Alma Mater Studiorum - Università di Bologna
SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Verso una teoria delle differenze tra documenti semi-strutturati

Relatore: Chiar.mo Prof. Fabio Vitali.
Correlatori: Dott. Angelo Di Iorio, Dott. Gioele Barabucci.

Anno Accademico 2012-2013

Sessione I

Andrea Aquino

Indice

Citazioni

"Trovarsi davanti a un pazzo sapete che significa? Trovarsi davanti a uno che vi scrolla dalle fondamenta tutto quanto avete costruito in voi, attorno a voi, la logica di tutte le vostre costruzioni."

Luigi Pirandello, "Enrico IV"

"Parlavano certo di me, perché, come mi videro entrare, esclamarono a un tempo: ≪Oh, eccolo qua!≫

E poiché erano due a vedermi entrare, mi venne la tentazione di voltarmi a cercare l'altro che entrava con me, pur sapendo bene che il caro Vitangelo del mio paterno Quantorzo non solo era anch'esso in me come il Gengè di mia moglie Dida, ma che io tutto quanto, per Quantorzo, altri non ero che il suo caro Vitangelo, proprio come per Dida altri che il suo Gengè. Due, dunque, non agli occhi loro, ma soltanto per me che mi sapevo per quei due uno e uno; il che per me, non faceva un piú ma un meno, in quanto voleva dire che ai loro occhi, io come io, non ero nessuno."

Luigi Pirandello, "Uno, nessuno e centomila"

1 Introduzione

Da tempo immemore l'uomo si è interrogato sul concetto di uguaglianza, di similarità e di diversità. L'uguaglianza è un concetto cardine altresì presente nel motto nazionale della Repubblica Francese

"Liberté, Égalité, Fraternité".

L'uguaglianza tra gli uomini indipendentemente dal sesso, lo stato sociale, la religione è un diritto ad oggi riconosciuto ed inviolabile. Ma è proprio dalla parola "indipendentemente" che tale definizione trae la propria forza. Filosoficamente non ha senso parlare di uguaglianza tra due oggetti (così come tra due individui) distinti. Due mele sono diverse perchè non sono la stessa mela, due persone sono diverse perchè non sono la stessa persona. I primi versi della "Divina Commedia" citati da Dante Alighieri e da un noto critico lettario sono veramente gli stessi versi?

A meno di un certo numero di caratteristiche è veramente possibile dichiarare che due cose sono uguali l'una all'altra anche se fisicamente o etimologicamente differenti (nella forma o nel contenuto). Se ci limitassimo a considerare il numero dei lati di una figura geometrica, un quadrato ed un rombo sarebbero indistinguibili. Sarebbero invece diversi se considerassimo anche il loro angolo di inclinazione.

Poichè la percezione umana è limitata ad un contesto finito in generale è infruttuoso considerare diversi oggetti che, in pratica, sono molto simili tra di loro. Anche se impropriamente è prassi dire uguali cose che differiscono per pochi trascurabili dettagli. Il livello di dettaglio considerato trascurabile è fortemente contestuale. Due persone, anche se fratelli gemelli, sono spesso considerate diverse l'una dall'altra. Due cerchi del medesimo diametro se anche disegnati in posizioni diverse su un foglio di carta sono invece spesso considerati uguali.

Il presente elaborato di tesi intende indagare prima filosoficamente e poi matematicamente il concetto di *differenza* tra due oggetti e discutere un formalismo teorico che fornisce gli strumenti per comprendere quali sono

effettivamente tali differenze. Questo formalismo prende nome di **teoria delle differenze**.

L'interesse per una simile teoria è effettivamente smisurato. In particolare, nell'ambito della scienza dell'informazione, essere in grado di comprendere cosa distingue due documenti è essenziale per poter sviluppare sistemi di versionamento affidabili e potenti. Tuttavia le potenzialità di tale formalismo non sono esclusivamente limitate al mondo informatico.

I dipendenti del parlamento di uno Stato confrontano giorno dopo giorno migliaia di proposte di legge che differiscono per pochi tratti di punteggiatura ma che sono, nel significato, profondamente diverse. Possedere un sistema in grado di sottolineare le differenze rilevanti (non solo di carattere testuale ma anche ad un più alto livello livello di astrazione) tra tali documenti garantirebbe un risparmio incredibile tanto in termini di tempo quanto in termini di forza lavoro e quindi di denaro.

Al fine di sviluppare una teoria delle differenze che sia implementabile nella forma di un algoritmo per un calcolatore elettronico, il presente testo investiga in principio le proprietà di strutture dati intrinsecamente infinite quali gli insiemi numerabili ed i grafi infiniti. Le primissime sezioni indirizzano il problema di individuare le caratteristiche che rendono tali strutture rappresentabili con un computer.

Alcuni dei più celebri problemi sui grafi quali ad esempio determinare l'esistenza di un cammino tra due nodi, individuare effettivamente un cammino tra due nodi ed individuare il cammino minimo tra due nodi si complicano notevolmente quando la dimensione del grafo sul quale operano non ha limite. Anche sotto ipotesi molto forti quali la forte connessione di un grafo infinito è impossibile sviluppare algoritmi che, nel caso generale, siano in grado di individuare il cammino più breve tra due nodi. Le sezioni introduttive, con il fermo appoggio delle nozioni evidenziate in appendice, forniscono un certo numero di dimostrazioni di quanto sia effettivamente calcolabile in tale contesto e di quanto non lo sia affatto.

Rammentando che il contesto applicativo cardine dell'analisi delle differenze cui questo documento è mirato è quello dei documenti di testo, la sezione dal titolo "Documenti" definisce matematicamente in modo molto semplice il concetto di documento semi-strutturato. Un documento semi-strutturato è un documento il cui testo sottostà ad una data gerarchia. Si potrebbe pensare ad un documento semi-strutturato come ad un albero le cui foglie sono stringhe di testo ed i cui nodi esplicitano la gerarchia del documento stesso. A tutti gli effetti un documento testuale classico è un caso particolare di documento semi-strutturato (privo di gerarchia).

E' inoltre scopo di tale sezione individuare un insieme non ambiguo di documenti semi-strutturati ben formati e discutere delle alternative per rappresentare univocamente ogni singolo nodo di un documento specifico.

Il formato dati Extensible Markup Language (XML) ha reso celebre questa categoria di documenti ed è ad oggi lo standard più diffuso per lo scambio di informazioni sulla rete internet mediante il protocollo Hypertext Transfer Protocol (HTTP). La crescente mole di documenti semi-strutturati disponibili rende evidente la necessità di sviluppare un algoritmo in grado di confrontarli mettendone in risalto le similarità e le differenze.

Oggigiorno esistono svariati algoritmi del genere (detti algoritmi di diffing), tra i più famosi è doveroso citare *diffX*, *DocTreeDiff*, *Meaningful Change Tool*, *Ndiff* e *XyDiff*. Sfortunatamente questi algoritmi sono ancora piuttosto limitati. Il numero e la qualità delle differenze che sono in grado di individuare varia ampiamente da algoritmo ad algoritmo.

Come discusso precedentemente il tipo di differenze cui un utente è interessato quando analizza due documenti distinti dipende fortemente dal contesto applicativo. Sebbene gli algoritmi citati siano estremamente generali, lo sono a scapito della qualità e dell'interesse nelle differenze che sono in grado di individuare.

La sezione "Verso una teoria delle differenze" suggerisce gli espedienti atti a sviluppare un algoritmo di diffing che sia completamente generale in termini del tipo di differenze tra documenti che è in grado di individuare.

Sfruttando una serie di nozioni tipiche della teoria dei linguaggi quali ad esempio le funzioni di ordine superiore sono state sviluppate delle strutture dati infinite (finitamente rappresentabili) il cui scopo è quello di semplificare la ricerca delle caratteristiche che distinguono due documenti. Facendo uso dei risultati della sezione "Cenni sulla calcolabilità di algoritmi su strutture infinite" vengono discusse le ulteriori ipotesi necessarie a rendere calcolabile l'algoritmo di analisi delle differenze oggetto di questa ricerca.

Inoltre è scopo di questa sezione discutere della complessità (in termini del tempo di calcolo) di tale algoritmo. Poichè questi opera principalmente su un grafo infinito la comune definizione di complessità asintotica non è sufficiente a stimarne il tempo di esecuzione in nessun caso. Anche volendo limitarsi a considerare la rappresentazione finita di tale grafo il tempo di calcolo è completamente svincolato da quest'ultima. Viene pertanto presentata una stima della complessità in tempo dell'algoritmo proposto in funzione della dimensione dell'output.

Nella sezione "Un algoritmo di diffing parametrico" viene discusso ad altissimo livello il funzionamento dell'algoritmo parametrico di analisi delle differenze **Paradiff**. Tale sezione discute le tecnologie più adatte allo sviluppo di un algoritmo del genere e suggerisce una serie di espedienti atti alla codifica delle strutture dati sulle quali è basata l'intera teoria delle differenze. Vengono infine analizzati alcuni dei limiti più evidenti dell'algoritmo presentato lasciando spazio alla ricerca di più sofisticate tecniche che rendano Paradiff equiparabile (in termini prestazionali) ai moderni e più efficienti algoritmi di diffing.

Occorre considerare che, poichè la complessità teorica dell'algoritmo Paradiff non è in alcun modo calcolabile con metodologie classiche, le sue prestazioni possono essere stimate esclusivamente a mezzo di test su campioni di documenti con caratteristiche prefissate. Ciò rende Paradiff inconfrontabile a livello teorico con gli algoritmi di diffing citati in precedenza. Purtroppo questo è uno dei limiti più evidenti del prodotto della presente tesi.

L'appendice di questo testo raccoglie le nozioni matematiche di base che fissano in via stabile la terminologia ed il significato dei costrutti discussi in ogni altra sezione. Ivi sono raccolti molti concetti chiave della teoria della calcolabilità a partire dalla tesi di Church-Turing fino alla formulazione e alla dimostrazione di indecidibilità del problema della terminazione. I risultati discussi permettono al lettore di comprendere alcuni dei dettagli più sottili di questo lavoro.

In conclusione questo elaborato è rivolto a chiunque sia interessato a sviluppare nuovi sistemi di versionamento che siano in grado di individuare differenze arbitrariamente sottili tra documenti di qualsiasi genere; a chiunque voglia realizzare sistemi per il rapido confronto di documenti; a chiunque voglia confrontare oggetti di un determinato tipo evidenziandone le similarità e le dissimilarità.

La teoria delle differenze è uno strumento teorico di incredibile potenziale. Quanto sarà effettivamente in grado di influire sulla modernità sapranno dirlo il tempo ed i risultati in tale contesto, tanto teorici quanto pratici, che verranno.

2 Cenni sulla calcolabilità di algoritmi su strutture infinite

Giorno dopo giorno l'uomo esplora il mondo che lo circonda. Foreste, oceani e montagne si stagliano maestosi sotto lo sguardo fermo dell'osservatore. I sensi umani permettono di carpirne le caratteristiche rilevanti quali la dimensione, il colore, la consistenza. E' affascinante notare come la percezione umana sia riservata, nella quasi totalità dei casi, alle cose finite. Una goccia di acqua contiene circa $1.66 * 10^{21}$ molecole ed un numero ancora superiore di atomi, ma tale numero è finito; sarebbe virtualmente possibile contare ogni singolo atomo ivi presente sebbene lo sforzo richiesto sarebbe immenso ed i benefici da esso derivanti pressochè nulli. Eppure basta volgere il proprio sguardo all'universo, pensare a quante stelle vi sono adesso e quante ve ne saranno, lasciarsi intristire dallo scorrere inesorabile del tempo per rendersi conto che le cose finite sono mera finzione.

I più famosi problemi di interesse pratico sono fortemente legati alla realtà in cui viviamo e di conseguenza relativi ad oggetti finiti. L'idea di dedicarsi ad una mansione per un lasso di tempo illimitato quando si è consci del fatto che la vita è invece destinata a terminare fa paura. E' proprio perchè ognuno è intuitivamente conscio di tale limite biologico che qualsiasi problema si incontri, che abbia a che fare con pochi oggetti o con una interminabile sequenza di numeri, va risolto in tempo finito.

La presente sezione si occupa di definire un certo numero di problemi relativi a collezioni infinite di oggetti. In particolare si discuterà di **insiemi infiniti** e di **grafi infiniti**. Algoritmi estremamente efficaci per risolvere problemi sulle controparti finite di tali collezioni falliscono miseramente il loro scopo quando la dimensione del loro input non ha limite. Alle volte si è in grado di risolvere tali problemi in un lasso di tempo finito, alle volte ciò è del tutto impossibile.

2.1 Rappresentabilità finita

Un calcolatore moderno, per quanto potente o costoso, è dotato di memoria finita. Tra l'altro la memoria principale[1] è ad oggi ancora particolarmente poco capiente. E' quindi impensabile poterla utilizzare per memorizzare una struttura infinita.

Tuttavia, in generale, avere accesso ad un'infinità di oggetti contemporaneamente non è pratico nè utile. Se tali oggetti fossero accessibili "on-demand" in un qualche ordine ciò non dovrebbe in alcun modo rivelarsi limitante. Si pensi ad esempio ad un insieme A infinito. Rappresentare tutti i suoi elementi nella memoria di un computer è impossibile ma potrebbe invece essere possibile rappresentarli mediante una funzione $f : \mathbb{N} \to A$ che, su input n restituisce l'ennesimo elemento dell'insieme A. Ad esempio l'insieme infinito dei quadrati perfetti

$$\{ e \in \mathbb{N} \mid \sqrt{e} \in \mathbb{N} \},$$

è rappresentabile mediante la funzione

$$f(n) = n^2.$$

Per comprendere al meglio l'espediente di **rappresentare finitamente** un insieme infinito con una funzione si pensi ad un medico generale dell'ospedale Maggiore di Bologna. Il medico, seduto alla scrivania del suo studio, chiama i suoi (infiniti) pazienti uno alla volta piuttosto che stare tra di loro comunicando con tutti contemporaneamente. Così come ci sono sempre nuovi pazienti, è sempre possibile estrarre nuovi elementi da un insieme infinito.

[1] Detta Random-Access-Memory (RAM), la memoria principale è uno strumento in grado di memorizzare dati per periodi di tempo ridotti. E' destinata all'esecuzione dei programmi.

2.2 Numerabilità di insiemi

Definizione 2.1 (Insieme numerabile)**.**
Un insieme S si dice **numerabile** (o **enumerabile**) se è finito oppure se esiste (ossia se è rappresentabile mediante) una funzione biettiva $f : \mathbb{N} \to S$. La funzione f è detta **funzione di enumerazione** di S.

Moltissimi insiemi sono numerabili. L'insieme dei numeri naturali $\mathbb{N}$ è numerabile, così come lo è l'insieme dei numeri razionali $\mathbb{Q}$ e l'insieme Σ^* delle stringhe su un dato alfabeto finito Σ.

A scapito di quanto si potrebbe credere non tutti gli insiemi infiniti sono numerabili. Un celebre controesempio ad opera di Georg Cantor è l'insieme dei numeri reali $\mathbb{R}$. Ciò significa che un vastissimo numero di insiemi infiniti di largo interesse non sono banalmente rappresentabili finitamente con una funzione che mappi numeri naturali negli elementi dell'insieme e che sia in grado di restituirli tutti prima o poi. Nella pratica, tuttavia, gli insiemi infiniti numerabili trovano vasto impiego ed a tutti gli effetti rappresenteranno l'unico tipo di insiemi infiniti oggetto della presente tesi.

2.3 Appartenenza ad insiemi infiniti

Dato un insieme A numerabile rappresentato mediante la sua funzione di enumerazione $f : \mathbb{N} \to A$ il problema di determinare se un dato elemento e appartiene ad A è indecidibile.

