni della libreria si mappano su tre codici. Nessuna corrispondenza per l'immagine richiesta ferma con 4, più di un candidato quando il matcher ne richiede uno solo ferma con 5, e una sostituzione il cui formato differisce dall'immagine che rimpiazzerebbe ferma con 6.

$$\frac{\mathsf{replace}(\sigma(t), \sigma(img)) \text{ solleva } \mathsf{ImageNotFound}}{\langle \mathsf{swap}, \sigma\rangle \to_{\mathrm{st}} \mathsf{esci}\,4} \qquad \frac{\mathsf{replace}(\sigma(t), \sigma(img)) \text{ solleva } \mathsf{MultipleImages}}{\langle \mathsf{swap}, \sigma\rangle \to_{\mathrm{st}} \mathsf{esci}\,5}$$

$$\frac{\mathsf{replace}(\sigma(t), \sigma(img)) \text{ solleva } \mathsf{FormatMismatch}}{\langle \mathsf{swap}, \sigma\rangle \to_{\mathrm{st}} \mathsf{esci}\,6}$$

Save. Il comando scrive il nuovo template al percorso di output risolto. La scrittura si assume riuscita, la cartella essendo creata quando assente, perciò la fase ha una sola regola. Quando nessun PDF è stato richiesto questo è l'ultimo passo e il comando ha successo. Altrimenti passa a pdf.

$$\frac{\sigma(pdf) = \mathrm{ff} \qquad \mathsf{save}(\sigma(t'), \sigma(o)) \to ok}{\langle \mathsf{save}, \sigma\rangle \to_{\mathrm{st}} \mathsf{esci}\,0} \qquad \frac{\sigma(pdf) = \mathrm{tt} \qquad \mathsf{save}(\sigma(t'), \sigma(o)) \to ok}{\langle \mathsf{save}, \sigma\rangle \to_{\mathrm{st}} \langle \mathsf{pdf}, \sigma\rangle}$$

PDF. Quando un PDF è stato richiesto, il comando converte il documento scritto attraverso LibreOffice. Una conversione riuscita chiude l'esecuzione con successo, una fallita con codice 7. Il documento intermedio è già stato scritto a questo punto, ed è il motivo per cui un fallimento del PDF ha un codice proprio invece di confondersi con uno precedente, chi invoca sa che il `.docx` esiste anche quando il `.pdf` non esiste.

$$\frac{\mathsf{topdf}(\sigma(o)) \to ok}{\langle \mathsf{pdf}, \sigma\rangle \to_{\mathrm{st}} \mathsf{esci}\,0} \qquad \frac{\mathsf{topdf}(\sigma(o)) \text{ solleva } \mathsf{PDFConversion}}{\langle \mathsf{pdf}, \sigma\rangle \to_{\mathrm{st}} \mathsf{esci}\,7}$$

I codici d'uscita insieme.

Codice	Raggiunto da
0	save o pdf, su successo
1	resolve (sorgente assente o vuota), build (sorgente malformata)
3	open, template mancante o archivio non valido
4	swap, nessuna immagine combaciante
5	swap, più di un'immagine combaciante
6	swap, formato della sostituzione diverso dal bersaglio
7	pdf, conversione fallita dopo che il documento è stato scritto

7 Esecuzioni

Un'esecuzione del comando su argomenti parsati σ_0 è la derivazione che parte da $\langle \mathsf{resolve}, \sigma_0 \rangle$ e procede per $\to_{\mathrm{st}}$ finché raggiunge una terminale. Due proprietà della relazione seguono per ispezione delle regole della Sezione 6.

Proprietà 7.1 (Progresso)**.** *Ogni configurazione non terminale raggiungibile ha un successore. Per ogni fase le regole coprono i suoi casi senza sovrapposizione, perciò si applica una regola e nessuna configurazione è bloccata.*

Le fasi formano un ordine fisso, resolve, open, build, swap, save, e poi pdf solo quando un PDF è stato richiesto. Nessuna regola torna a una fase precedente. Un'esecuzione visita perciò ciascuna fase al più una volta e raggiunge una terminale in un numero di passi limitato dal numero delle fasi.

