Chiara Paci

Progetto di un'applicazione per la raccolta di informazioni linguistiche da documenti storici d'archivio: discussione generale e database morfologico.

I documenti d'archivio, di solito considerati per il valore storico, rappresentano dal punto di vista linguistico una finestra sulla lingua comune non letteraria dei secoli passati, anche se non strettamente colloquiale: si tratta per lo più di testi commerciali o diplomatici formalmente corretti e scritti da persone colte.

Lo strumento che si vuole realizzare servirà ad agevolare la raccolta dei fatti linguistici da questi documenti, in modo da minimizzare gli errori e i passi ripetitivi.

Si è ritenuto più utile concentrarsi sul come strutturare i dati da raccogliere piuttosto che sulla logica dell'applicazione per l'analisi dei testi o la reportistica (v. sezione 1), in quanto la seconda è molto più facile da modificare e adattare una volta definita la base dati. Ma è ovvio che la definizione della base dati non può del tutto prescindere da come questi dati verranno poi utilizzati.

Questo lavoro, che è solo una fase iniziale del progetto, comprende l'analisi preliminare e la realizzazione del modulo per le regole morfologiche (v. sezione 2).[1]

1. Scopo e vincoli dell'applicazione

L'applicazione agisce in modo supervisionato:

1. l'utente definisce le regole base di parsing, stabilendo come individuare le parole e cosa delimita i periodi;
2. l'utente trascrive un documento come testo senza tag, a parte quelli di formattazione (allineato a destra o sinistra, sottolineato, ecc.); i tag di formattazione sono definiti dal programma;
3. l'utente imposta delle regole grammaticali e inserisce informazioni sul lessico;
4. il programma analizza il testo, riportando se e come le regole vengono verificate dal testo;
5. l'utente aggiusta le regole finché non sono conformi al testo, dopodiché passa al documento successivo.

Il programma finale dovrà essere in grado di:

- dato un insieme di documenti, individuare quali regole lo riguardano (e solo quelle) e quali parti non corrispondono a nessuna regola;
- data una regola o un vocabolo, rintracciare in quali documenti vengono usati e riportare esempi di uso;

1. Un prototipo del programma, a cui si riferiscono gli esempi, è stato realizzato col framework Django 2.2 [Django]. Il motore di database usato è PostgreSQL 8.4.11 [PostgreSQL] e il linguaggio di programmazione è Python 2.6 [Python]. Il framework Django è stato scelto perché, pur mantenendosi fedele alla struttura di un database relazionale, si preoccupa di gestire i passi ripetitivi (ad esempio la creazione di tabelle intermedie per le relazioni molti a molti).

- stilare una grammatica e un dizionario di un insieme di documenti;
- raccogliere dati per più lingue.

I testi devono essere forniti in modo omogeneo e pulito al programma: rimane compito dell'utente sistemarli in modo che siano analizzabili, per esempio introducendo degli spazi o dei segni di interpunzione per quelle lingue che non li prevedono. L'utente è libero di scegliere cosa o come inserire, ma al minimo devono essere forniti al programma (v. sezione 3.1):

- uno o più delimitatori di periodo;
- regole univoche su come separare le parole.

Regole grammaticali e lessico vengono desunte dai documenti, ma spesso sono integrate da uno studio collaterale su materiale di riferimento (dizionari, grammatiche, altri documenti, precedenti ricerche, ecc.): l'utente può quindi inserire più regole di quelle necessarie (per esempio, può introdurre l'intero paradigma di un verbo anche se nel documento compare una sola voce). In questo caso, nell'output finale il programma escluderà tutto quello che non trova riscontro nei documenti.

2. Considerazioni preliminari

2.1. Schema generale dell'applicazione

In figura 1 è riportato lo schema di tutta l'applicazione: la parte centrale rappresenta il database. L'applicazione ha dei moduli pressoché indipendenti per l'esecuzione dei vari compiti, ognuno dei quali utilizza il database in modo diverso.

2.1.1. La base dati

La base dati è di tipo relazionale, ossia mantiene i dati in forma tabellare e definisce relazioni tra le tabelle.[2]

Le caratteristiche che hanno fatto scegliere un database relazionale, rispetto alle alternative, sono:

2. Ogni riga di una tabella è detta anche *record*, ogni colonna è detta anche *campo* ed è di un tipo predefinito (intero, float, stringa, boolean, ecc.). Ogni record contiene uno e un solo valore per ogni campo.

   Esistono tre tipi di relazioni fra i dati di due tabelle A e B:

   - *relazione uno a uno*, quando ad ogni record della tabella A corrisponde uno e un solo record della tabella B: in questo caso, la tabella B ha un campo che contiene l'identificativo di un record della tabella A e questo campo ha il vincolo di essere unico;
   - *relazione uno a molti*, quando un record della tabella A può corrispondere a più di un record della tabella B: in questo caso, la tabella B ha un campo che contiene l'identificativo di un record della tabella A, ma questo campo non ha il vincolo di essere unico e il valore contenuto può essere ripetuto;
   - *relazione molti a molti*, quando un record della tabella A può corrispondere a più di un record della tabella B e viceversa: in questo caso, esiste un ulteriore tabella C (detta *tabella di relazione*) che ha due campi, uno che contiene l'identificativo di un record della tabella A e un altro che contiene l'identificativo di un record della tabella B; la tabella di relazione può contenere anche dati aggiuntivi che contribuiscono a definire la relazione.