Dimostrazione.
Si supponga per assurdo che esista una Macchina di Turing $Member$ in grado di decidere l'appartenenza o la non appartenenza di un elemento e ad un insieme numerabile A (rappresentato mediante la sua funzione di enumerazione f). Il comportamento di $Member$ è esplicitato di seguito

$$Member(e, f) = \begin{cases} \text{termina accettando} & \text{se } \exists n.\ f(n) = e, \\ \text{termina rifiutando} & \text{altrimenti.} \end{cases}$$

Sia M' una Macchina di Turing che presa in input la codifica di un'altra Macchina di Turing $< M >$ ed una stringa w, simula M su input w. Se $M(w)$ accetta o rifiuta allora M' cancella tutto il contenuto del suo nastro ed accetta. Data la codifica di una Macchina di Turing arbitraria $< M >$ ed una stringa arbitraria w, la funzione g definita come segue enumera tutte le possibili configurazioni di $M'(< M >, w)$.

$$g(n) = \text{configurazione ottenuta simulando } M'(< M >, w) \text{ per } n \text{ passi.}$$

Si noti che $M'(< M >, w)$ termina accettando nella configurazione q_{accept} se e solamente se $M(w)$ termina accettando o rifiutando. Ma allora

$$Member(q_{accept}, g) = \begin{cases} \text{termina accettando} & \text{se } M(w) \text{ termina,} \\ \text{termina rifiutando} & \text{se } M(w) \text{ non termina.} \end{cases}$$

Ciò è evidentemente assurdo in quanto il problema della terminazione non è decidibile. Pertanto la Macchina di Turing $Member$ non può esistere. □

2.4 Un'introduzione ai grafi infiniti

Mentre un insieme infinito numerabile è rappresentabile mediante una funzione di enumerazione, la rappresentazione finita di un grafo infinito è più sottile. Ai fini della presente trattazione si è interessati a rappresentare finitamente solo grafi infiniti il cui insieme dei nodi sia numerabile. Una tecnica conveniente per far ciò consiste nel rappresentare un grafo infinito della forma $G = (V, E)$ con due funzioni di enumerazione f_V ed f_E.

La funzione f_V enumera i nodi del grafo mentre la funzione di ordine superiore f_E prende in input un nodo del grafo e restituisce un enumeratore dei nodi ad esso direttamente connessi (i suoi vicini). Si noti che l'enumeratore dei vicini di un dato nodo u restituito dalla funzione f_E può a sua volta restituire il simbolo speciale $\diamond$ qualora u non abbia nodi vicini oppure tutti i suoi vicini siano già stati enumerati.

Definizione 2.2 (Rappresentazione finita di un grafo infinito)**.**
La **rappresentazione finita di un grafo infinito** $G = (V, E)$ è una coppia (f_V, f_E) tale che

$$
\begin{aligned}
&f_V : \mathbb{N} \to V \text{ è biettiva,}\\
&\diamond \notin V,\\
&f_E : V \to \mathbb{N} \to V \cup \{ \diamond \},\\
&\text{se } (u, v) \in E \text{ allora } \exists! n.\ f_E(u)(n) = v,\\
&\text{se } u \in V \text{ ha } k \text{ vicini allora } \forall n \geq k.\ f_E(u)(n) = \diamond,\\
&\text{se } u \in V \text{ ha infiniti vicini allora } \forall n.\ f_E(u)(n) \neq \diamond.
\end{aligned}
$$

Esistono svariati problemi di largo interesse relativi ai grafi e quindi ai grafi infiniti. Tra questi spiccano per importanza il problema di determinare se esiste un cammino tra due nodi, il problema di individuare un cammino tra due nodi ed il problema di individuare il cammino più breve tra due nodi.

Poichè come dimostrato in precedenza non è possibile decidere se un elemento verrà o meno enumerato da una funzione di enumerazione allora non è nemmeno possibile determinare se un nodo appartiene ad un grafo infinito oppure no. Di conseguenza ogni successiva digressione su grafi o multigrafi infiniti sarà basata sull'assunto che gli eventuali nodi presi in esame siano noti appartenere ad un dato grafo.

2.4.1 Esistenza di un cammino in un grafo infinito

Una funzione che prenda in input la rappresentazione finita di un grafo infinito (f_V, f_E), due nodi u e v appartenenti a tale grafo e sia in grado di decidere se esiste un cammino tra tali nodi nel grafo non è calcolabile.

Dimostrazione.
Si supponga per assurdo che esista una Macchina di Turing *Path* che, presa in input la rappresentazione finita di un grafo infinito (f_V, f_E) e due nodi di tale grafo u e v, sia in grado di decidere se esiste o meno un cammino tra u

e v. Il comportamento di $Path$ è esplicitato di seguito

$$Path((f_V, f_E), u, v) = \begin{cases} \text{termina accettando} & \text{se esiste un cammino tra } u \text{ e } v, \\ \text{termina rifiutando} & \text{altrimenti.} \end{cases}$$

Sia $M' = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ una Macchina di Turing che presa in input la codifica di un'altra Macchina di Turing $< M >$ ed una stringa w, simula M su input w. Se $M(w)$ accetta o rifiuta allora M' cancella tutto il contenuto del suo nastro ed accetta. Sia δ_{conf} la funzione δ opportunamente modificata per prendere in input una configurazione piuttosto che uno stato ed un simbolo. Data la codifica di una Macchina di Turing arbitraria $< M >$ ed una stringa arbitraria w, è possibile definire le seguenti funzioni p e q

$$p(n) = \text{configurazione ottenuta simulando } M'(< M >, w) \text{ per n passi,}$$

$$q(c) = q'(n) = \begin{cases} c' & \text{se } \exists c'.\ \delta_{conf}(c) = c' \text{ e } n = 0, \\ \diamond & \text{altrimenti.} \end{cases}$$

La funzione p enumera tutte le possibili configurazioni della Macchina di Turing M'. La funzione q prende in input una configurazione e restituisce una nuova funzione q'. Tale funzione q' enumera come unico eventuale vicino di una data configurazione c quella configurazione c' che è indotta dalla funzione di transizione di M' quando tale Macchina di Turing si trova nella configurazione c.

Si noti che le funzioni p e q definiscono il grafo infinito delle configurazioni della Macchina di Turing M' nel quale esiste un arco tra due configurazioni se e solamente se la funzione di transizione di M' permette di transire dall'una all'altra.

E' chiaro che $M'(< M >, w)$ termina accettando nella configurazione q_{accept} se e solamente se $M(w)$ termina accettando o rifiutando. Ma allora se

c_0 è la configurazione iniziale di M' vale che

$$Path((p,q), c_0, q_{accept}) = \begin{cases} \text{termina accettando} & \text{se } M(w) \text{ termina,} \\ \text{termina rifiutando} & \text{se } M(w) \text{ non termina.} \end{cases}$$

Ciò è assurdo in quanto risolverebbe il problema della terminazione e pertanto la Macchina di Turing $Path$ non può esistere. □

2.4.2 Calcolo di un cammino in un grafo infinito connesso

La sezione precedente dimostra che non è possibile decidere se esiste un cammino tra due nodi in un grafo infinito rappresentato finitamente. Sotto l'assunzione che tale grafo sia fortemente connesso (ossia esista un cammino diretto tra ogni coppia di nodi) tale problema diventa triviale. Si apre dunque un ulteriore interrogativo. E' possibile per un calcolatore individuare un cammino tra due nodi (ossia restituire esplicitamente la sequenza di nodi facenti parte di tale cammino) in un grafo infinito fortemente connesso? La risposta è sorprendentemente si.

Si noti che i classici approcci algoritmici di visita in ampiezza (BFS) ed in profondità (DFS) falliscono in tale contesto. Di fatti l'approccio BFS prevede di analizzare in primis tutti i vicini del nodo di partenza, ma in un grafo infinito questi potrebbero essere infiniti. Allo stesso modo l'approccio DFS prevede di considerare il primo dei vicini del nodo di partenza e quindi il primo vicino di tale vicino e così via. Sfortunamente in un grafo infinito tale sequenza di nodi potrebbe non avere fine. La Figura 1 rappresenta due grafi infiniti fortemente connessi. Per snellire la rappresentazione grafica di tali grafi sono stati adoperati esclusivamente degli archi non orientati. Tuttavia si pensi all'arco non orientato tra due nodi u e v come alla coppia di archi orientati (u, v) e (v, u). In entrambi i grafi si è interessati ad individuare un cammino tra il nodo rosso ed il nodo blu; nel primo mediante la tecnica BFS, nel secondo mediante DFS. Entrambi gli approcci tuttavia non saranno mai in grado di raggiungere il risultato desiderato.

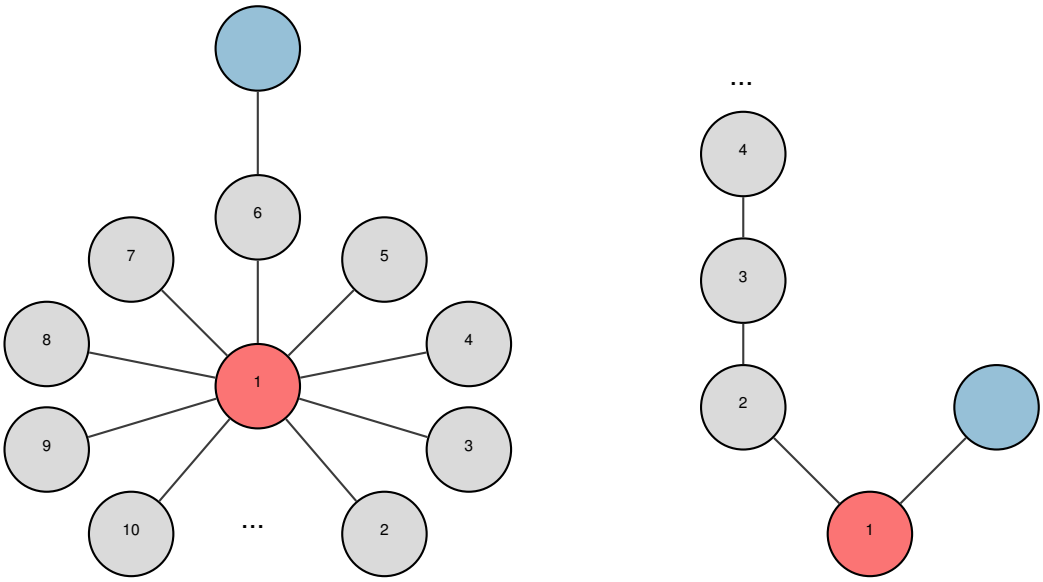

Figura 1: BFS e DFS falliscono su grafi infiniti connessi.

Una tecnica che invece serve lo scopo consiste nell'enumerare tutti i possibili cammini a partire dal nodo di partenza verso qualsiasi altro nodo. Poichè per ipotesi il grafo è fortemente connesso almeno un cammino tra il nodo di partenza e quello di destinazione esiste e quindi prima o poi verrà enumerato e restituito.

Un modo per enumerare tutti i cammini a partire da un determinato nodo u consiste nel tenere traccia dell'insieme dei nodi considerati fino ad un dato istante, in principio tale insieme contiene solo il nodo u. Passo dopo passo tale insieme viene modificato aggiungendo il primo vicino di ogni elemento in esso contenuto non ancora presente nell'insieme. E' importante che tale vicino venga scelto mediante la funzione di enumerazione degli archi del grafo f_E opportunamente istanziata sul nodo in questione al fine di avere la certezza che tutti i vicini di tale nodo verranno enumerati prima o poi.

Nonostante questa tecnica permetta di individuare un cammino tra due nodi in un grafo infinito fortemente connesso non fornisce nessuna garanzia sulla dimensione di tale cammino. Si pensi ad esempio ad un grafo infinito in cui tra due nodi u e v esistono due cammini distinti. Il primo consta di

centomila nodi, il secondo di uno soltanto. Purtroppo l'algoritmo discusso potrebbe enumerare e restituire in prima battuta il cammino più lungo al posto dell'altro più breve e semplice.

2.4.3 Il cammino minimo in un grafo infinito connesso non è calcolabile

Data l'esistenza di un algoritmo per individuare un cammino tra due nodi di un grafo infinito connesso si potrebbe credere che ne esista un altro in grado di individuare il cammino di dimensione minima tra questi. Malauguratamente un algoritmo che calcoli **il cammino minimo** tra due nodi appartenenti ad un grafo infinito connesso non può esistere. Ciò non dovrebbe sorprendere il lettore già abituato dal presente elaborato a questo genere di risultati (di carattere negativo). Al fine di provare quanto detto è necessario in primo luogo dimostrare il seguente lemma.

Non esiste una Macchina di Turing $First$ che, presa in input la codifica di un'altra Macchina di Turing $< M >$, restituisce il più piccolo numero naturale n tale per cui $M(n)$ termina.

Dimostrazione.
Si supponga per assurdo che esista una Macchina di Turing $First$ che, presa in input la codifica di un'altra Macchina di Turing $< M >$, restituisca il più piccolo numero naturale n tale per cui $M(n)$ termina, ossia

$$First(< M >) = \text{il più piccolo } n \in \mathbb{N}.\ M(n) \text{ termina.}$$

Allora sarebbe possibile realizzare un'altra Macchina di Turing D tale che $D(< M >, w) = < D'_{M,w} >$. Tale Macchina di Turing prende in input la codifica di un'altra Macchina di Turing $< M >$ ed è una stringa w e restituisce la codifica di una nuova Macchina di Turing $D'_{M,w}$ definita di

seguito

$$D'_{M,w}(n) = \begin{cases} \text{termina accettando} & \text{se } n = 0 \text{ e } M(w) \text{ termina,} \\ \text{termina accettando} & \text{se } n = 1, \\ \text{non termina} & \text{altrimenti.} \end{cases}$$

Si noti che $D'_{M,w}(0)$ termina se e solamente se $M(w)$ termina. Inoltre $D'_{M,w}(1)$ termina. Di conseguenza il più piccolo input su cui $D'_{M,w}$ termina è 0 se $M(w)$ termina, è invece 1 se $M(w)$ non termina.

Ma quindi, data una Macchina di Turing M arbitraria ed una stringa di testo w arbitraria,

$$First(D(< M >, w)) = \begin{cases} 0 & \text{se } M(w) \text{ termina,} \\ 1 & \text{se } M(w) \text{ non termina.} \end{cases}$$

Ciò è evidentemente assurdo in quanto risolverebbe il problema della terminazione che è noto essere indecidibile. Quindi la Macchina di Turing $First$ non può esistere. □

A questo punto è possibile dimostrare che una Macchina di Turing Min che prenda in input la rappresentazione finita di un grafo infinito connesso, due nodi u e v appartenenti a tale grafo e restituisca i nodi facenti parte del cammino minimo tra u e v, non può esistere.

Dimostrazione. Si supponga per assurdo che esista una Macchina di Turing Min che, presa in input la rappresentazione di un grafo infinito connesso (f_V, f_E) e due nodi di tale grafo u e v, sia in grado di calcolare il cammino minimo tra u e v.

$$Min((f_V, f_E), u, v) = \text{cammino minimo tra } u \text{ e } v.$$

Sia M' una Macchina di Turing che presa in input la codifica di un'altra Macchina di Turing $< M >$ ed una stringa w, simula M su input w. Se

$M(w)$ accetta o rifiuta allora M' cancella tutto il contenuto del suo nastro ed accetta.

Si consideri quindi il grafo delle computazioni parallele della Macchina di Turing M' descritto in Figura 2.