Proprietà 7.2 (Terminazione)**.** *Ogni esecuzione è finita e termina in una configurazione terminale.*

Una conseguenza lega il quadro operazionale a quello assiomatico. Quando un'esecuzione raggiunge esci 0, è passata per swap con la regola di successo, il che significa che l'operazione replace della libreria ha restituito un template. La postcondizione di quell'operazione, dal documento gemello, è ciò che garantisce che il documento scritto preservi il nameset dell'archivio. Il codice d'uscita 0 è, in questo senso, l'ombra operazionale della postcondizione assiomatica, il comando ha successo precisamente quando il contratto su cui si appoggia è stato soddisfatto.

8 Il sottocomando `inspect`

Il sottocomando `inspect` legge un template ed elenca le immagini che contiene, senza cambiare nulla. La sua semantica richiede solo due fasi, open e list, e tre dei codici. Apre il template come fa `stamp`, con lo stesso codice 3 in caso di fallimento, poi elenca.

$$\frac{\mathsf{open}(\sigma(\mathit{template})) \to t \qquad \mathsf{images}(t) \to \mathit{imgs}}{\langle \mathsf{open}, \sigma \rangle \to_{\mathrm{st}} \mathsf{esci}\,0} \qquad \frac{\mathsf{open}(\sigma(\mathit{template})) \text{ solleva } \mathsf{BadZip} \vee \mathsf{OSError}}{\langle \mathsf{open}, \sigma \rangle \to_{\mathrm{st}} \mathsf{esci}\,3}$$

L'elenco riesce sempre una volta che il template è aperto, che ci sia o meno un'immagine, perciò il comando in sola lettura ha solo il codice di successo e il fallimento dell'apertura. Non raggiunge mai i codici da 4 a 7, che appartengono allo scambio e alla conversione PDF che `inspect` non esegue.

9 Mappatura al codice

Le fasi corrispondono a tratti di `cmd_stamp` in `cli.py`, nell'ordine in cui la funzione li esegue. La tabella nomina la funzione invece di una riga, per la ragione data nel documento gemello, i numeri di riga si corrompono mentre i nomi no.

Fase	Locazione nel codice	Codici d'uscita
resolve	`cmd_stamp` (risoluzione sorgente)	1
open	`cmd_stamp` → `Template.open`	3
build	`cmd_stamp` → `_build_replacement_image`	1
swap	`cmd_stamp` → `Template.replace_image`	4, 5, 6
save	`cmd_stamp` → `Template.save`	—
pdf	`cmd_stamp` → `to_pdf`	7

I codici d'uscita nella tabella sono gli interi che `cmd_stamp` restituisce, e il dispatcher `main` li passa a `sys.exit` invariati, salvo due codici fuori da questa semantica, 2 per una riga di comando non parsabile, che `argparse` restituisce prima che `cmd_stamp` giri, e 130 per un'interruzione.

L'insieme dei codici non è tenuto allineato a mano. Il test `test_spec_crossref.py` estrae dal sorgente ogni intero restituito da `cmd_stamp` e `cmd_inspect` e lo confronta con i codici qui tabulati, in entrambe le edizioni linguistiche. Un codice aggiunto o tolto nel codice senza il documento, o una divergenza fra le due edizioni, fa fallire la build. È la controparte operazionale del legame `spec=` che la specifica assiomatica usa.

Riferimenti bibliografici

[1] R. Barbuti, P. Mancarella, F. Turini. *Elementi di Semantica Operazionale.* Lecture notes for Fondamenti di Programmazione, B.Sc. in Computer Science, University of Pisa, A.A. 2004/05. `https://pages.di.unipi.it/mancarella/FP/materiale/nuovasemop.pdf`

[2] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981. Reprinted in *J. Log. Algebr. Program.*, 60–61:17–139, 2004.

[3] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction.* MIT Press, Cambridge, MA, 1993. ISBN 0-262-73103-7. `https://direct.mit.edu/books/monograph/4338/The-Formal-Semantics-of-Programming-LanguagesAn`

[4] P. Mancarella. *Sintassi dei Linguaggi di Programmazione.* Dispensa per il corso di Fondamenti di Programmazione, Università di Pisa. `https://pages.di.unipi.it/mancarella/FP/materiale/syntax.pdf`