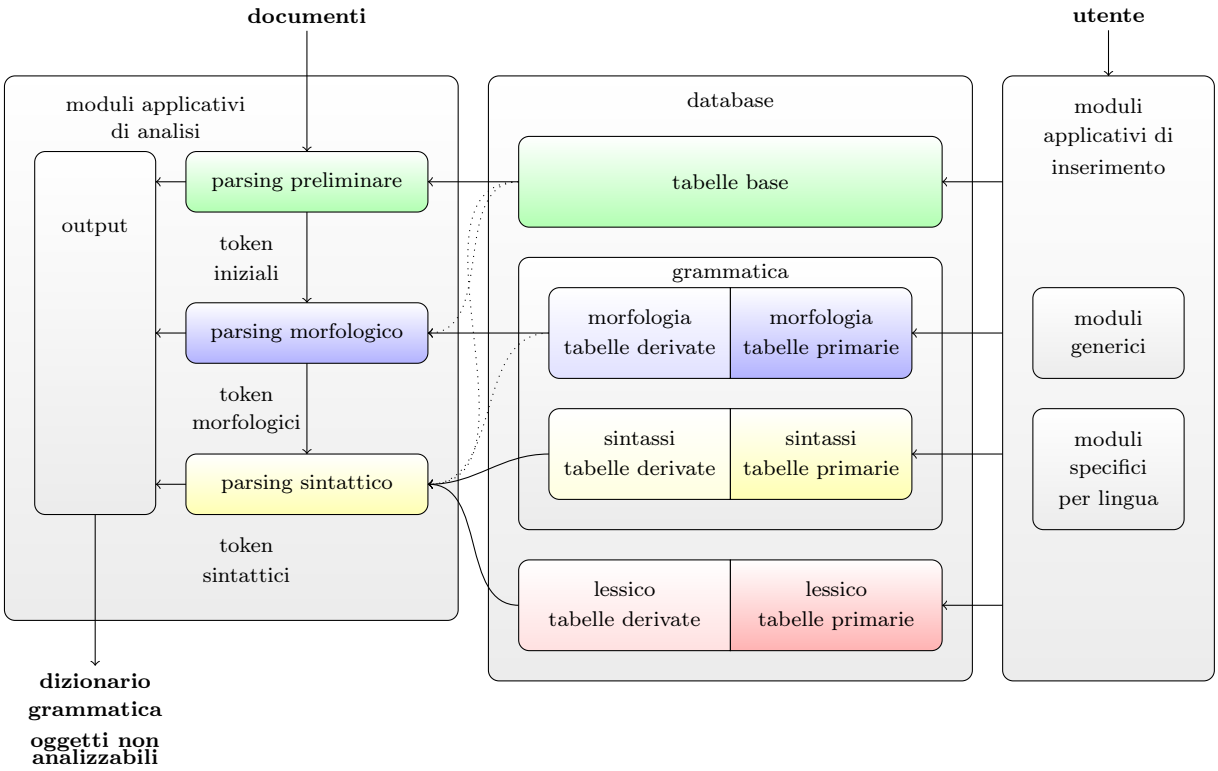


Figura 1: Schema dell'applicazione.

- la possibilità di definire relazioni tra i dati anche complesse (si veda, per esempio, la definizione di descrizione in sezione 3.2);
- la possibilità di gestire in modo efficiente grandi quantità di dati, grazie soprattutto a ottimizzazioni nei prodotti e tecniche consolidate di definizione dei dati e programmazione: allo stato attuale è possibile con un computer personale eseguire query su tabelle con decine di milioni di record pressoché in tempo reale.

Questi vantaggi si pagano con una certa rigidità dei dati dovuta alla forma tabellare: un record avrà sempre quel numero e quel tipo di campi. A questo si può ovviare usando le relazioni e definendo un oggetto con più di una tabella: il database risulta più complesso, però anche qui è possibile usare tecniche di programmazione e librerie che consentono di gestire la complessità.

2.1.2. Inserimento dei dati

In fase di inserimento sono previsti più moduli per agevolare l'utente nell'inserire i dati. Questi moduli sono tanto più necessari quanto più la rappresentazione dei dati è astratta. Per esempio, una forma verbale (poniamo la prima persona singolare dell'indicativo presente) potrebbe essere descritta da una struttura di questo tipo (v. sezione 2.2):

$$\left[\begin{array}{lcl} \text{tempo} & = & \text{presente} \\ \text{modo} & = & \text{indicativo} \\ \text{persona} & = & \left[\begin{array}{lcl} \text{persona} & = & \text{prima} \\ \text{numero} & = & \text{singolare} \end{array}\right] \end{array}\right]. \tag{2.1}$$

È chiaro che diventa tedioso dover inserire questa descrizione per ogni voce verbale che si definisce: serve quindi un modulo che consenta all'utente di inserire le varie forme di un paradigma verbale, preoccupandosi di completare le informazioni e inserirle nel database. Questi moduli non potranno essere generici, ma saranno specifici per una data lingua (o per un gruppo di lingue simili).