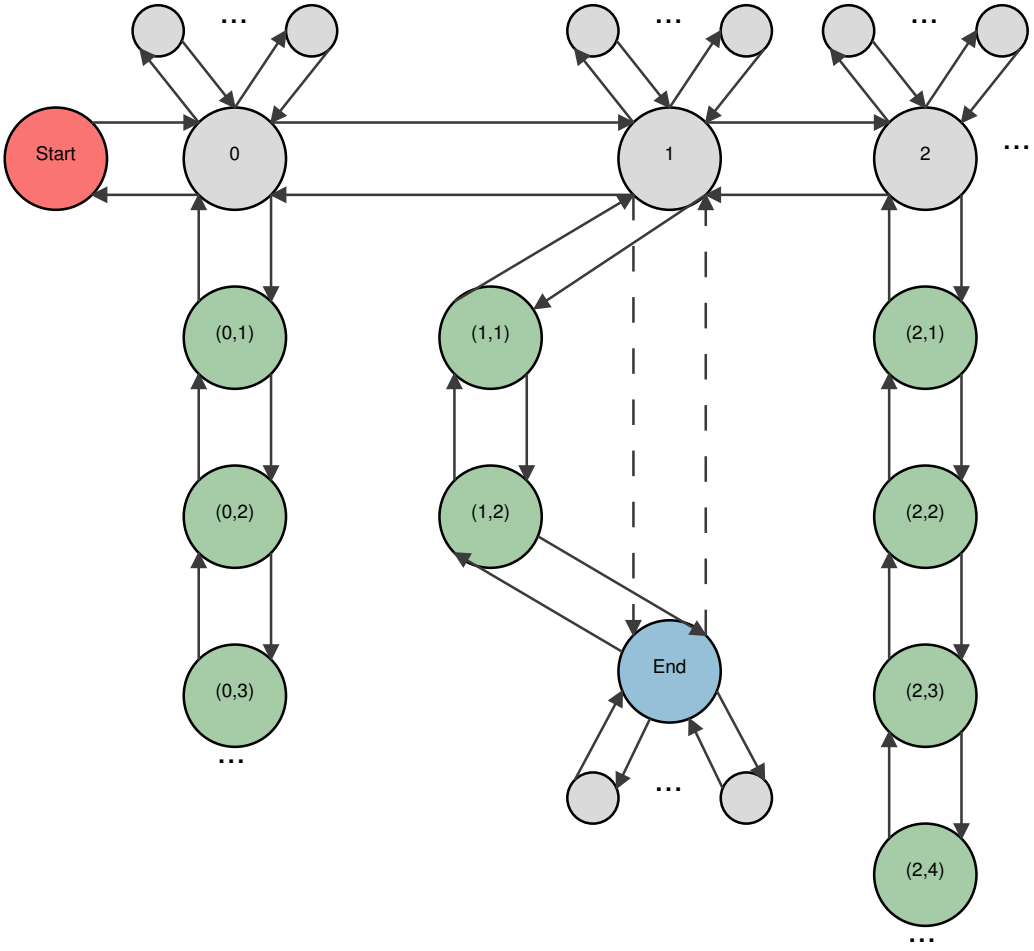

Figura 2: Grafo delle computazioni parallele di una Macchina di Turing.

Tale grafo è composto da un nodo di partenza (in rosso, etichettato "Start") e da infiniti nodi etichettati con i numeri naturali. Intuitivamente ognuno di questi nodi rappresenta un input diverso per la Macchina di Turing M' quando eseguita su una Macchina di Turing arbitraria che termina su almeno un input. Da ogni nodo etichettato con un numero naturale si dirama una catena di altri nodi ognuno dei quali è etichettato con una coppia

di naturali. Il nodo etichettato con la coppia (i, j) rappresenta la j-esima configurazione che la Macchina di Turing M' incontra quando eseguita sulla Macchina di Turing fissata e l'input i. Se una sequenza di nodi termina nell'unica configurazione finale (q_{accept}) di M' allora il grafo contiene un'ulteriore coppia di archi (tratteggiati in figura), a partire dal nodo che rappresenta l'input alla base di quella catena ed il nodo raffigurante tale configurazione finale (etichettato "End") e viceversa. Gli infiniti altri nodi connessi al nodo "End" e ai nodi etichettati con i numeri naturali (detti nodi **dummy**) sono un espediente che permette la rappresentazione finita di tale grafo ma non hanno alcun significato.

Tale grafo infinito è rappresentabile finitamente. Una funzione f che enumeri i suoi nodi esiste banalmente in virtù del fatto che i numeri naturali sono enumerabili (e pertanto i nodi etichettati con i numeri naturali), il nodo "Start" è un unico nodo, le configurazioni di una Macchina di Turing su di un dato input sono numerabili ed i nodi dummy sono tanti quanti i numeri naturali.

La funzione di ordine superiore g che dato un nodo restituisce un enumeratore dei suoi vicini richiede un artificio più astuto. Nel caso del nodo "Start" la funzione è triviale così come quando eseguita su di un nodo che rappresenta una configurazione intermedia della Macchina di Turing in esame su un dato input oppure su uno degli infiniti nodi dummy. Quando eseguita su un nodo etichettato con un numero naturale la funzione g restuisce una funzione g' che prende in input un numero naturale n e si comporta come segue.

Su input 0, g' restituisce il nodo etichettato con il naturale immediatamente precedente a quello con il quale è etichettato il nodo corrente. Su input 1 restituisce il nodo etichettato con il naturale immediatamente successivo. Su input n esegue la Macchina di Turing di riferimento per n passi e se questa termina (raggiungendo la configurazione q_{accept}) allora la funzione restituisce tale configurazione in output (se non enumerata in precedenza) altrimenti enumera il primo nodo dummy connesso a tale nodo non ancora enumerato.

I nodi dummy connessi al nodo in questione vengono inoltre enumerati da g' ad intervalli regolari.

La funzione g eseguita sul nodo rappresentante la configurazione finale "End" restituisce un'altra funzione g' che opera invece nel modo seguente. $g'(n)$ esegue la Macchina di Turing di riferimento in parallelo su tutti i possibili input naturali minori o uguali a n per n passi e, se almeno una di tali computazioni termina enumera in sequenza il nodo corrispondente alla penultima configurazione della Macchina di Turing sull'input relativo ed il nodo etichettato con il naturale corrispondente a quell'input. Se nessuna di tali computazioni termina (ed ad intervalli regolari) la funzione g' enumera il primo degli infiniti nodi dummy connessi al nodo "End" non ancora enumerato.

Le funzioni f e g garantiscono la rappresentazione finita del grafo in questione. Si osservi quindi che $Min((f, g), Start, End)$ restituisce il cammino più breve tra i nodi $Start$ ed End il quale penultimo nodo (necessariamente etichettato con un numero naturale) rappresenta il più piccolo input sul quale la Macchina di Turing in esame termina. Ma ciò è assurdo in quanto il precedente lemma dimostra che è impossibile sviluppare una Macchina di Turing del genere. Pertanto la Macchina di Turing Min non può esistere. □

2.4.4 Calcolo di un cammino minimo in un grafo infinito connesso con grado finito

Sotto l'ulteriore ipotesi che il grado massimo di un grafo infinito fortemente connesso sia un numero fissato k è possibile sviluppare un algoritmo che calcoli il cammino minimo tra due nodi u e v di tale grafo. Non dovrebbe essere sorprendente che tale algoritmo coincide con la classica procedura di visita in ampiezza di un grafo (BFS). Di fatti, poichè il numero dei vicini di ogni nodo è limitato da k è possibile allontanarsi progressivamente dal nodo u e poichè il grafo è fortemente connesso prima o poi si incontrerà il nodo v. Poichè la funzione BFS considera prima i nodi vicini del nodo di partenza,

poi i vicini di tali vicini, quindi i vicini dei vicini di tali vicini e così via è chiaro che il primo cammino individuato tra u e v sarà quello più breve.

3 Documenti

3.1 Documenti strutturati e semi-strutturati

Un **documento non strutturato** o **testuale** è una stringa di testo su di un dato alfabeto. Formalmente dato un alfabeto finito Σ, un documento non strutturato d è semplicemente un elemento di Σ^*. La "Divina Commedia" di Dante Alighieri, il celebre poema "Ed è subito sera" di Salvatore Quasimodo ed il testo del singolo "La guerra di Piero" di Fabrizio De Andrè sono tutti esempi di documenti non strutturati sull'alfabeto italiano.

Un **documento semi-strutturato** è invece una collezione di stringhe di testo su di un alfabeto organizzate in una data relazione gerarchica. Una lista di scrittori, di ognuno dei quali viene tenuta traccia del nome e della data di nascita, è un documento semi-strutturato. E' conveniente immaginare un documento semi-strutturato come un albero le cui foglie sono stringhe di testo. Nell'immagine in Figura 3 i cerchi rappresentano i nodi che contribuiscono alla struttura del documento (detti **tag**), i rettangoli (detti invece **nodi testuali**) rappresentano il testo in essi contenuto.

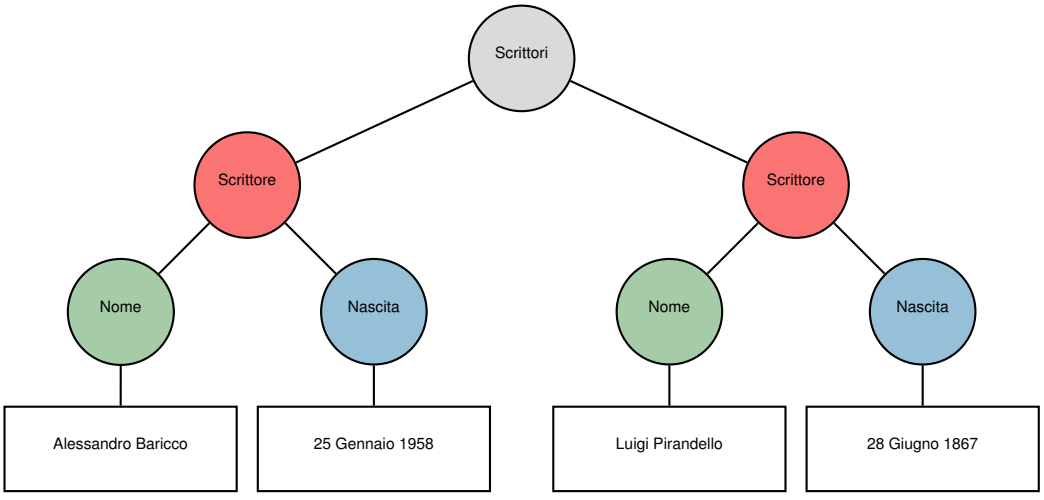

Figura 3: Documento semi-strutturato.

3.2 Definizione dei documenti semi-strutturati

L'insieme dei documenti semi-strutturati Γ è definito di seguito per induzione strutturale.

Definizione 3.1 (Insieme dei documenti semi-strutturati)**.**
Sia Σ un alfabeto finito e sia T un insieme finito (detto insieme dei tag). L'**insieme dei documenti semi-strutturati** Γ è il più piccolo insieme che contiene tutte le stringhe di testo sull'alfabeto Σ e, se $t \in T$ è un tag allora la coppia composta da t e dalla tupla contenente un qualsiasi numero di documenti semi-strutturati è a sua volta un documento semi-strutturato.

$$\Sigma^* \subseteq \Gamma \qquad \text{(Base)}$$

$$\text{Se } t \in T \text{ e } d_1, \ldots, d_n \in \Gamma \text{ allora } (t, (d_1, d_2, ..., d_n)) \in \Gamma \qquad \text{(Induzione)}$$

Si noti che se l'insieme dei tag T è vuoto allora l'insieme dei documenti semi-strutturati su di un dato alfabeto Σ coincide con l'insieme dei documenti non strutturati su tale alfabeto.

Per praticità notazionale le parentesi di una tupla contenente un solo elemento possono essere omesse. Pertanto il documento (t, A) è da considerarsi equivalente al documento $(t, (A))$. Nel contesto dei documenti semi-strutturati la stringa vuota ϵ viene detta **documento vuoto** mentre un documento della forma (t, ϵ) viene comunemente chiamato **tag vuoto**.

In accordo a quanto detto, il documento d rappresentato nella Figura 3 è matematicamente formalizzato mediante i seguenti alfabeti e la seguente tupla

$$\Sigma = \{\ a, \ldots, z, A, \ldots, Z\ \},$$
$$T = \{\ \text{Scrittori}, \text{Scrittore}, \text{Nome}, \text{Nascita}\ \},$$
$$s_1 = (\text{Scrittore}, ((\text{Nome}, \text{Alessandro Baricco}), (\text{Nascita}, 25\ \text{Gennaio}\ 1958))),$$
$$s_2 = (\text{Scrittore}, ((\text{Nome}, \text{Luigi Pirandello}), (\text{Nascita}, 28\ \text{Giugno}\ 1867))),$$
$$d = (\text{Scrittori}, (s_1, s_2)).$$

Se $d = (t, (d_1, d_2, ..., d_n)) \in \Gamma$ è un documento semi-strutturato, ogni d_i è detto **figlio** di d. L'insieme dei **sotto-documenti** di un documento semi-strutturato d è il più piccolo insieme definito dalle seguenti regole

1. d è un sotto-documento di d,
2. se s è un figlio di d allora tutti i sotto-documenti di s sono sotto-documenti di d.

A titolo esemplificativo la Figura 4 rappresenta un documento semi-strutturato (sopra la linea) ed i suoi sotto-documenti (immediatamente sotto la linea).

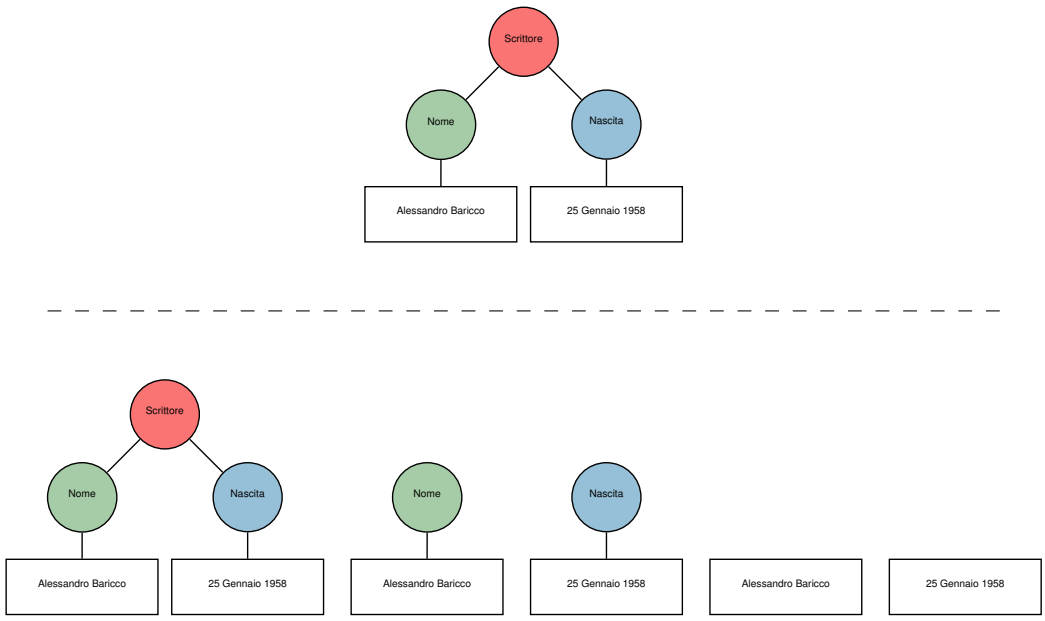

Figura 4: Documento semi-strutturato con i suoi sottodocumenti.

3.2.1 Documenti ben formati

Nonostante la formalizzazione dei documenti semi-strutturati fornita in precedenza sia verosimile, semplice ed elegante potrebbe ancora rivelarsi ambigua.

Si pensi ad esempio ai seguenti documenti

$$\Sigma = \{\ a, \ldots, z\ \},\ T = \{\ t\ \},$$
$$d_1 = (t, ab),\ d_2 = (t, (a, b)),$$
$$d_3 = (t, (t, \epsilon)),\ d_4 = (t, (\epsilon, (t, \epsilon))).$$

Intuitivamente d_1 e d_2 dovrebbero essere rappresentazioni differenti dello stesso documento. Di fatti il primo consta di un nodo t contenente la stringa ab mentre il secondo consta di un nodo t contenente la stringa a concatenata alla stringa b. Lo stesso dovrebbe valere per i documenti d_3 e d_4. Il primo problema nasce dal fatto che non esiste una rappresentazione univoca dei nodi testuali. Il secondo problema è invece dovuto al fatto che la stringa vuota concatenata a qualsiasi altro documento lo lascia invariato. Da questa intuizione sorge la necessità di definire un sottoinsieme dei documenti semi-strutturati che non lasci spazio ad ambiguità.