Niente vieta, inoltre, di scrivere moduli in grado di importare dati da altre applicazioni o di generarli in modo automatico.

2.1.3. L'analisi

L'analisi si comporta come una successione di filtri: ogni passo riceve in input una sequenza di token e produce in output un'altra sequenza di token. Un token è un oggetto che associa porzioni di testo a informazioni specifiche del livello di analisi.

Per esempio, la sequenza *la panchina del parco* viene suddivisa dal parser preliminare:

$$\begin{array}{l} p_1{=}(\text{'la'},\text{stringa}),\ p_2{=}(\text{' '},\text{spazio}),\ p_3{=}(\text{'panchina'},\text{stringa}),\ p_4{=}(\text{' '},\text{spazio}), \\ p_5{=}(\text{'del'},\text{stringa}),\ p_6{=}(\text{' '},\text{spazio}),\ p_7{=}(\text{'parco'},\text{stringa})\ , \end{array} \tag{2.2}$$

dall'analizzatore morfologico (tralasciando i dettagli):

$$\begin{array}{l} m_1{=}(p_1,\text{articolo}),\ m_2{=}(p_2,\text{non-parola}),\ m_3{=}(p_3,\text{nome}), \\ m_4{=}(p_4,\text{non-parola}),\ m_5{=}(p_5,\text{preposizione}),\ m_6{=}(p_5,\text{articolo}), \\ m_7{=}(p_6,\text{non-parola}),\ m_8{=}(p_7,\text{nome}), \end{array} \tag{2.3}$$

e da quello sintattico (anche qui semplificato):[3]

$$s_1{=}((m_1,m_2,m_3),\text{NP}),\ s_2{=}((m_4,m_5),\text{PP}),\ s_3{=}((m_6,m_7,m_8),\ \text{NP}). \tag{2.4}$$

2.1.4. Tabelle primarie e derivate

Le regole grammaticali non vengono definite per oggetti linguistici, ma per classi: ad esempio, ci sarà una regola che stabilisce che per tutti i verbi regolari della prima coniugazione la prima persona singolare del presente indicativo sarà in '-o', ma non una regola che dica che per il verbo 'amare' questa voce sarà 'amo'. Quindi l'utente inserirà da una parte la regola per le voci di un verbo della prima coniugazione, dall'altra il verbo 'amare' classificandolo come della prima coniugazione. L'associazione è in questo caso implicita e retta dalle descrizioni dei due oggetti (v. sezione 2.2). Queste tabelle su cui agisce l'utente sono *primarie*.

3. I sintagmi sono indicati con le convenzioni definite in [Cecchetto, 2002].

L'applicazione (nello specifico il modulo di analisi morfologica) ha però bisogno di utilizzare una relazione esplicita, cioè ha bisogno di sapere che la voce 'amo' corrisponde al verbo 'amare' e alla regola 'prima persona singolare presente indicativo'. È necessario quindi prevedere un passaggio in più tra l'inserimento dei dati e l'analisi, che si preoccupi di precalcolare le relazioni esplicite e salvarle in tabelle apposite (le tabelle *derivate*).

2.2. Descrizioni

Per classificare i vari oggetti linguistici si ricorre a *descrizioni*, ad esempio:

$$\begin{bmatrix} \text{modo} & = & \text{indicativo} \\ \text{tempo} & = & t \\ \text{persona} & = & \begin{bmatrix} \text{persona} & = & \text{prima} \\ \text{numero} & = & \text{singolare} \end{bmatrix} \end{bmatrix} \tag{2.5}$$

Si tratta di un insieme di coppie attributo/valore, dove un valore può essere una costante, una variabile (indicate in corsivo in (2.5)) o un'altra descrizione. Le operazioni possibili su una coppia di descrizioni sono la *sussunzione* e l'*unificazione* [Kay, 1992, p. 11-15; Fenstad, Langholm, Vestre, 1992, p. 33-34].

Una descrizione B *sussume* A ($B \sqsupseteq A$ o $A \sqsubseteq B$) quando:

1. B ha tutti gli attributi definiti in A (può averne di più);
2. se un attributo in A e in B ha come valore una variabile o una costante, i due valori devono essere uguali;
3. se un attributo in A ha come valore una descrizione A_1, in B deve avere una descrizione B_1, e $A_1 \sqsubseteq B_1$.

L'*unificazione* di A e B ($A \cup B$) è la minima descrizione C che sussume sia A che B. In pratica, se $C = A \cup B$, allora:

1. C contiene tutti gli attributi di A e B;
2. per gli attributi che compaiono solo in A, il valore in C è quello in A;
3. per gli attributi che compaiono solo in B, il valore in C è quello in B;
4. per gli attributi che compaiono sia in A che in B e hanno come valore una costante o una variabile, i due valori devono essere uguali; se non lo sono l'unificazione *fallisce*, se lo sono il valore in C è lo stesso che in A e B;
5. per gli attributi che compaiono sia in A che in B e hanno come valore una descrizione (rispettivamente A_1 e B_1), il valore in C è la descrizione $C_1 = A_1 \cup B_1$; se l'unificazione $A_1 \cup B_1$ fallisce, fallisce anche $A \cup B$.

Le descrizioni vengono utilizzate da tutte le componenti dell'applicazione: il lavoro dei moduli è pensato come un'insieme di operazioni sulle descrizioni degli oggetti.

2.3. Morfologia

I fenomeni morfologici da registrare sono [Simone, 2008, p. 136]:

- la *derivazione*,
- la *flessione*,
- la *composizione*.

Non sono considerate le parole complesse (ad esempio 'mettere in moto' o i tempi composti dei verbi), rimandando all'analisi successiva. Ognuno di questi fenomeni corrisponde a un oggetto dell'applicazione (v. sezione 4.1, sezione 4.2 e sezione 4.3).

Oltre a questi, vengono definiti altri quattro oggetti:

- *radice lessicale* di un termine ('amic⊢');
- *radice tematica*, una volta applicata una regola di derivazione ('amicizi-'); se la regola di derivazione è nulla, la radice tematica è identica a quella lessicale (ma non è lo stesso oggetto);
- *parola*: una radice tematica a cui è stata applicata una regola di flessione ('amicizia', 'amicizie'); ogni radice tematica è rappresentata nel dizionario da una particolare parola, detta *voce del dizionario*;
- *parola composta*: un insieme di parole unite da una regola di composizione ('di'+'la'='della').

Di questi, è chiaro che radici tematiche, parole e parole composte sono oggetti derivati:

$$\left.\begin{array}{r}\text{radici lessicali}\\ \text{derivazione}\end{array}\right\}\ \text{radici tematiche}\ \left\{\begin{array}{r}\\ \text{flessione}\end{array}\right.\quad\left.\begin{array}{r}\text{parole}\\ \\ \text{composizione}\end{array}\right\}\ \text{parole composte} \tag{2.6}$$

Sfruttando le caratteristiche dei database relazionali, è possibile ipotizzare un approccio esaustivo per quanto riguarda l'analisi morfologica. In base ai dati registrati (radici e regole) vengono calcolate tutte le possibili forme (parole e parole composte).

In fase di analisi, i token del testo, prodotti da un primo parsing (v. sezione 3.1), vengono associati agli oggetti corrispodenti grazie a una semplice ricerca.

Un token può venire associato a più di un oggetto (ad esempio, 'la' verrà riconosciuto sia come articolo che come pronome): l'ambiguità, se è possibile farlo, verrà risolta nel passo successivo.

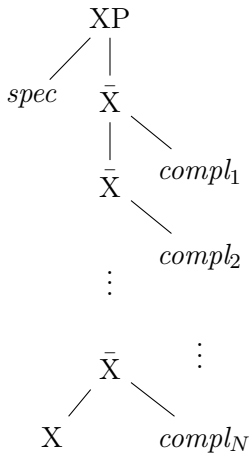
2.4. Sintassi

Se l'approccio esaustivo può andar bene nel caso della morfologia, non è pensabile utilizzarlo per quanto riguarda la sintassi, non solo perché il numero frasi possibili è molto maggiore del numero di parole: la generazione di frasi è ricorsiva e con un metodo esaustivo si arriverebbe a generare infinite frasi. È necessario quindi un metodo di parsing basato su regole.

Per la descrizione delle regole è stata scelta la teoria X-barra,[4] ossia le regole verranno rappresentate con oggetti (*sintagmi*) del tipo:[5]

4. Seguiamo per formalismi e definizioni, dove non altrimenti specificato, [Cecchetto, 2002].

5. Usare una descrizione o un'altra, purché porti a una grammatica context-free, è in effetti una questione di gusti personali e tipo di ricerca. Niente vieta di aggiungere anche qui un modulo per la conversione, per esempio, da una grammatica definite-clause o di prevedere anche metodi ibridi di specificazione.