Definizione 3.2 (Insieme dei documenti ben formati)**.**
Sia Γ un insieme di documenti semi-strutturati. L'**insieme dei documenti ben formati** $\Gamma_{wf} \subseteq \Gamma$ è l'insieme di tutti i documenti $d \in \Gamma$ per cui valgono le proprietà elencate di seguito. Per ogni sotto-documento d_i di $d = (t, (d_1, \ldots, d_n))$

1. se $d_i \in \Sigma^*$ allora $d_{i+1} \notin \Sigma^*$,
2. se esiste un $d_i = \epsilon$ allora $n = 1$.

Ciò vuol dire che un documento semi-strutturato è **ben formato** quando nessun suo sotto-documento contiene due stringhe come figli consecutivi e, se un sotto-documento ha per figlio la stringa vuota, allora non ha altri figli. Tale definizione preferisce il formato in cui stringhe che sono figli consecutivi di un certo sotto-documento sono state accorpate in un unico nodo testuale. Di fatti se $\Sigma = \{\ a, b, c\ \}$ e $T = \{\ t\ \}$ allora $(t, abc) \in \Gamma$ e $(t, (a, b, c)) \in \Gamma$ ma

mentre il primo è ben formato

$$(t, abc) \in \Gamma_{wf},$$

lo stesso non vale per il secondo

$$(t, (a, b, c)) \notin \Gamma_{wf}.$$

E' opportuno considerare che svariati approcci equivalenti sarebbero potuti essere presi in considerazione. Uno di questi consiste nel permettere di inserire solo singoli caratteri piuttosto che intere stringhe nei nodi testuali; le stringhe di testo di un documento vengono quindi rappresentate nella forma di lunghe sequenze di caratteri consecutivi. Sebbene di indubbia semplicità quest'ultimo espediente tende a produrre documenti di dimensioni eccessive; vi è stato dunque preferito l'approccio discusso in precedenza.

Si osservi che per ogni alfabeto finito Σ e per ogni insieme di tag T, l'insieme dei documenti ben formati Γ_{wf} su Σ e T è numerabile.

Dimostrazione.
Ogni documento $d \in \Gamma_{wf}$ è essenzialmente una stringa di simboli sull'alfabeto $\Theta = \Sigma \cup T \cup S$ dove S è un insieme di simboli ausiliari utilizzati per rappresentare le tuple (come ad esempio le parentesi tonde e la virgola). Poichè Σ, T e S sono insiemi finiti allora Θ^* è numerabile. Sia $f : \Gamma_{wf} \to \Theta^*$ la funzione che mappa un documento ben formato nella stringa di caratteri che lo compone. Allora $\{\ f(d) \in \Theta^* \mid d \in \Gamma_{wf}\ \} \subseteq \Theta^*$ è banalmente numerabile. Pertanto Γ_{wf} è numerabile. □

3.2.2 Indicizzazione

Per poter individuare uno specifico sotto-documento di un documento semi-strutturato d è necessario che ogni sotto-documento di d sia univocamente identificato. Poichè, per costruzione, un documento semi-strutturato contiene un numero finito di sotto-documenti, è possibile assegnare un numero

naturale ad ognuno di questi. Una strada possibile consiste nel visitare in ampiezza il documento a partire dalla sua radice assegnando numeri progressivi ai nodi incontrati. Questa tecnica prende nome di **indicizzazione BFS** di un documento semi-strutturato.

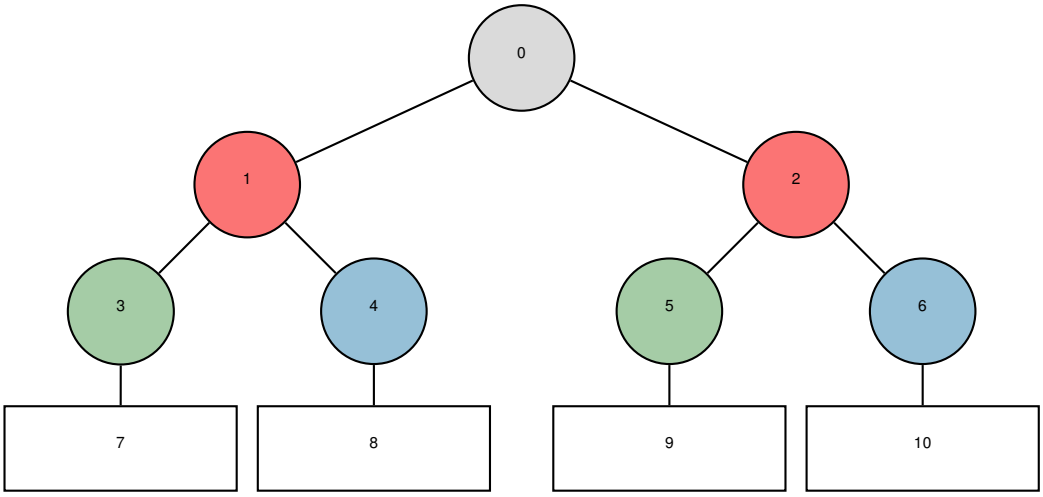

Figura 5: Indicizzazione BFS di un documento semi-strutturato.

Una alternativa a tale indicizzazione consiste nell'**indicizzazione relativa**. L'indicizzazione relativa prevede che ad ogni sotto-documento s di un documento d venga assegnata una sequenza di numeri naturali $(i_1, \ldots, i_k)$ in modo che, partendo dalla radice di d e selezionando il sotto-documento i_1-esimo, e quindi l'i_2-esimo e così via fino all'i_k-esimo si ottenga il sotto-documento s. La Figura 6 esprime l'indicizzazione relativa di un documento semi-strutturato di esempio.

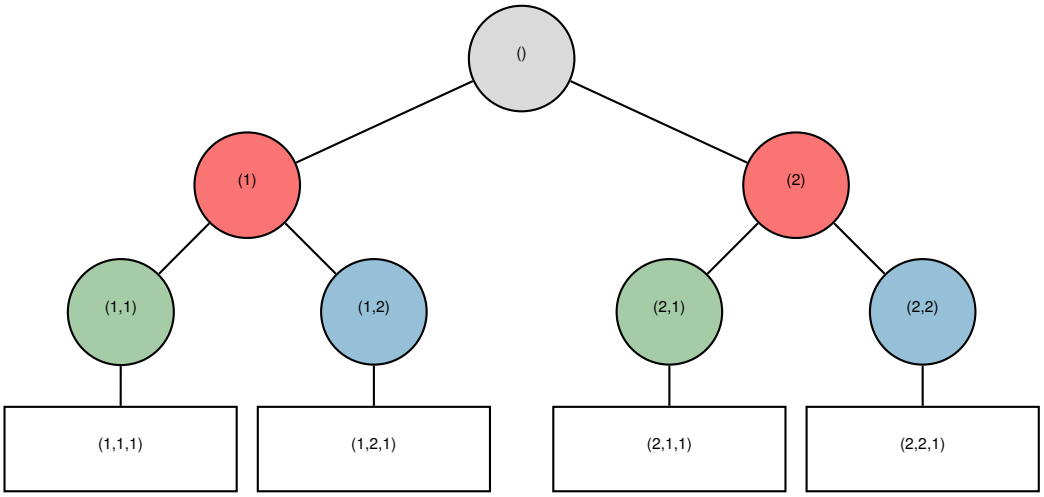

Figura 6: Indicizzazione relativa di un documento semi-strutturato.

4 Verso una teoria delle differenze

Correva l'anno 1961 e, sul palco della trasmissione "Studio uno", le gemelle Kessler danzavano al ritmo incalzante del Da-da-um-pa. Splendide e leggere erano una lo specchio dell'altra, identiche all'apparenza ma allo stesso tempo distinte dalle movenze, dal ritmo e dalla personalità. Indipendentemente da quanto simili fossero nell'aspetto le gemelle Kessler erano diverse per il solo fatto di essere due.

Un oggetto, un animale, una persona, finanche un concetto sono uguali solo a se stessi. Due entità distinte saranno sempre rese diverse da qualcosa. La stringa "l'uguaglianza è relativa" e la stringa "l'uguaglianza è relativa" contengono esattamente gli stessi caratteri e nello stesso ordine ma sono riportate in due posizioni differenti in questa pagina.

Tralasciando l'aspetto filosofico dell'uguaglianza e della diversità, nella pratica spesso si considerano oggetti diversi (nella forma o nel funzionamento) uguali gli uni agli altri. Se si è interessati ad analizzare solo determinate proprietà due oggetti possono veramente risultare indistinguibili e risultare invece differenti se se ne analizzano altre. Ad esempio le tre figure geometriche rappresentate in Figura 7 sono identiche in termini del numero di lati

ma diverse in termini del colore o dell'angolo di inclinazione.

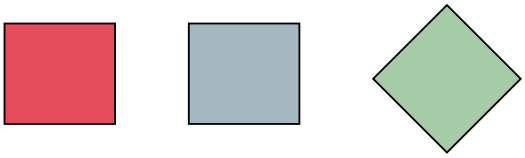

Figura 7: Uguaglianza e diversità di figure geometriche.

Sebbene Johann Wolfgang Goethe parlasse di bellezza, il suo aforisma "La bellezza è negli occhi di chi guarda" vale anche per l'uguaglianza, la diversità e la similarità. Ma che cos'è, in fondo, una **differenza** tra due oggetti? In aritmetica una differenza tra due numeri a e b è quel numero c che sommato a b, produce a. In termini più generali si potrebbe pensare alla differenza tra due oggetti (appartenenti rispettivamente ad un insieme A ed ad un insieme B) come ad una funzione $f : A \to B$ che, quando applicata ad uno dei due, produce l'altro. Tale idea è rappresentata in Figura 8.

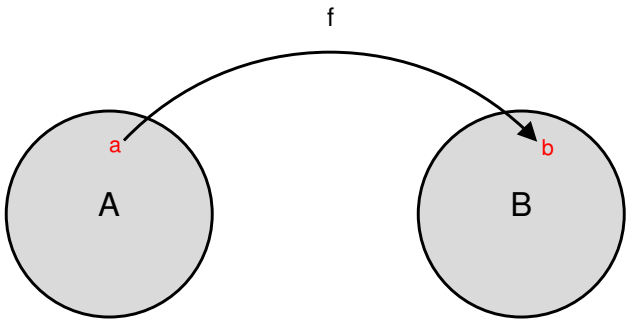

Figura 8: Differenza tra un elemento a ed un elemento b.

E' proprio questa l'intuizione da cui prende forma la **teoria delle differenze**. Tale formalismo matematico intende fornire uno strumento in grado di individuare le differenze tra oggetti nella forma di funzioni che permettono di trasformare gli uni negli altri. Partendo da una formalizzazione molto generale di tale teoria si giungerà infine all'applicazione di tecniche per l'analisi delle differenze ai documenti semi-strutturati definiti nelle precedenti sezioni. Si investigheranno le ipotesi necessarie che inducono la calcolabilità di un al-

goritmo di analisi delle differenze ed una serie di espedienti atti a migliorare sensibilmente le prestazioni di tale algoritmo.

4.1 Funzioni di trasformazione

Quando si parla di diversità tra oggetti è bene osservare che gli oggetti confrontati devono fare parte della stessa categoria. Ha poco senso mettere a confronto un sentimento (ad esempio l'amore) con un frutto (ad esempio una mela). E' invece molto interessante individuare le differenze tra due persone, tra due circuiti elettrici o tra due documenti.

Sulla base di quanto discusso in precedenza una differenza tra due oggetti a_1 e a_2 del medesimo tipo, ossia appartenenti ad uno stesso insieme A, è una funzione di ordine superiore che prende in input un certo oggetto s in un dato insieme S e restituisce una funzione (istanza parziale della funzione di partenza) che applicata all'elemento a_1 produce l'elemento a_2. Si pensi ad esempio alla differenza tra i numeri naturali 5 e 7. Questa potrebbe essere espressa in termini di una funzione $f : \mathbb{Z} \to \mathbb{N} \to \mathbb{N}$ definita nel modo seguente

$$f(k) = g_k(n) = n + k.$$

Dovrebbe essere chiaro che l'istanza parziale $f(2)$ rappresenta la differenza tra il numero 5 ed il numero 7, infatti $f(2)(5) = 7$.

In accordo a quanto discusso finora, una **funzione di trasformazione** o più semplicemente una **trasformazione**, così chiamata in virtù del suo principale utilizzo (la trasformazione di oggetti in altri oggetti), è formalmente definita di seguito.

Definizione 4.1 (Trasformazione)**.**
Dati due insiemi numerabili A ed S, una (funzione di) **trasformazione** su A è una funzione di ordine superiore calcolabile $f : S \to A \to A$.

4.2 Multigrafo delle trasformazioni

Essendo interessati alle differenze tra gli elementi di un determinato insieme A è opportuno definire una struttura matematica in grado di rappresentare tali elementi e le differenze (in termini di funzioni di trasformazione) tra ogni coppia di elementi in esame. La struttura più semplice che serve lo scopo è un multigrafo diretto etichettato.

Ogni nodo del multigrafo rappresenta un elemento dell'insieme A mentre un arco tra due nodi a_1 e a_2 è etichettato con l'istanza parziale $f(s)$ di una funzione di trasformazione f se la funzione $f(s)$ applicata all'elemento a_1 restituisce l'elemento a_2.

Si noti che tale struttura è effettivamente un multigrafo e non un grafo in quanto potrebbero esistere due funzioni di trasformazione diverse f e g che, opportunamente istanziate parzialmente, si comportano allo stesso modo quando applicate ad un determinato elemento $a \in A$. Per esempio si consideri l'insieme P dei pesi di un gruppo di persone

$$P = \{\ 30, 42, 50, 56, 58, 60, 74\ \}$$

e le due funzioni di trasformazione $ingrassa : \mathbb{N} \to P \to P$ e $raddoppia : \{\ u\ \} \to P \to P$. La prima funzione prende in input un numero naturale (rappresentate un certo numero di chilogrammi) e restituisce una funzione che preso un peso lo incrementa di tale ammontare (ingrassa la persona di un certo valore). La seconda invece prende in input un elemento u superfluo (il cui unico scopo è garantire l'esistenza della funzione), lo ignora, e restituisce una funzione che preso un peso ne raddoppia il valore (in pratica raddoppia il peso di un individuo). Si noti che se $p = 30$ allora

$$ingrassa(30)(p) = raddoppia(u)(p) = 60.$$

Ossia le istanze parziali $ingrassa(30)$ e $raddoppia(u)$ individuano entrambe la stessa differenza tra il peso di 30 chilogrammi ed il peso di 60 chilogrammi.

Purtroppo in letturatura non esiste una definizione univoca di multigrafo

etichettato. Ai fini della presente trattazione si consideri dunque la seguente definizione.

Definizione 4.2 (Multigrafo etichettato)**.**
Un **multigrafo etichettato** su un insieme di etichette L è un multigrafo $G = (V, E)$ dove V rappresenta l'insieme dei nodi del multigrafo mentre $E \subseteq V \times V \times L$ rappresenta l'insieme dei suoi archi. La tripla $(u, v, l) \in E$ rappresenta l'arco che connette il nodo u con il nodo v etichettato con l.

Esattamente come un grafo diretto differisce da un grafo non orientato per il fatto che i suoi archi sono delle frecce piuttosto che delle linee, un **multigrafo diretto etichettato** è un multigrafo etichettato in cui gli archi sono delle frecce invece che delle linee. Si noti che in un multigrafo etichettato l'arco (a, b, l) coincide con l'arco (b, a, l) mentre in un multigrafo diretto etichettato tali archi sono diversi.

Il multigrafo delle trasformazioni tra gli elementi di un dato insieme A è definito come segue.

Definizione 4.3 (Multigrafo delle trasformazioni)**.**
Dato un insieme numerabile A ed un insieme F di funzioni di trasformazione su A, il **multigrafo delle trasformazioni** su F è un multigrafo diretto etichettato $G = (V, E)$ tale che

1. $V = A$,
2. $L = A \to A$,
3. $(u, v, f(i)) \in E$ se e solamente se $f \in F$ e $f(i)(u) = v$.

L'immagine in Figura 9 è un esempio di multigrafo delle trasformazioni su un insieme infinito numerabile A. Tale multigrafo consta di infiniti nodi (gli infiniti elementi di A), alcuni dei quali dotati di grado infinito. Gli archi colorati con colori diversi rappresentano istanze parziali diverse di varie funzioni di trasformazione.