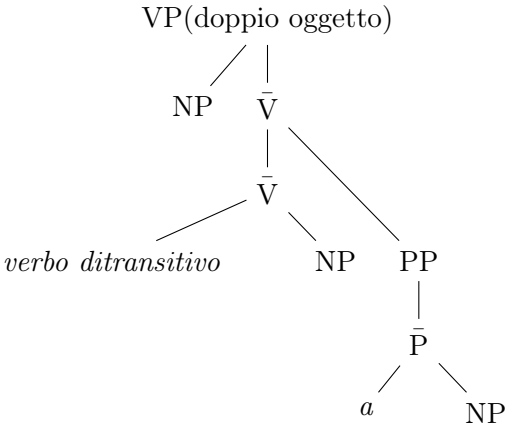
(2.7)

Quindi ogni sintagma (XP) avrà una testa (X), uno specificatore (spec) e uno o più complementi, che potranno essere attributi o aggiunti. Ognuno di questi oggetti dovrà avere una descrizione e la descrizione del sintagma sarà l'unificazione di tutte le descrizioni. Ognuno di loro potrà essere:

- vuoto;
- qualcosa di sottinteso (pro e PRO);
- una traccia di un movimento;
- una parte di token morfologico (la flessione nel caso del sintagma della flessione IP, per esempio);
- una sequenza di token morfologici;
- un altro sintagma.

È necessario prevedere anche degli oggetti che descrivano i movimenti sintattici: in questo modo è possibile specificare quali casi e ruoli tematici vengono assegnati ai vari sintagmi, dato che questi vengono assegnati a livello di struttura profonda [Cecchetto, 2002, p. 145].

È chiaro che non è pensabile di svolgere quest'analisi senza includere nella descrizione anche informazioni che provengono dal modulo sul lessico [Del Monte, 2008, p. 115-117]: i due tipi di informazione possono restare distinti al momento dell'inserimento delle regole, ma non al momento dell'analisi. Cioè, l'utente può inserire una regola del tipo:



(2.8)

e, separatamente, classificare 'dare' e 'assegnare' come verbi ditransitivi in due momenti distinti. Ma il programma, incontrando ad esempio 'assegna', dovrà tenere conto non solo della sua descrizione

morfologica (verbo, terza persona, ecc.), ma anche di quella lessicale (quindi, in questo caso, che si tratta di un verbo classificato come verbo ditransitivo).

Il parser ipotizzato per l'analisi è un parser canonico LR [Aho, Sethi, Ullman, 1986, p. 215-247], in cui però viene introdotto il concetto di stack strutturato a grafo [Tomita, 1985; Tomita, 1987] per gestire grammatiche ambigue.[6]

Una grammatica context-free $G = (V, \Sigma, P, S)$ è definita come [Aho, Sethi, Ullman, 1986, p. 26-27,165-166]:

1. un insieme di simboli terminali o token, Σ; nel nostro caso sarebbero le parole, ma è più conveniente utilizzare le classi di equivalenza individuate dalle descrizioni; per questo, bisognerà pianificare delle tabelle derivate opportune per velocizzare l'associazione tra il token da analizzare e la corrispondente descrizione usata come simbolo terminale;
2. un insieme di simboli non-terminali, V, ognuno dei quali rappresenta un diverso tipo di frase (più o meno coincide con l'insieme dei sintagmi);
3. un particolare simbolo non-terminale, S, che rappresenta un'intera sentenza;
4. un insieme di regole di produzione P (più o meno coincide con le regole grammaticali); le regole di produzione hanno un simbolo non-terminale a sinistra e una sequenza di simboli terminali e non-terminali a destra.

2.4.1. Parser LR

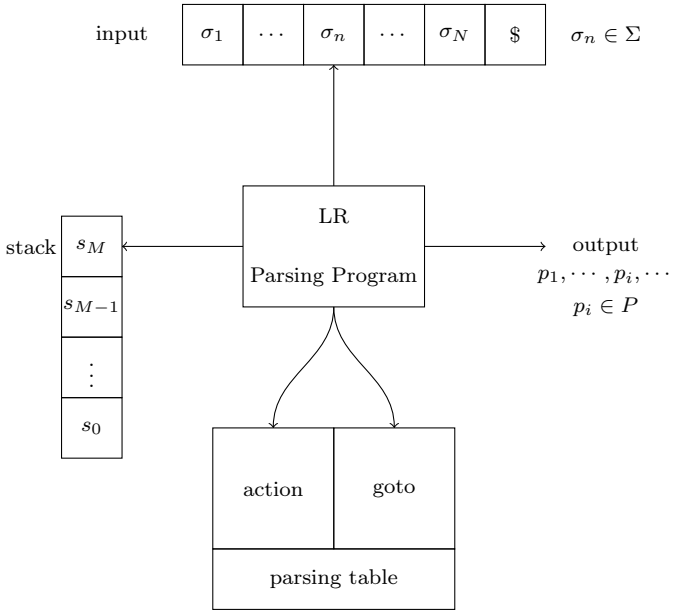
Un parser LR è un parser bottom-up shift-reduce nonbacktracking, schematizzato in figura 2, che utilizza una tabella (*tabella di parsing*) per identificare le transizioni di stato, dati una coppia (s_i, σ_j) di stati (righe) e simboli (colonne).

Esistono diversi algoritmi che consentono di generare la tabella di parsing a partire dalla definizione della grammatica (per i quali si rimanda a [Aho, Sethi, Ullman, 1986, p. 221-247]), che differiscono tra loro in base alla complessità e al numero di grammatiche in grado di tradurre. Le tabelle generate, anche se sono diverse per la stessa grammatica, hanno comunque la stessa struttura di figura 2. A fronte di una coppia (s_i, σ_j) ci possono essere quattro azioni [Aho, Sethi, Ullman, 1986, p. 217-218]:

1. *shift k (s_k)*: inserisce s_i sullo stack, si posiziona dopo σ_j e passa allo stato s_k;
2. *reduce n (r_n)*: dà in output la regola di produzione $p_n = v_h \rightarrow t_1 \cdots t_M$, toglie dallo stack M stati, esponendo lo stato s_l; come stato, sceglie quello indicato da (s_l, v_h) (area *goto*) e rimane prima di σ_j;
3. *accept (acc)*: termina il parsing con successo;
4. *error (cella vuota)*: segnala un errore.

Quello che bisogna definire quindi sono degli oggetti per le *azioni* e gli *stati*, e delle relazioni che associno i simboli (terminali e no, ossia le classi di equivalenza e i sintagmi) con gli stati e le azioni.

6. Questo parsing dovrà essere preceduto da una conversione tra la descrizione data dall'utente e una grammatica in forma normale, conversione che è comunque banale e tralasciamo qui.



p_1	E →E + T
p_2	E →T
p_3	T →T * F
p_4	T →F
p_5	F →(E)
p_6	F →a
	Σ={a,+,*,(,)}
	V={E,T,F}
	S=E

	action (terminali)						*goto* (non term.)		
state	a	+	*	(	)	$	E	T	F
0	s_5			s_4			s_1	s_2	s_3
1		s_6				**acc**			
2		r_2	s_7		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	s_5			s_4			s_8	s_2	s_3
5		r_6	r_6		r_6	r_6			
6	s_5			s_4				s_9	s_3
7	s_5			s_4					s_{10}
8		s_6			s_{11}				
9		r_1	s_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

Figura 2: Parser LR ed esempio di parsing table per la grammatica indicata [Aho, Sethi, Ullman, 1986, p. 217-219]. I simboli terminali sono indicati tra apici.

2.4.2. Parser LR con stack strutturato a grafo

In tabella 1 è riportato un esempio di tabella di parsing ottenuta da una grammatica ambigua. Come si vede nelle celle evidenziate, ci sono dei casi in cui una coppia stato-simbolo (s_i, σ_j) corrisponde a più azioni possibili. In questo caso un parser LR standard fallirebbe.

Esiste un altro tipo di algoritmo, che procede normalmente come un parser LR standard. Quando incontra una sequenza di azioni, l'esecuzione viene suddivisa in più rami, ognuno dei quali procede in parallelo finché non si raggiunge uno stato comune [Tomita, 1985; Tomita, 1987]. È più semplice vederlo con uno schema. L'algoritmo LR standard procede per passi successivi, con un'azione per ogni passo, ad esempio:

Tabella 1: Esempio di tabella di parsing per una grammatica ambigua

	action (terminali)						*goto* (non term.)		
state	a	b	c	d	e	$	X	Y	Z
0	s_5			s_4			s_1	s_2	s_3
1		s_6				**acc**			
2		r_2	s_7,r_6		r_2	r_2			
3		r_4	r_4		r_4	r_4			
4	s_5			s_4			s_8	s_2	s_3
5		r_6	r_6		r_6	r_6			
6	s_5			s_4,r_3				s_9	s_3
7	s_5			s_4					s_{10}
8		s_6			s_{11}				
9		r_1	s_7		r_1	r_1			
10		r_3	r_3		r_3	r_3			
11		r_5	r_5		r_5	r_5			

$$\cdots \to s_{42} \to r_7 \to s_{26} \to s_{33} \to r_{69} \to \cdots \tag{2.9}$$

mentre per il parser di Tomita l'esecuzione procede come segue:

$$\begin{array}{ccccccccccccc} \cdots & \to & s_{42} & \to & s_7 & \to & s_{26} & \to & r_{33} & \to & s_{69} & \to & s_{36} & \to & r_{74} & \to & \cdots \\ & & & \searrow & & & & \nearrow & & \nearrow & & & & \searrow & & & \\ & & & & r_{65} & \to & s_{88} & \to & s_{44} & & & & & & r_7 & \cdots & \\ & & & & & & & \searrow & & & \nearrow & & & & & \\ & & & & & & & & s_{69} & \to & r_{13} & & & & & \end{array} \tag{2.10}$$

È chiaro quindi che con l'algoritmo indicato in (2.10) è possibile analizzare anche sequenze del tipo di quelle in tabella 1 e quindi permettere alla grammatica di essere ambigua e di rilevare quest'ambiguità.

Per la strutturazione dei dati non cambia nulla: è sufficiente prevedere che ogni coppia (s_i, σ_j) possa essere associata a più di un'azione.