Poichè come dimostrato in precedenza l'insieme dei documenti ben formati Γ_{wf} su di un dato alfabeto ed un dato insieme dei tag è numerabile,

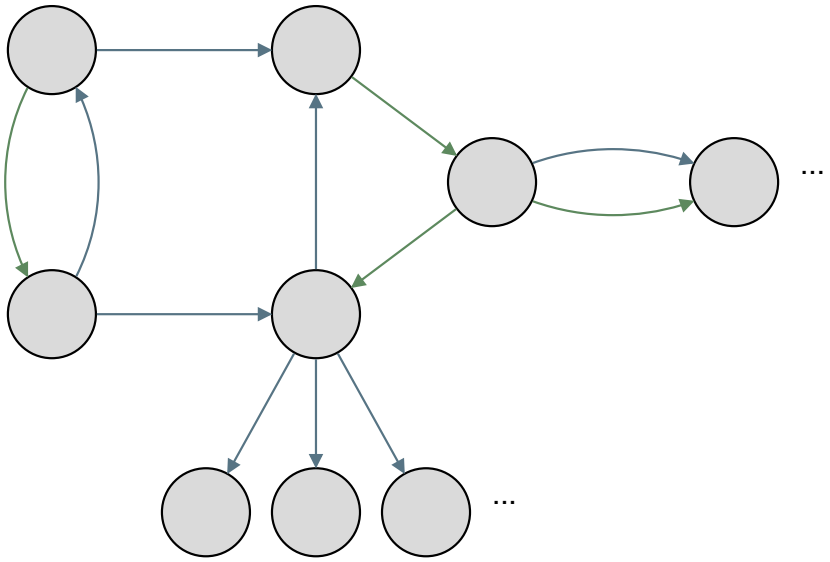

Figura 9: Multigrafo delle trasformazioni.

è possibile definire il multigrafo delle trasformazioni dei documenti semi-strutturati ben formati.

4.3 Diffing

Il multigrafo delle trasformazioni dei documenti semi-strutturati ben formati è uno strumento teorico di grande potere. La possibilità di individuare differenze tra documenti semi-strutturati permette di sviluppare **sistemi di versionamento**[2] ottimizzati per tale tipologia di documenti che siano efficaci ed efficienti (in termini dello spazio su disco richiesto per tener traccia delle varie versioni di un progetto).

Ad oggi esistono numerosi sistemi di versionamento (quali ad esempio "Git" o "Subversion") ampiamente impiegati in pratica, che permettono di gestire vaste collezioni di dati con il minimo sforzo. In pratica tali sistemi

[2]Un sistema di versionamento è un programma che permette di memorizzare dei dati (generalmente file facenti parte di un progetto di sviluppo software, ma anche documenti di varia natura quali libri o relazioni) tenendo traccia delle modifiche che vi vengono apportate e permettendo di ripristinare una versione dei dati di lavoro antecedente alla versione attuale qualora necessario.

memorizzano la versione originale dei dati che vengono loro sottomessi e, ogni volta che tali dati vengono aggiornati producendone una nuova versione, memorizzano le differenze tra la versione corrente e la precedente onde poter successivamente ricostruire ogni eventuale versione intermedia sotto richiesta. Non è affatto sorprendente scoprire che tali differenze vengono calcolate in automatico mediante celebri algoritmi di analisi delle differenze (anche detti algoritmi di **diffing**) tra i quali spicca per importanza *diff*, programma nativamente disponibile sui sistemi Unix.

Tuttavia le differenze tra due documenti che *diff* è in grado di individuare sono limitate alla mera aggiunta e cancellazione di linee di testo. Ne consegue che tale algoritmo, applicato a documenti inerentemente semi-strutturati, produce spesso un insieme di differenze indecifrabile. Un utente interessato a comprendere in che modo un documento semi-strutturato è stato realmente modificato non ha speranza di riuscire in tale intento analizzando l'output del programma in questione. Pertanto numerose alternative (rigorosamente orientate all'analisi delle differenze tra documenti semi-strutturati) sono state sviluppate negli ultimi anni. Tra le più famose ed acclamate è doveroso citare *diffX*, *DocTreeDiff*, *Meaningful Change Tool*, *Ndiff* e *XyDiff*. Le differenze che tali programmi sono in grado di individuare sono molteplici, ad esempio il movimento di un nodo da una data posizione di un documento semi-strutturato ad un'altra, l'incapsulamento di un nodo in un nuovo tag, il riordinamento di nodi e così via.

Purtroppo tutti gli algoritmi citati fissano un insieme di differenze considerate rilevanti (generalmente differente per ogni algoritmo) e sono in grado di individuare esclusivamente tali differenze. Al fine di garantire generalità ed applicabilità tali algoritmi non sono ancora sufficientemente generali da poter essere utilizzati per il confronto di documenti semi-strutturati di qualsivoglia natura.

Si osservino ad esempio in via intuitiva le differenze tra i due documenti semi-strutturati in Figura 10. Ambedue i documenti rappresentano la stessa funzione in un determinato linguaggio di programmazione i cui attributi sono

resi nel primo in italiano, nel secondo in inglese. Un algoritmo di diffing in grado di rilevare solamente la cancellazione e l'inserimento di un sotto-documento dichiararebbe che le operazioni che permettono di trasformare l'uno nell'altro consistono nel cancellare il primo documento per intero e rimpiazzarlo con il secondo. Tuttavia una operazione ben più rappresentativa del tipo di differenza espresso sarebbe stata la *traduzione in inglese di un sotto-documento*. Talune differenze hanno più o meno peso in dipendenza del loro contesto applicativo esattamente come uno spettacolo teatrale viene apprezzato di più da una determinata platea e meno da un'altra.

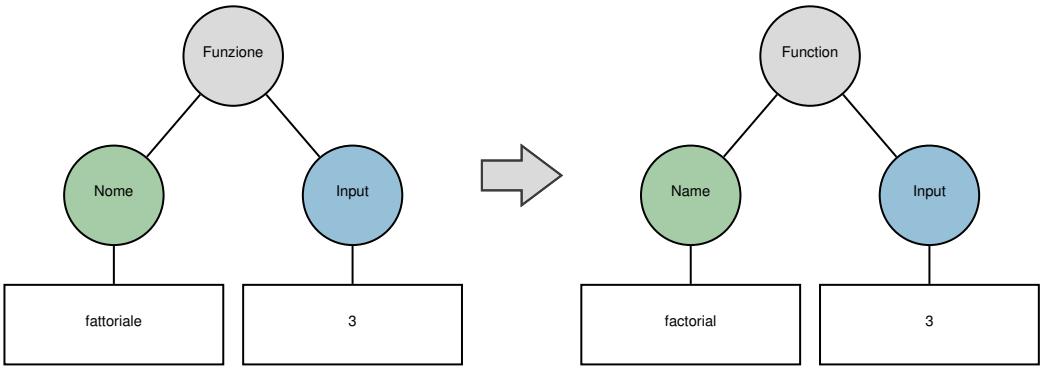

Figura 10: Traduzione di un sotto-documento in inglese.

4.3.1 Diffing parametrico

Si consideri l'insieme dei documenti semi-strutturati ben formati Γ_{wf} ed un insieme finito di trasformazioni $F = \{ f_1, \ldots, f_n \}$ su Γ_{wf}. Sia G il multigrafo delle trasformazioni su F. Se fosse possibile individuare un cammino tra due nodi (documenti semi-strutturati ben formati) in G allora la sequenza di etichette presenti sugli archi che connettono i nodi di tale cammino sarebbe a tutti gli effetti una sequenza di funzioni che, composte tra loro ed applicate al primo documento permetterebbero di ottenere il secondo. Si noti di fatti che se $((d_1, d_2, f_{i_1}(x_1)), (d_2, d_3, f_{i_2}(x_2)), \ldots, (d_{k-1}, d_k, f_{i_{k-1}}(x_{k-1})))$ è la sequenza

di archi etichettati lungo un cammino tra d_1 e d_k in G allora

$$f_{i_1}(x_1)(d_1) = d_2,$$
$$f_{i_2}(x_2)(d_2) = d_3,$$
$$\ldots,$$
$$f_{i_{k-1}}(x_{k-1})(d_{k-1}) = d_k.$$

Un algoritmo che permetta di individuare un cammino nel grafo G sarebbe quindi un algoritmo di diffing orientato ai documenti semi-strutturati. Tuttavia come dimostrato in precedenza un algoritmo che sia in grado di individuare un cammino tra due nodi di un grafo infinito non è calcolabile pertanto, a maggior ragione, non lo è un algoritmo che sia in grado di individuare un cammino tra due nodi di un multigrafo infinito diretto etichettato.

4.3.2 Diffing parametrico ottimale

Se anche fosse possibile sviluppare un algoritmo in grado di individuare un cammino tra due nodi nel multigrafo G rimarrebbe il problema che tale cammino potrebbe essere molto lungo. Un algoritmo di diffing che restituisca un insieme di differenze troppo vasto quando applicato a due versioni diverse di un dato documento semi-strutturato sarebbe comunque inservibile in pratica.

Ciò motiva il desiderio di sviluppare un algoritmo di diffing che sia in grado di trovare il più piccolo insieme di differenze tra due documenti semi-strutturati. Nella teoria di cui sopra la variante che permetterebbe di ottenere tale risultato è ovvia: invece di cercare un cammino tra due nodi è sufficiente cercare il cammino minimo.

Poichè è impossibile sviluppare un algoritmo che sia in grado di trovare un cammino tra due nodi nel multigrafo di riferimento è di certo impossibile anche svilupparne uno in grado di trovare il cammino minimo. Senza ulteriori assunzioni la teoria discussa perde ogni speranza di realizzabilità.

4.4 Consistenza

Definizione 4.4 (Insieme consistente)**.**
Un insieme F di trasformazioni su di un dato insieme A si dice **consistente** quando il risultante multigrafo delle trasformazioni su F è fortemente connesso.

Intuitivamente un insieme F di trasformazioni su A è consistente quando tali funzioni, istanziate parzialmente in vario modo, permettono di ottenere qualsiasi elemento $a_2 \in A$ a partire da qualsiasi altro elemento $a_1 \in A$.

E' del tutto ragionevole limitarsi a considerare multigrafi delle trasformazioni indotti da insiemi di trasformazioni consistenti. Di fatti dato un insieme F non consistente, l'algoritmo di diffing la cui ricerca è scopo della presente tesi non sarebbe in grado di funzionare in tutti i casi; le funzioni in oggetto sarebbero così deboli da non permettere molte (spesso anche infinite) ragionevoli trasformazioni.

4.4.1 Un insieme di trasformazioni consistente

L'assunzione di consistenza è interessante, tuttavia è lecito interrogarsi sulla sua fondatezza. Esiste almeno un insieme di funzioni che sia consistente? La risposta è certamente si come dimostrano i numerosi algoritmi di diffing citati in grado di individuare l'insieme delle differenze tra qualsiasi coppia di documenti semi-strutturati. Ai fini della presente trattazione si consideri l'insieme di trasformazioni consistente

$$\{\ addtag, erasetag, appendchar\ \}.$$

Le funzioni citate sono definite di seguito ed operano su un insieme di documenti ben formati Γ_{wf} definito su un alfabeto finito Σ ed un insieme finito dei tag T. Sia I l'insieme degli indici che permettono di individuare uno specifico sotto-documento di un documento semi-strutturato.

La funzione $addtag : (T \times I) \rightarrow \Gamma_{wf} \rightarrow \Gamma_{wf}$ prende in input un tag t, l'indice rappresentante un nodo di un documento semi-strutturato e restitu-

isce una nuova funzione che applicata ad un dato documento produce il nuovo documento ottenuto aggiungendo un nodo della forma (t, ϵ) come fratello immediatamente precedente di tale nodo (o sostituendo il nodo corrente se questi è un documento vuoto). Qualora ciò fosse impossibile, come nel caso in cui l'indice si riferisca alla radice del documento cui viene applicata, allora tale funzione si comporta banalmente come la funzione identità.

La funzione $erasetag : I \to \Gamma_{wf} \to \Gamma_{wf}$ prende in input un indice e restituisce la funzione che applicata ad un documento rimuove il sotto-documento relativo a tale indice sostituendolo con il documento vuoto qualora necessario.

La funzione $appendchar : (\Sigma \times I) \to \Gamma_{wf} \to \Gamma_{wf}$ prende in input un carattere c ed un indice e restituisce la funzione che applicata ad un dato documento modifica il contenuto del sotto-documento puntato da tale indice aggiungendo c in fondo ad ogni nodo testuale in esso contenuto.

Dati due documenti arbitrari d_1 e d_2 è sempre possibile trasformare l'uno nell'altro utilizzando la funzione *erasetag* per cancellare il primo documento per intero ottenendo il documento vuoto, applicando più volte la funzione *addtag* per mimare la struttura del documento d_2 ed applicando infine più volte la funzione *appendchar* per ricostruire il testo del documento d_2 in modo opportuno.

4.4.2 Osservazioni

E' importante osservare che dato un insieme F di trasformazioni su di un insieme A vale che per ogni altro insieme F' di trasformazioni su A, l'insieme $F \cup F'$ è ancora consistente. Di fatti poichè le sole funzioni in F sono sufficienti a garantire la forte connessione del multigrafo delle trasformazioni relativo questi rimarrà fortemente connesso anche in presenza degli ulteriori archi introdotti dalle funzioni in F'.

L'ipotesi di consistenza garantisce l'esistenza di un algoritmo che sia in grado di individuare un cammino tra qualsiasi coppia di nodi di un multigrafo delle trasformazioni. Sfortunatamente non è però sufficiente a garantire

l'esistenza di un algoritmo che sia in grado di trovare il cammino minimo tra tali nodi.

4.5 Multigrafo a grado finito

Sotto l'ulteriore ipotesi che un insieme di trasformazioni F induca un multigrafo delle trasformazioni su F con grado finito allora è possibile utilizzare una classica procedura di visita BFS per individuare il cammino (e quindi l'insieme delle differenze) minimo tra due documenti.

E' importante osservare che l'insieme di trasformazioni

$$\{\ addtag, erasetag, appendchar\ \}$$

discusso in precedenza induce un grafo non solo connesso ma con grado finito. Ciò potrebbe risultare anti-intuitivo in quanto l'insieme degli indici I sul quale tali funzioni sono definite è effettivamente infinito. In pratica, però, poichè un documento semi-strutturato è una struttura finita che contiene quindi un numero finito di nodi, il numero di indici veramente rilevanti per produrre nuovi documenti a partire da tali funzioni sarà finito a sua volta.

4.6 Complessità temporale nella dimensione dell'output

La teoria della complessità si occupa di definire un certo numero di metodologie per analizzare il tempo di calcolo (spesso nel caso più sfavorevole) di vari algoritmi. L'assunzione preliminare di tale formalismo è che l'input di tali algoritmi sia finito. La nozione di complessità (ossia il numero di passi di computazione necessari per risolvere un problema) è definita in termini di una funzione della dimensione di tale input.

Si pensi ad esempio ad un algoritmo per il calcolo dell'elemento minimo di un insieme $A \subset \mathbb{N}$ con $n \geq 1$ elementi. Tale algoritmo comincia supponendo che il primo elemento dell'insieme sia il minimo e, ad ogni passo di

computazione, considera un nuovo elemento confrontandolo con il minimo locale calcolato fino a quel punto; se scopre che tale elemento è più piccolo del minimo locale, sostituisce il minimo locale con il valore di tale elemento, altrimenti procede semplicemente considerando il valore successivo. Quando tutti gli elementi di A sono stati considerati allora il minimo locale coincide con l'elemento minimo dell'insieme.

Tale algoritmo richiede nel caso peggiore circa $2n$ passi di computazione in quanto ad ogni passo l'algoritmo è in grado di effettuare un confronto tra l'elemento correntemente considerato ed il minimo locale oppure è in grado di aggiornare il valore del minimo locale.