2.5. Lessico

Il lessico è forse la parte più importante del lavoro del filologo sui documenti d'archivio: sia perché l'evoluzione diacronica del linguaggio riguarda principalmente il lessico, sia perché è tramite il lessico che ci si riallaccia allo studio storico sui documenti. In questo senso, due sono le informazioni che è indispensabile registrare:[7]

- ◇ l'*etimologia* delle parole; (v. tabella 2 come esempio di analisi);
- ◇ le *note bibliografiche*, sia di attestazione dei termini, che di altre ricerche.

Visto che uno degli scopi è quello di generare un dizionario dei termini di un insieme di documenti (v. tabella 3), è opportuno registrare il significato (così come lo si trova nei normali vocabolari) e altre annotazioni a discrezione dell'utente (traslitterazioni, pronuncia, ecc.).

Oltre a questo, è necessario associare a ogni termine le informazioni utili per l'analisi sintattica, sotto forma di caratteristiche (transitivo, ditransitivo, collettivo, astratto, ecc.) e di tipo di attributi

7. Strettamente parlando, questo tipo di informazioni sarebbe necessario collezionarle anche per la grammatica. Ne parliamo solo qui per semplicità, ma è una cosa da tenere presente nei successivi sviluppi.

Tabella 2: Dati statistici sulla composizione del lessico del documento [DT, b. 9 n. 1099]. Le statistiche per funzione sono fatte seguendo l'uso che viene fatto del singolo termine nel documento. Le statistiche per etimologia seguono, per quanto possibile, la prima lingua di origine di ogni termine: per questo, le parole arabe mediate dal persiano sono contate comunque come arabe. In ogni cella, la prima riga rappresenta il numero di parole distinte, la seconda tiene conto anche delle ripetizioni.

	totale	*altro*	*arabo*	*cinese*	*persiano*	*arabo-persiano*	*mediterraneo*	*latino*	*turco*
grammaticali									
aggettivi pronominali	2				1 50.00%				1 50.00%
	24				3 12.50%				21 87.50%
congiunzioni	7		5 71.42%		2 28.57%				
	66		57 86.36%		9 13.63%				
numerali	9								9 100.00%
	15								15 100.00%
post e preposizioni	9		1 11.11%						8 88.88%
	20		1 5.00%						19 95.00%
pronomi	4								4 100.00%
	14								14 100.00%
verbi ausiliari	9								9 100.00%
	72								72 100.00%
tot. grammaticali	40		6 15.00%		3 7.50%				31 77.50%
	211		58 27.48%		12 5.68%				141 66.82%
lessico									
aggettivi	54		41 75.92%		12 22.22%				1 1.85%
	66		51 77.27%		14 21.21%				1 1.51%
aggettivi comparativi	1		1 100.00%						
	2		2 100.00%						
avverbi	18		9 50.00%		4 22.22%				5 27.77%
	23		10 43.47%		5 21.73%				8 34.78%
interiezioni	1		1 100.00%						
	2		2 100.00%						
nomi	156	2 1.28%	123 78.84%	1 0.64%	15 9.61%	1 0.64 %	4 2.56 %	2 1.28%	8 5.12%
	254	3 1.18%	196 77.16%	1 0.39%	29 11.41%	1 0.39 %	7 2.75 %	7 2.75%	10 3.93%
stati costrutti	1					1 100.00 %			
	1					1 100.00 %			
verbi	11				1 9.09%				10 90.90%
	15				1 6.66%				14 93.33%
tot. lessico	242	2 0.82%	175 72.31%	1 0.41%	32 13.22%	2 0.82 %	4 1.65 %	2 0.82%	24 9.91%
	363	3 0.82%	261 71.90%	1 0.27%	49 13.49%	2 0.55 %	7 1.92 %	7 1.92%	33 9.09%
altro	1		1 100.00%						
	1		1 100.00%						
nomi propri	9								
	17								
totale	292	2 0.70%	182 64.31%	1 0.35%	35 12.36%	2 0.70 %	4 1.41 %	2 0.70%	55 19.43%
	592	3 0.52%	320 55.65%	1 0.17%	61 10.60%	2 0.34 %	7 1.21 %	7 1.21%	174 30.26%

Tabella 3: Esempio di voci del dizionario generato dal database lessicale del documento [DT, b. 9 n. 1099], dove vengono registrate informazioni morfologiche, traslitterazioni, etimologie, significato e note bibliografiche divise in pronuncia e riferimenti generali. Abbreviazioni: n.=nome, a.=arabo.

إستِفسَار, *istifsār*, istifsar, n. a.: inchiesta, azione di fare domande; informazione.
Pron.: [Redhouse, 1997], [Meninski, 1680da, p. 192]
Rif.: [Kieffer, Bianchi, 1835a, p. 35]
إظهَار, *iẓhār*, äzher, n. a.: manifestazione, esposizione, testimonianza.
Pron.: [Redhouse, 1997], [Meninski, 1680da, p. 275]
Rif.: [Kieffer, Bianchi, 1835a, p. 56]

richiesti.

Infine, dal punto di vista più strettamente semantico, i temini andrebbero categorizzati. Qui si possono usare vari modelli, che possono associare caratteristiche a un certo termine oppure includere il termine in un insieme [Del Monte, 2008, p. 28-33].