Questa nozione di complessità non è abbastanza generale da permettere l'analisi del tempo di calcolo dell'algoritmo di diffing parametrico sviluppato in quanto il suo input comprende anche un grafo infinito. Anche volendo considerare la dimensione della rappresentazione finita di tale grafo il tempo di calcolo non dipende affatto da quest'ultima. Al variare dell'insieme delle trasformazioni F il multigrafo delle trasformazioni da esso indotto può contenere cammini più o meno lunghi tra determinati nodi.

Tuttavia, sebbene operante su una struttura intrinsecamente infinita, l'algoritmo sviluppato produce sempre un output (ossia un cammino tra due nodi) di dimensione finita. E' di conseguenza possibile stimare il suo tempo di calcolo in funzione della lunghezza del cammino prodotto. Sebbene insufficiente a fornire una stima adeguata questa tecnica garantisce almeno un'intuizione di quanto tempo tale algoritmo impiega per produrre un cammino di lunghezza d.

Ad esempio la banale procedura BFS che permette di calcolare un cammino minimo di dimensione d tra due nodi d_1 e d_2 in un dato multigrafo delle trasformazioni richiede la visita (nel caso pessimo) di tutti i vicini di d_1, e quindi di tutti i vicini di tali vicini, e quindi di tutti i vicini dei vicini di tali vicini e così via per d volte. Sotto l'assunzione che il grado massimo del grafo sia limitato da una costante b l'algoritmo di visita BFS richiede ben b^d passi. Poichè tale numero all'aumentare di d cresce molto velocemente la

procedura BFS, per quanto semplice, è in pratica inservibile. Per rendersene conto è sufficiente pensare che, per calcolare un insieme di sole 20 differenze tra due documenti su di un multigrafo delle trasformazioni che abbia anche soltanto grado 8 richiede 8^{20} passi di computazione. Anche assumendo che il computer che esegue tale algoritmo sia in grado di effettuare un milione di passi al secondo sarebbero necessari circa 36558 anni di calcolo.

4.6.1 Algoritmi euristici di ricerca su grafi

Un celebre algoritmo per la ricerca di un cammino su grafi (anche infiniti) è A^*. Tale tecnica, sviluppata nell'ambito dell'intelligenza artificiale, è dimostrata essere tra le più efficienti in termini di tempo di calcolo per tale scopo.

A^* prende in input un grafo finito (o alternativamente la rappresentazione finita di un grafo infinito) $G = (V, E)$, due nodi u e v facenti parte del grafo ed una funzione di euristica $h : V \times V \to \mathbb{N}$ che stima la distanza tra due nodi e restituisce un cammino tra u e v. Se la funzione h è **ammissibile** (ossia restituisce sempre una distanza stimata minore della distanza effettiva tra i due nodi cui viene applicata) allora l'algoritmo è ottimale (individua il cammino più breve ed in genere è in grado di farlo molto velocemente).

4.6.2 Euristiche ammissibili e non ammissibili

Ricordando che i documenti semi-strutturati sono sostanzialmente equiparabili a stringhe di testo (si pensi alla loro rappresentazione induttiva) una funzione di euristica che individui la distanza tra due documenti potrebbe essere la classica funzione **edit distance** altresì nota come **distanza di Levenshtein**. Tale funzione è definita come il numero minimo di modifiche che consentono di trasformare una stringa in un'altra cancellando un carattere, sostituendo un carattere con un altro o inserendo un nuovo carattere. Per quanto semplice la distanza di Levenshtein non è però una euristica ammissibile per il multigrafo delle trasformazioni indotto dall'insieme di trasformazioni consistente $\{ addtag, erasetag, appendchar \}$.

Se con T_d si indica l'insieme dei tag contenuti in un documento d allora una funzione di euristica h ammissibile per tale multigrafo è la seguente

$$h(d_1, d_2) = |T_{d_1}| + |T_{d_2}| - 2 * |T_{d_1} \cap T_{d_2}|.$$

5 Un algoritmo di diffing parametrico

Al fine di sviluppare un algoritmo che permetta di individuare un insieme parametrico di differenze tra due documenti semi-strutturati è in primo luogo necessario fissare un linguaggio di programmazione adeguato ed una serie di strutture dati atte alla rappresentazione degli oggetti matematici discussi fino ad ora.

Non tutti i linguaggi di programmazioni sono indicati per sviluppare un algoritmo del genere. Si pensi ad esempio che la teoria di cui sopra fa largo uso di insiemi e di funzioni di ordine superiore, sarebbe quindi meglio utilizzare un linguaggio che permetta di rappresentare con semplicità entrambi tali categorie di oggetti.

I linguaggi funzionali ed i linguaggi ibridi servono lo scopo meglio dei linguaggi imperativi in quanto implementano nativamente strumenti per la definizione di funzioni di ordine superiore. E' stato quindi sviluppato un prototipo di tale algoritmo (chiamato **Paradiff**) scritto in *Python* che, sebbene ancora troppo semplice per essere effettivamente applicabile in pratica, dimostra quanto meno la realizzabilità del progetto stesso.

5.1 Tecnologie

Il prototipo **Paradiff** consiste in una libreria di funzioni che permettono di individuare il più piccolo insieme di differenze tra due documenti semi-strutturati XML. In dettaglio, tale libreria è stata sviluppata in Python facendo uso di una libreria preesistente chiamata *lxml* che permette con semplicità di rappresentare e di editare questa tipologia di documenti.

5.2 Implementazione

La funzione chiave di Paradiff è chiamata *delta*. Tale funzione prende in input due documenti XML d_1 e d_2 espressi nel formato *etree* reso disponibile da lxml, una funzione di euristica (preferibilmente ammissibile) che dati due documenti restituisce la distanza tra i due, un insieme di trasformazioni

consistente su documenti XML e produce l'insieme delle differenze tra d_1 e d_2. La funzione delta non è altro che una implementazione dell'algoritmo di ricerca A^* sul multigrafo infinito delle trasformazioni che viene espanso passo dopo passo a partire dal documento d_1 fino a raggiungere il documento d_2. Se l'euristica fornita è ammissibile sull'insieme di trasformazioni specificato allora delta termina e produce il più piccolo insieme di differenze tra i due documenti. Se l'euristica è invece non ammissibile la terminazione di tale funzione non è garantita così come non lo è nel caso in cui l'insieme di trasformazioni non è consistente.

5.2.1 Funzionali di trasformazione

I funzionali di trasformazione sono espressi nella forma di semplici funzioni Python che prendono in input un documento, l'indice rappresentante un sotto-documento del documento cui vengono applicate e restituiscono un nuovo documento. Il dominio (necessariamente finito) di ogni singola funzione di trasformazione può dipendere dal documento cui tale funzione viene applicata ed è pertanto rappresentato mediante una funzione che prende in input un documento e produce la lista degli elementi facenti parte del dominio in oggetto.

5.2.2 Multigrafo delle trasformazioni

Come discusso in precedenza il multigrafo delle trasformazioni è banalmente rappresentato mediante una coda che, in principio, contiene soltanto il documento XML origine della ricerca d_1. Passo dopo passo viene selezionato il documento d presente in tale coda per il quale è minima la somma del numero di trasformazioni correntemente applicate al documento di partenza per ottenerlo e della funzione di euristica tra tale documento ed il documento obiettivo d_2. Vengono quindi incodati i nodi vicini del documento selezionato (ossia tutti i documenti ottenuti istanziando parzialmente in ogni modo possibile le trasformazioni nell'insieme fornito in input su ogni possibile sotto-documento del documento d).

Poichè la coda tende a crescere di dimensione molto rapidamente l'algoritmo Paradiff potrebbe essere soggetto ad overflow. Sfortunatamente non sembra esistere alcuna soluzione banale a questo tipo di problema.

6 Conclusioni

In definitiva il presente testo propone una teoria matematica di analisi delle differenze tra oggetti facenti parte di un determinato insieme. Tale teoria, benchè ancora puramente embrionale, fornisce un certo numero di strumenti che garantiscono la realizzabilità di un algoritmo di diffing che sia generale nel tipo di differenze che è in grado di individuare.

Le prime sezioni investigano la calcolabilità di algoritmi di largo interesse su strutture dati infinite (quali gli insiemi numerabili ed i grafi infiniti). In particolare, tali sezioni raccolgono prove formali dell'indecidibilità di celebri problemi la cui soluzione è nota in contesto finito. Il più interessante risultato presentato consiste nella prova dell'impossibilità di calcolare mediante un computer un cammino minimo tra due nodi noti appartenere ad un grafo diretto infinito fortemente connesso.

Sono stati quindi discussi gli strumenti matematici atti alla formalizzazione dei documenti semi-strutturati, esempio cardine della tipologia di documenti più generale attualmente sfruttata in numerosissimi contesti quali l'interscambio dati sulla rete internet a mezzo del protocollo Hypertext Transfer Protocol (HTTP). E' stato definito elegantemente (e senza ambuiguità) il concetto di documento semi-strutturato ben formato. E' stata fornita una definizione formale del concetto di sotto-documento di un documento semi-strutturato e sono state inoltre presentate due tecniche per l'attribuzione di indici univoci ai sotto-documenti di un dato documento. La seconda di queste, nota come indicizzazione relativa, semplifica notevolmente la definizione matematica delle funzioni di trasformazione oggetto delle successive sezioni.

La teoria delle differenze è stata introdotta mediante la definizione di funzioni di ordine superiore note come "trasformazioni". Sulla base di queste è stato definito un multigrafo diretto etichettato detto "multigrafo delle trasformazioni" che rappresenta il cuore della presente tesi.

Facendo uso dei risultati espressi nelle primissime pagine di questo testo sono state sviluppate due sole ragionevoli ipotesi che inducono la calcolabilità

di un algoritmo di analisi delle differenze tra documenti semi-strutturati ben formati.

La complessità temporale di tale algoritmo è stata analizzata in termini della dimensione del suo output in quanto le digressioni che chiudono l'elaborato dimostrano l'effettiva inutilità di approcciare tale problema in termini della dimensione del suo input. Poichè il tempo di calcolo dell'algoritmo è incredibilmente alto sono stati proposti algoritmi tipici del campo dell'intelligenza artificiale per migliorarne sensibilmente le prestazioni. In particolare l'algoritmo A^* garantisce l'applicabilità dell'algoritmo in questione.

Tuttavia un vasto numero di domande sono tuttora senza risposta. Quali altre ragionevoli ipotesi inducono un inferiore tempo di calcolo per l'algoritmo di diffing sviluppato? Considerare uno schema che aggiunga ulteriori vincoli alla struttura di un documento semi-strutturato induce di per sè alcune delle proprietà discusse (quali ad esempio la consistenza dell'insieme di trasformazioni lecite oppure la finitezza del grado del multigrafo delle trasformazioni)? E' possibile generalizzare la definizione di documenti semi-strutturati in modo che permetta altresì la definizione di documenti infiniti (ad esempio accessibili mediante uno stream di dati)? Di quali ulteriori proprietà gode il multigrafo delle trasformazioni? Sotto quali assunzioni è possibile sviluppare un'algebra delle differenze su documenti semi-strutturati?

E' certo che ognuna di queste domande è sufficientemente complessa da richiedere uno sforzo risolutivo non banale. Rispondere ad una di queste potrebbe essere lo scopo di un futuro progetto di ricerca oppure di una tesi di laurea.

Il prodotto della presente è incompleto ma offre grandiose opportunità. Nuove riflessioni sull'argomento potrebbero, nel migliore dei casi, produrre un nuovo punto di vista, apprezzato e rispettato, riguardo l'analisi automatica delle differenze tra documenti o oggetti di qualsiasi tipo e magari anche tra le persone.

7 Ringraziamenti

Nella stesura della presente tesi ho affrontato molteplici problemi; problemi cui non sarei riuscito a far fronte senza il valido aiuto di amici e colleghi cui va tutta la mia gratitudine ed ammirazione. In primissimo luogo mi preme ringraziare **Michael Sipser**, professore presso il Massachusetts Institute of Technology, autore del libro "Introduction to the Theory of Computation" da cui è stata tratta la maggior parte del materiale allegato in Appendice (a volte anche nella forma di testuale citazione). Tale libro mi ha permesso di affrontare i problemi teorici relativi alla Teoria della Calcolabilità con rinnovato vigore ed interesse, ha motivato e sostenuto la mia ricerca e mi ha permesso di apprendere ben oltre quanto avrei mai sperato di poter conoscere.

Un'altra persona che merita indubbiamente i miei ringraziamenti è **Justin Poliey**, autore di una implementazione Python estremamente elegante dell'algoritmo A*, il cui lavoro è stato ampiamente impiegato nell'implementazione del programma basato sulla teoria ivi discussa.

Tutto quanto è oggetto di questo studio è stato profondamente ispirato dalla mia passione per la Logica Matematica. Passione che mi è stata trasmessa nella sua forma più pura dal professore **Michal Walicki**, attualmente impiegato presso l'università di Bergen (Norvegia) dove ho trascorso gli ultimi cinque mesi in quanto studente ospite. Il professor Walicki ha supervisionato il mio lavoro con costanza e perspicacia, mi ha motivato ed ha completato le mie intuizioni con osservazioni argute. Alcuni dei dettagli teorici più sottili della presente tesi sono stati suggeriti o altrimenti guidati dalla sua persona. La mia stima per lui è immensa.

Diverse personalità dell'università di Bologna hanno avuto un impatto non trascurabile sul mio lavoro. Tra queste spiccano per genialità **Danilo Montesi** e **Simone Martini**. Entrambi hanno guidato la mia ricerca suggerendo articoli relativi al progetto e criticando l'elaborato con incredibile professionalità. In particolare i suggerimenti del professor Martini mi hanno spinto a investigare sulla Teoria della Rappresentabilità Finita e mi hanno permesso di scoprire dettagli che sono ora fulcro della Teoria delle Differenze.

I dottorandi **Angelo Di Iorio** e **Gioele Barabucci**, nonchè miei correlatori, hanno supervisionato il mio lavoro con costanza. Mi hanno spronato con continui interrogativi e hanno promosso i miei risultati nel contesto universitario ed extra-universitario. Hanno inoltre premuto affinchè la teoria in esame divenisse realtà pratica motivando la realizzazione dell'algoritmo parametrico di ricerca delle differenze che conclude il mio lavoro di ricerca. Nonostante i loro numerosissimi impegni si sono dedicati con pazienza alla presente, correggendone errori e guidando le mie proposte e le mie intuizioni. Sono ad oggi tra le persone più preparate che io conosca, mi auguro che i nuovi tesisti della mia università possano condividere la mia stessa fortuna.

Che ne sia conscio o meno, **Raffaele Giannella** è stato per me, fin dal primo momento in cui l'ho conosciuto, fonte di analisi critica e di ispirazione. Le sue argute osservazioni, la sua passione per l'apprendimento e la sua stima di me mi hanno continuamente motivato negli ultimi due anni. La genialità dimostrata nell'affrontare la sua esperienza universitaria coadiuvata alla pazienza nel tener testa alle mie continue e pedanti intromissioni lo rende una persona di eccezionale qualità. Che io l'abbia da sempre considerato il migliore degli amici ed il migliore degli allievi non è mistero. Non solo è stato in grado di effettuare brillanti correzioni al mio lavoro ma mi ha supportato in ogni istante della mia ultima esperienza universitaria. Il mio augurio è che possa, ed è indubbio che ne abbia potenzialità, fare anche meglio di me, ad oggi e nella vita.

E' doveroso citare gli amici ed i colleghi conosciuti a Bergen, Abel Bordonado Lillo, Alessandra Tedeschi, Ana Belén Dono Matellano, Camilla Piccardo, Daniele Grasso, Daniel Nydal, Diego Martín López, Elvira García de Jalón, Guillem Rovira García-Teresa, Irina Hinterlechner, Julia Attout, Lisa Betz, Marika Ivanová, Marissa Murphy, Mariya Kuzmenko, Marta Aguilera Anguita, Matteo Remo Luzzi, Rafael Nozal Cañadas, Roberto Laface, So Komiyama, Tommaso Gambato. Persone splendide senza le quali il mio soggiorno in terra straniera sarebbe stato decisamente spento. Per tutto il tempo trascorso insieme scherzando, giocando o lavorando, a loro va il mio ringrazi-

amento più profondo. Tengo a ringraziare moltissimo anche Oscar Elfving ed Alejandra Huerta Moragón, cari amici che mi hanno ospitato presso il loro appartamento in Copenhagen con pazienza ed affetto. Infine i miei colleghi più stretti nonchè amici fidati, Matteo Brucato, Michele Consiglio e Miro Mannino che hanno contribuito al mio successo in incredibile misura.

Per concludere tengo a ringraziare la mia famiglia, mio padre **Carmine Aquino**, mia madre **Laura Cerruti** e mio fratello **Carlo Aquino**, che mi hanno sempre trattato con stima, rispetto ed amore. Grazie per avermi permesso di affrontare questa esperienza universitaria, per avermi spronato continuamente e per avere creduto in me fin dal primo istante. Ogni mio risultato, che abbia già conseguito o che conseguirò, è per tutti voi.

8 Bibliografia

Riferimenti bibliografici

[1] Raihan Al-Ekram, Archana Adma, Olga Baysal, *diffX: An algorithm to Detect Changes in Multi-Version XML Documents*, (Proceeding) CASCON '05 Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research, Pages 1-11, IBM Press, 2005.

[2] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, Marc Tommasi, *Tree Automata Techniques and Applications*, (Book), 2007.

[3] Tancred Lindholm , *A three-way merge for XML documents*, (Proceeding) DocEng '04 Proceedings of the 2004 ACM symposium on Document engineering, Pages 1-10, ACM New York, NY, USA, 2004.

[4] Tancred Lindholm, Jaakko Kangasharju, Sasu Tarkoma, *Fast and simple XML tree differencing by sequence alignment* (Proceeding) DocEng '06 Proceedings of the 2006 ACM symposium on Document engineering, Pages 75-84, ACM New York, NY, USA, 2006.

[5] Ermir Qeli, Julinda Gllavata, Bernd Freisleben, *Customizable detection of changes for XML documents using XPath expressions*, (Proceeding) DocEng '06 Proceedings of the 2006 ACM symposium on Document engineering, Pages 88-90, ACM New York, NY, USA, 2006.

[6] Sebastian Rönnau , Geraint Philipp, Uwe M. Borghoff, *Efficient change control of XML documents*, (Proceeding) DocEng '09 Proceedings of the 9th ACM symposium on Document engineering, Pages 3-12, ACM New York, NY, USA, 2009.

[7] Michele Schirinzi, *Ndiff, un approccio naturale al confronto di documenti XML*, (Thesis) Tesi di Laurea Specialistica in Informatica, Università di Bologna, 2007.

[8] Michael Sipser, *Introduction to the Theory of Computation, Second Edition*, (Book) Thomson Course Technology, a division of Thomson Learning, 2006.

[9] Andrea Tagarelli, Sergio Greco, *Semantic clustering of XML documents*, (Journal) ACM Transactions on Information Systems (TOIS) Volume 28 Issue 1, Article No. 3, ACM New York, NY, USA, January 2010.

[10] Jean-Yves Vion-Dury, *Diffing, patching and merging XML documents: toward a generic calculus of editing deltas*, ACM Symposium on Document Engineering '10, Pages 191-194, 2010.

[11] Michal Walicki, *Introduction to Mathematical Logic*, (Book) University of Bergen, 2006.

[12] Yu Yijun, Tun Thein Than, Nuseibeh Bashar, *Specifying and Detecting Meaningful Changes in Programs*, (Conference) 26th IEEE/ACM International Conference on Automated Software Engineering, 2011.

A Appendice

A.1 Nozioni matematiche usate in questo testo

Al fine di garantire la precisione e l'uniformità dell'elaborato, la presente sezione investiga i principali concetti alla base della teoria della computazione e della teoria delle differenze. Le definizioni ed i risultati ivi presentati, largamente impiegati in ogni altra sezione, intendono colmare eventuali lacune del lettore o dimostrarsi altrimenti valente opportunità di ripetizione.

A.1.1 Insiemi

Un **insieme** è una collezione di oggetti. Un insieme può contenere oggetti di ogni tipo: numeri, simboli, stringhe di testo e persino altri insiemi. Gli oggetti contenuti in un insieme sono chiamati i suoi **elementi**. Esistono svariati metodi per rappresentare un insieme, il più semplice consiste nell'elencare i suoi elementi tra parentesi graffe. Pertanto l'insieme

$$\{\ 7, 21, 57\ \}$$

contiene gli elementi 7, 21 e 57. I simboli $\in$ e $\notin$ denotano rispettivamente l'appartenenza o la non appartenenza ad un insieme. Ad esempio $7 \in \{\ 7, 21, 57\ \}$ e $8 \notin \{\ 7, 21, 57\ \}$. Per ogni coppia di insiemi A e B, si dice che A è un **sottoinsieme** di B (la cui notazione relativa è $A \subseteq B$) se ogni elemento di A appartiene a B. Si dice invece che A è un **sottoinsieme proprio** di B (la cui notazione associata è invece $A \subsetneq B$) se A è un sottoinsieme di B ed A non è uguale a B.

Nel descrivere un insieme non è considerato rilevante nè l'ordine degli elementi nè la loro eventuale ripetizione. A tutti gli effetti gli insiemi $\{\ 7, 21, 57\ \}$ e $\{\ 21, 57, 57, 7, 7, 7\ \}$ sono uguali. Se si è interessati a tener conto del numero di occorrenze di ogni elemento di un insieme quest'ultimo prende nome di **multiinsieme**. Per esempio $\{\ 7\ \}$ e $\{\ 7, 7\ \}$ sono multiinsiemi differenti ma sono lo stesso insieme.

Un insieme si dice **infinito** quando contiene un numero infinito di elementi. Poichè è impossibile listare tutti gli elementi di un insieme infinito spesso si utilizza la notazione "..." per intendere "la sequenza procede all'infinito in modo intuitivo". Ad esempio l'insieme dei **numeri naturali** $\mathbb{N}$ è rappresentato come

$$\{ 0, 1, 2, 3, \dots \}.$$

Mentre quello dei **numeri interi** $\mathbb{Z}$ come

$$\{ \dots, -2, -1, 0, 1, 2, \dots \}.$$

L'insieme contenente 0 elementi è chiamato **insieme vuoto** ed è indicato con il simbolo $\emptyset$.

Un modo particolarmente conveniente per definire un insieme è quello di dichiararlo come l'insieme di tutti gli elementi (di un'altro insieme) per cui vale una determinata regola. La notazione relativa è

$$\{ n \in A \mid \text{regola su } n \}.$$

Ad esempio l'insieme dei numeri naturali pari $\mathbb{P}$ è definito come

$$\{ n \in \mathbb{N} \mid n \text{ è pari} \}.$$

Dati due insiemi A e B si definisce **unione** di tali insiemi ($A \cup B$) l'insieme che contiene tutti gli elementi di A e tutti gli elementi di B. Si definisce **intersezione** di tali insiemi ($A \cap B$) l'insieme che contiene tutti gli elementi che appartengono sia ad A che a B.

A.1.2 Sequenze e tuple

Una sequenza è una lista di oggetti in un certo ordine. Spesso le sequenze vengono rappresentate listando il loro contenuto tra parentesi tonde. Ad

esempio la sequenza 7, 21, 57 viene rappresentata con la notazione

$$(7, 21, 57).$$

Mentre in un insieme non è importante nè l'ordine degli elementi nè quante volte questi sono ripetuti, in una sequenza entrambe queste caratteristiche sono prese in considerazione. Pertanto la sequenza $(7, 21, 57)$ è diversa sia dalla sequenza $(7, 7, 21, 57)$ che dalla sequenza $(7, 57, 21)$. Esattamente come gli insiemi, le sequenze possono essere finite o infinite. Una sequenza finita viene chiamata **tupla**. Una sequenza con k elementi viene invece chiamata **k-upla**. Una tupla con 2 elementi viene chiamata **coppia**, una tupla con 3 elementi viene invece chiamata **tripla**.

Dati due insiemi A e B si definisce **prodotto cartesiano** (la cui notazione è $A \times B$) l'insieme di tutte le coppie tali per cui il primo elemento della coppia è un elemento di A ed il secondo è invece un elemento di B. Ad esempio

$$\text{se } A = \{\ 1, 2\ \} \text{ e } B = \{\ x, y, z\ \},$$
$$A \times B = \{\ (1, x),\ (1, y),\ (1, z),\ (2, x),\ (2, y),\ (2, z)\ \}.$$

Il prodotto cartesiano di k insiemi $A_1 \times A_2 \cdots \times A_k$ è l'insieme di tutte le k-uple $(a_1, a_2, \ldots, a_k)$ tali per cui $a_i \in A_i$.

A.1.3 Funzioni e relazioni

Una **funzione** è un oggetto che prende in input un valore e produce un valore in output. In una funzione lo stesso input produce sempre lo stesso output. Per dichiarare che f è una funzione che su input a produce l'output b si scrive

$$f(a) = b.$$

Una funzione è anche detta **mappa**. Se $f(a) = b$ si dice che la funzione f mappa l'elemento a nell'elemento b. Un esempio di funzione particolarmente semplice è la funzione valore assoluto abs. Tale funzione prende in input un

numero x e se questi è positivo restituisce in output x, altrimenti restituisce $-x$. Ad esempio $abs(2) = abs(-2) = 2$. La funzione di addizione add prende invece in input una coppia di numeri e ne restituisce la somma.

L'insieme dei possibili input per una funzione è detto **dominio** mentre l'insieme dei possibili output è detto **codominio**. La notazione per dire che f è una funzione con dominio D e codominio R è

$$f : D \to R.$$

Nel caso della funzione valore assoluto, se si intende lavorare con i numeri interi, il dominio ed il codominio sono entrambi l'insieme $\mathbb{Z}$. Di conseguenza è possibile scrivere $abs : \mathbb{Z} \to \mathbb{Z}$. Allo stesso modo la funzione di addizione su interi ha per dominio le possibili coppie di interi $\mathbb{Z} \times \mathbb{Z}$ e per codominio $\mathbb{Z}$. Di conseguenza è possibile scrivere $add : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$.

Si noti che una funzione non deve necessariamente essere in grado di produrre tutti gli elementi del suo codominio. Ad esempio la funzione valore assoluto non è mai in grado di produrre -1 in output anche se $-1 \in \mathbb{Z}$.

Una funzione che è in grado di produrre in output ogni elemento del proprio codominio è detta **suriettiva**. Se una funzione mappa elementi nel dominio diversi tra loro in elementi nel codominio diversi tra loro allora si dice che la funzione è **iniettiva**. Una funzione che è sia suriettiva che iniettiva si dice **biettiva**.

Esistono svariati modi per descrivere una funzione, uno di questi consiste nel rappresentarla mediante un insieme di coppie il cui primo elemento rappresenta un input della funzione ed il secondo l'output associato a quell'input. Poichè una funzione in corrispondenza di un determinato input produce sempre un solo output, affinchè un insieme di coppie sia una funzione valida è necessario che non contenga due coppie con il primo elemento in comune ma il secondo differente. Per esempio la funzione $abs : \mathbb{N} \to \mathbb{N}$ può essere

rappresentata mediante l'insieme

$$\{\ (x, y) \in \mathbb{N} \times \mathbb{N} \mid y = x \text{ se } x \geq 0; y = -x \text{ altrimenti }\ \}$$

Quando il dominio di una funzione è $A_1, \times \cdots \times A_k$ per qualche insieme $A_1, \ldots, A_k$ allora la funzione prende in input una k-upla $(a_1, \ldots, a_k)$; gli elementi a_i di tale k-upla sono chiamati **argomenti** della funzione ed il numero k è chiamato **arietà** della funzione. Una funzione con arietà 1 è detta funzione **unaria** mentre una funzione con arietà 2 è detta funzione **binaria**.

Una funzione il cui dominio o codominio sia un insieme di funzioni è detta **funzione di ordine superiore**. Un esempio di funzione di ordine superiore consiste nella funzione di mapping

$$map : (A \times \cdots \times A) \times (A \to B) \to (B \times \cdots \times B).$$

Tale funzione prende in input una tupla di elementi appartenenti ad un dato insieme A, prende in input una funzione che trasforma un elemento dell'insieme A in un elemento dell'insieme B e restituisce la tupla ottenuta applicando la funzione ad ogni elemento della tupla di partenza.

Una funzione di ordine superiore $f : S \to A \to B$ quando eseguita su di un elemento $s \in S$ produce in output una funzione $f(s) : A \to B$. Tale funzione $f(s)$ è detta **istanza parziale** della funzione f.

Un **predicato** è una funzione il cui codominio è l'insieme $\{\ 0, 1\ \}$. Intuitivamente un predicato è una funzione che prende in input un elemento e restituisce 0 (falso) se una determinata proprietà non vale per tale elemento, restituisce invece 1 (vero) altrimenti. Ad esempio il predicato "essere un numero primo" *prime* definito sui numeri naturali è una funzione il cui dominio è $\mathbb{N}$, il cui codominio è $\{\ 0, 1\ \}$, definita come segue

$$\{\ (x, y) \in \mathbb{N} \times \{\ 0, 1\ \} \mid y = 1 \text{ se } x \text{ è un numero primo}; y = 0 \text{ altrimenti }\ \}.$$

Un predicato il cui dominio è un insieme di k-uple $A \times \cdots \times A$ è detto

relazione su A.

A.1.4 Grafi

Un **grafo non orientato** (o più semplicemente un **grafo**) è un insieme di punti, alcuni dei quali possono essere connessi da linee. I punti sono chiamati **nodi** o **vertici** del grafo mentre le linee sono chiamate **archi**. La seguente figura è un esempio di grafo.

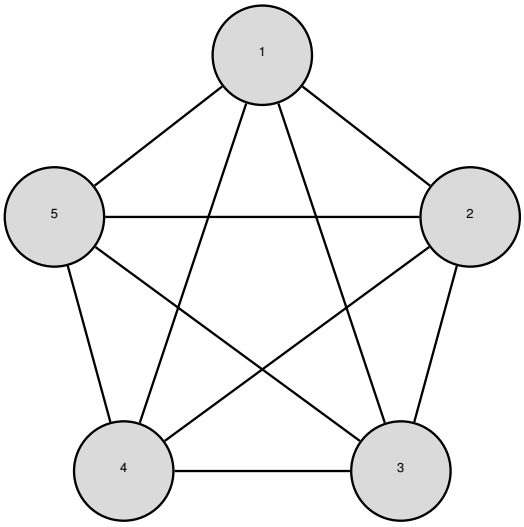

Figura 11: Grafo non orientato.

Il numero di archi incidenti in un dato nodo è chiamato **grado** di quel nodo. Nel grafo in Figura 11 tutti i nodi hanno grado 4. In un grafo tra ogni coppia di nodi non può mai esserci più di un arco. Se tra due nodi sono presenti due o più archi il grafo prende nome di **multigrafo**. In un grafo G che contiene i nodi i e j, la coppia (i, j) rappresenta l'arco che connette tali nodi. Poichè in un grafo non orientato le coppie (i, j) e (j, i) rappresentano lo stesso arco a volte si preferisce rappresentarlo mediante la notazione insiemistica $\{i, j\}$. Se V è l'insieme dei nodi di un grafo G ed E è l'insieme dei suoi archi allora è possibile scrivere $G = (V, E)$. Ad esempio

una descrizione formale del grafo in Figura 11 è

$$(\{\ 1, 2, 3, 4, 5\ \},$$
$$\{\ (1,2), (1,3), (1,4), (1,5), (2,3), (2,4), (2,5), (3,4), (3,5), (4,5)\ \}).$$

I grafi sono spesso utilizzati per rappresentare dati. I nodi di un grafo possono ad esempio essere città e gli archi le strade che connettono queste città. A volte per praticità è possibile etichettare i nodi e/o gli archi di un grafo. Ad esempio il grafo rappresentato in Figura 12 seguente contiene diverse città dell'Emilia-Romagna e gli archi tra queste città sono etichettati con la loro distanza.

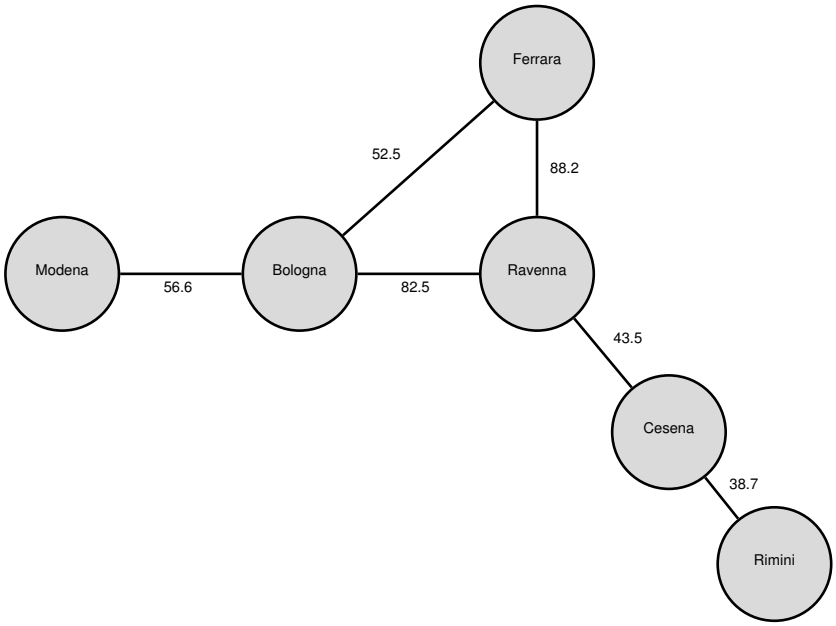

Figura 12: Grafo etichettato dell'Emilia-Romagna.

Un **cammino** in un grafo è una sequenza di nodi connessi da archi. Un **cammino semplice** è un cammino che non ripete nessun nodo. Un grafo si dice **connesso** se per ogni coppia di nodi esiste un cammino che li connette. Un cammino è un **ciclo** se comincia e finisce con lo stesso nodo. Un ciclo è detto **ciclo semplice** se contiene almeno tre nodi e ripete solo il primo e l'ultimo.

Se gli archi sono frecce piuttosto che linee allora il grafo prende nome di **grafo orientato** o di **grafo diretto**. Un cammino in cui tutte le frecce puntano nella stessa direzione è detto **cammino diretto**. Un grafo orientato si dice **fortemente connesso** se esiste un cammino diretto tra ogni coppia di nodi.

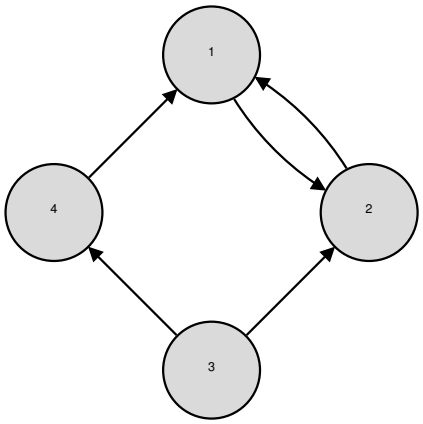

Figura 13: Grafo orientato.

A.1.5 Stringhe

Un **alfabeto** è un qualsiasi insieme non vuoto. Gli elementi di un alfabeto sono chiamati **simboli** dell'alfabeto. Di solito si utilizza la lettera Greca sigma maiuscola per rappresentare un alfabeto. I seguenti sono esempi di alfabeti

- $\Sigma_1 = \{\ 0, 1\ \}$;
- $\Sigma_2 = \{\ a, b, c, d, e, f, g, h, i, l, m, n, o, p, q, r, s, t, u, v, z\ \}$;
- $\Sigma_3 = \{\ \odot, \square, *\ \}$.

Una **stringa di caratteri su un alfabeto** o più semplicemente una **stringa** è una sequenza finita di caratteri appartenenti a quell'alfabeto, scritti l'uno di seguito all'altro senza alcun separatore. Ad esempio se $\Sigma_1 = \{\ 0, 1\ \}$ allora 001011 è una stringa sull'alfabeto Σ_1. Se w è una stringa sull'alfabeto Σ, la

lunghezza di w (la cui notazione relativa è $|w|$) è il numero di caratteri che contiene. La stringa con zero caratteri è chiamata **stringa vuota** ed è spesso rappresentata con il simbolo ϵ. Se w ha lunghezza n è possibile scrivere $w = w_1 w_2 \dots w_n$ con $w_i \in \Sigma$. Date due stringhe x di lunghezza m ed y di lunghezza n, si definisce **concatenazione** di x e di y (scritto xy) la stringa ottenuta facendo seguire la stringa y alla stringa x, ossia $x_1 \dots x_m y_1 \dots y_n$. Per indicare la concatenazione di una stringa con se stessa k volte si usa la notazione esponenziale x^k.

L'insieme delle stringhe su di un dato alfabeto Σ viene indicato con il simbolo Σ^*. Pertanto se $\Sigma = \{ 0, 1 \}$ allora

$$\Sigma^* = \{ \epsilon, 0, 1, 00, 01, 10, 11, 000, \dots \}.$$

A.2 Teoria della calcolabilità

Nell'ultimo secolo svariate personalità del mondo informatico si sono occupate di definire formalmente cosa significhi "calcolare"; tra queste spiccano per genialità Alan Turing ed Alonzo Church. Il primo è autore di un celebre modello matematico noto come **Macchina di Turing**, proposto nel 1936. La Macchina di Turing è sostanzialmente una versione primitiva di un calcolatore moderno. E' dotata di un nastro infinito che rappresenta la sua memoria, di una testina che le permette di leggere o scrivere simboli su tale nastro e di un programma che regola gli spostamenti della testina e di conseguenza il modo con cui tale macchina opera.

Alonzo Church è invece autore di un formalismo matematico noto come **Lambda Calcolo** che si basa su un sistema di regole per la riscrittura di stringhe ed è stato provato di potere equivalente a quello delle Macchine di Turing. Sebbene entrambi i formalismi discussi siano di indubbia valenza e permettano di calcolare soluzioni ad una vastissima gamma di problemi, ne esistono alcuni che non sono in grado di risolvere. Lo studio di questi problemi prende nome di **teoria della calcolabilità**.

A.2.1 La Macchina di Turing

La Macchina di Turing è uno strumento di calcolo particolarmente simile ad un computer. E' dotata di un nastro di lavoro che contiene infinite celle utilizzabili per memorizzare dei simboli (al massimo un simbolo per ogni cella). Tale nastro ha un punto di inizio e a partire da tale punto si estende all'infinito. In principio il nastro contiene nelle sue primissime celle l'input della Macchina di Turing e tutte le altre celle sono vuote. La testina (ossia la cella del nastro a cui la Macchina di Turing ha accesso in un dato istante) coincide con la prima cella del nastro. La macchina continua a calcolare fino a che non decide di produrre un output. Gli output leciti di una Macchina di Turing sono *accettazione* e *rifiuto*. Quando eseguita su di uno specifico input, una Macchina di Turing può accettarlo, rifiutarlo o continuare la propria computazione all'infinito. La definizione formale di suddetto strumento di calcolo è fornita di seguito

Definizione A.1 (Macchina di Turing)**.**
Una **Macchina di Turing** (TM) è una 7-upla, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ dove Q, Σ, Γ sono insiemi finiti e

1. Q è l'insieme degli stati,
2. Σ è l'alfabeto di input non contenente il simbolo rappresentante la **cella vuota** $\sqcup$,
3. Γ è l'alfabeto del nastro, $\sqcup \in \Gamma$ e $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{ L, R \}$ è la funzione di transizione,
5. $q_0 \in Q$ è lo stato iniziale,
6. $q_{accept} \in Q$ è lo stato di accettazione,
7. $q_{reject} \in Q$ è lo stato di rifiuto, $q_{reject} \neq q_{accept}$.

Una Macchina di Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ calcola nel modo seguente. In primo luogo M riceve il suo input $w = w_1 \dots w_n \in \Sigma^*$ nelle

prime n celle del nastro. La testina è posizionata sulla prima cella. Dopo di che la computazione procede in accordo a quanto espresso dalla funzione di transizione δ. Si pensi a tale funzione di transizione come un insieme di regole della forma: se la Macchina di Turing si trova in un determinato stato e la cella indicata dalla testina contiene un determinato simbolo allora la Macchina di Turing sostituisce il simbolo in quella cella con un'altro (anche lo stesso), sostituisce lo stato in cui si trova con un altro (anche lo stesso) e muove la testina di una cella verso sinistra (L) o verso destra (R). Ad esempio se la funzione di transizione di M dichiara

$$\delta(q_3, a) = (q_5, b, R)$$

allora quando tale macchina si trova nello stato q_3 e la cella indicata dalla testina contiene il simbolo a, modificherà il proprio stato con q_5, sostituirà il simbolo a con il simbolo b e muoverà la testina di una posizione verso destra.

Si noti che se in un determinato istante M tenta di muovere la propria testina verso sinistra quando questa è posizionata sulla prima cella del nastro, la testina non si muove affatto. Non appena la Macchina di Turing sostituisce il proprio stato corrente con lo stato di accettazione o di rifiuto la computazione termina.

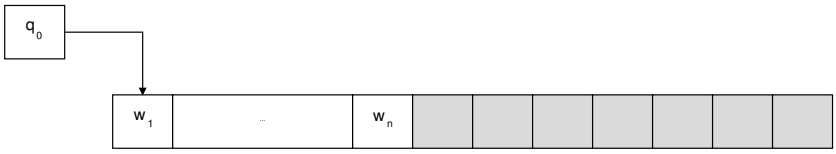

Figura 14: Configurazione iniziale di una Macchina di Turing.

Durante la computazione M attraversa diversi stati, modifica il suo nastro e muove la testina lungo di questi. La tripla composta dal contenuto del nastro, lo stato corrente di M e la posizione della testina è chiamata **configurazione** della Macchina di Turing M. Le configurazioni di una Macchina di Turing sono spesso rappresentate in modo particolare. Dato uno stato q e due stringhe u e v la stringa uqv rappresenta la configurazione della Macchi-

na di Turing che si trova nello stato q, il cui nastro contiene la stringa uv e la cui testina è posizionata sul primo simbolo di v. Ad esempio la stringa

$$1011q_701111$$

rappresenta la configurazione di una Macchina di Turing che si trova nello stato q_7, il cui nastro contiene 101101111 e la cui testina si trova sul secondo 0. Tale configurazione è rappresentata in Figura 15.

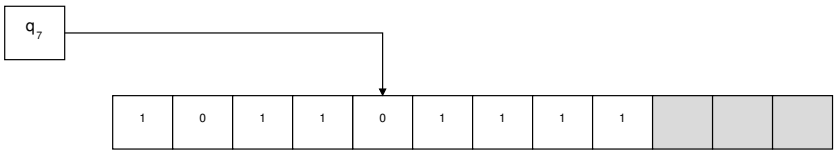

Figura 15: Configurazione $1011q_701111$.

Data una Macchina di Turing M ed una stringa w è prassi utilizzare la notazione M(w) per intendere l'atto dell'esecuzione della macchina M su input w.

Poichè una Macchina di Turing è a tutti gli effetti una struttura finita (dotata di un alfabeto finito, di stati finiti e di una funzione di transizione finita) è possibile codificarla sfruttando una stringa finita di simboli. Per indicare la stringa di simboli che **codifica** una Macchina di Turing M si usa spesso la notazione $<$ M $>$. E' perfettamente lecito che l'input di una data Macchina di Turing sia la codifica di un'altra Macchina di Turing. In fondo la codifica di una Macchina di Turing è una stringa e le Macchine di Turing prendono in input delle stringhe. Il lettore non sia fuorviato da questo espediente; esistono svariati esempi di programmi di uso quotidiano che prendono in input altri programmi. Si pensi ad esempio ad un compilatore[3] che traduce programmi *C* in *Python*.

[3]Un compilatore è un programma che prende in input un'altro programma scritto in un determinato linguaggio di programmazione e lo traduce in un programma equivalente scritto in un'altro linguaggio.

A.2.2 La tesi di Church-Turing

Secondo Alan Turing qualsiasi funzione che sia algoritmicamente calcolabile è calcolabile mediante una Macchina di Turing. Questa congettura dal contenuto ragionevole dichiara che l'atto stesso di calcolare coincide con il semplice formalismo delle Macchine di Turing. Se un calcolo può essere svolto da un moderno supercomputer oppure da uno studente delle scuole elementari che espande formule scrivendo su di un pezzo di carta allora può essere svolto anche da una Macchina di Turing. Questa intuizione prende nome di **Tesi di Church-Turing** in quanto, come discusso in precedenza, ambedue queste personalità hanno sviluppato indipendentemente e più o meno nello stesso periodo di tempo due formalismi equivalenti che forniscono pertanto, in accordo a tale tesi, due nozioni che definiscono allo stesso modo il concetto di algoritmo e di funzione calcolabile.

A.2.3 Il problema della terminazione

Sebbene le Macchine di Turing permettano di calcolare funzioni anche molto complesse come la radice quadrata di numeri a cento cifre, il modo migliore di organizzare gli orari delle lezioni di un intero ateneo (tenendo conto della disponibilità delle aule e di determinate preferenze) o il percorso più breve e meno costoso tra Bologna e New York, esistono numerosi problemi di incredibile importanza pratica che non sono in grado di risolvere.

Come discusso in precedenza una Macchina di Turing eseguita su di un dato input può terminare la propria esecuzione dopo un determinato lasso di tempo accettando o rifiutando tale input oppure continuare la propria computazione all'infinito. E' importante osservare che nella maggior parte dei casi è preferibile che una Macchina di Turing termini la sua computazione per ogni input; una Macchina di Turing in grado di determinare se un numero è primo oppure no non ha motivo di computare all'infinito su nessun input. Se su un determinato input non terminasse mai la propria computazione allora, con ogni probabilità, sarebbe stata codificata in maniera erronea.

Ciò motiva il desiderio di sviluppare una Macchina di Turing H che, presa in input la codifica di un'altra Macchina di Turing $<$ M $>$ ed un input w termini la propria esecuzione accettando se la macchina M eseguita sull'input w termina la propria esecuzione accettandolo o rifiutandolo; termini invece rifiutando se la macchina M non termina affatto la propria esecuzione quando eseguita su tale input.

Purtroppo H non può esistere. Tale risultato (ad opera di Alan Turing ma profondamente basato sul lavoro del celeberrimo Georg Cantor) prende nome di **indecidibilità del problema della terminazione**. L'intuizione alla base di tale risultato è incredibilmente semplice. Infatti, se H esistesse, sarebbe possibile sviluppare un'altra Macchina di Turing D definita nel modo seguente.

$$D(< M >) = \begin{cases} \text{termina accettando} & \text{se } M(< M >) \text{ non termina,} \\ \text{non termina} & \text{se } M(< M >) \text{ termina.} \end{cases}$$

A tutti gli effetti la Macchina di Turing D prende in input la codifica di un'altra Macchina di Turing $<$ M $>$ e sfruttando la Macchina di Turing H determina se M termina quando eseguita sua stessa codifica $<$ M $>$. Infine D si comporta esattamente al contrario di quanto farebbe M: terminando la propria computazione accettando quando M non termina e non terminando volontariamente quando M invece termina (accettando o rifiutando la sua stessa codifica). Cosa succederebbe se D fosse eseguita sulla sua stessa codifica? E' banale evincere che tale situazione si rivelerebbe paradossale.

$$D(< D >) = \begin{cases} \text{termina accettando} & \text{se } D(< D >) \text{ non termina,} \\ \text{non termina} & \text{se } D(< D >) \text{ termina.} \end{cases}$$

Di fatti $D(< D >)$ termina accettando se e solamente se non termina la propria esecuzione e non termina se invece termina. Ciò è evidentemente assurdo in quanto una Macchina di Turing eseguita su di un dato input può terminare la propria esecuzione accettando o rifiutando tale input oppure non terminare

affatto, ma chiaramente non può fare entrambe le cose contemporaneamente.

In quanto l'unica assunzione proposta è l'esistenza della Macchina di Turing H tale assunzione è erronea, pertanto H non può esistere.