Si possono identificare due categorie di oggetti per ogni termine: uno che riguarda la *forma* dell'oggetto (etimologia e altre caratteristiche decise dall'utente) e uno che riguarda il *significato* (caratteristiche sintattiche, categorizzazione, ecc.).

In entrambi i casi, i dati possono essere strutturati in più modi, per esempio:

- tramite descrizioni (in modo analogo a quanto fatto nella parte morfologica);
- tramite associazioni dirette tra due termini (nel caso di un'antinomia per esempio);
- tramite inclusione di termini in insiemi di termini collegati (sul modello di FrameNet).

L'organizzazione di questi dati dipende molto dal tipo di ricerca: è quindi opportuno dare all'utente più metodi di lavoro possibili. È necessario però prevedere delle tabelle derivate dove le diverse classificazioni vengano tradotte in una forma consona all'analisi sintattica (v. sezione 2.4).

3. Tabelle di base

3.1. Parsing preliminare

Come abbiamo visto in sezione 2, è necessario definire come suddividere il testo in modo che ogni token ottenuto rappresenti una parola (nel senso comune del termine). Il metodo scelto prevede l'uso di espressioni regolari, ognuna delle quali individua un tipo di token, come scelto dall'utente.

In figura 3 sono rappresentate le tabelle necessarie. Una lingua è definita da un insieme di espressioni regolari. Di queste, una viene scelta per definire i delimitatori di periodo: è necessario imporre un vincolo, in modo che il delimitatore di periodo sia tra i token previsti per la lingua.[8] Il delimitatore

8. Il modo in cui imporre questo vincolo dipende dal motore di database e dall'applicazione. In Django è sufficiente definire un controllo aggiuntivo in fase di salvataggio dei dati. Volendo, con PostgreSQL si può definire un trigger, ossia

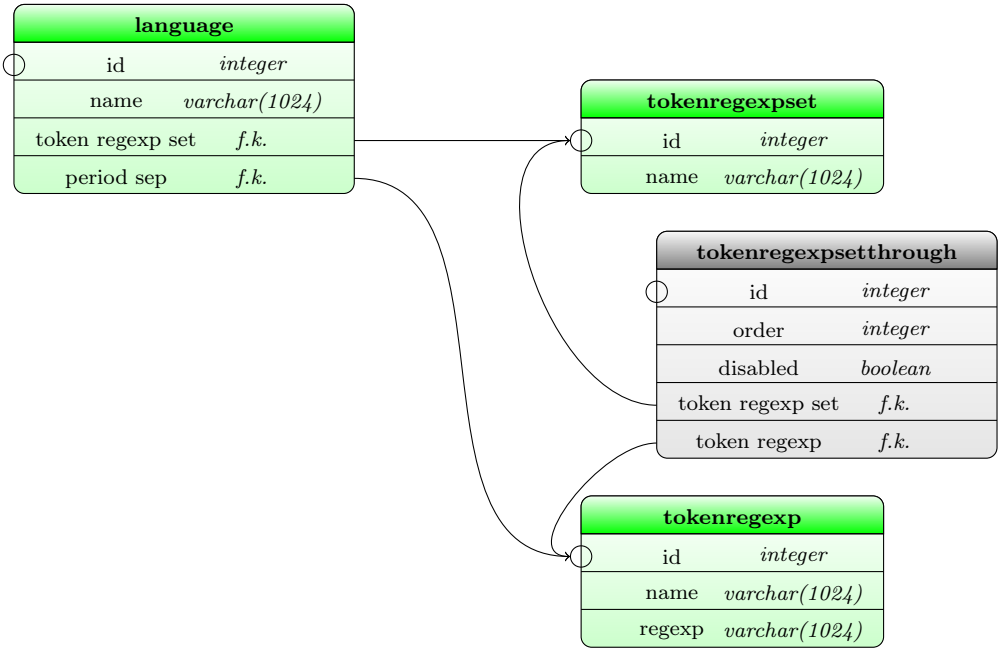


Figura 3: Tabelle di base: definizione di lingua e espressioni regolari per il parsing iniziale. Le frecce indicano una foreign key (f.k.). In grigio le tabelle di relazione per le relazioni molti a molti.

di periodo individua quei token che fanno fermare l'analizzatore sintattico (una funzione equivalente a quella del simbolo $ in figura 2).

In tabella 4 è riportato un esempio di come possono essere definite delle regole di parsing per un testo in italiano corrente. Le espressioni regolari sono un formalismo molto potente, che consentono di suddividere un brano con un'unica operazione e una precisione definibile a piacere. Nella versione più semplice, l'applicazione concatena tutte le espressioni regolari definite dall'utente, aggiunge quelle per i tag di formattazione e utilizza l'espressione risultante per suddividere il testo. Quindi confronta i token ottenuti con l'elenco delle espressioni regolari per etichettarli.[9]

Quindi per esempio, definite le espressioni regolari:

alpha	`[a-z]+`
space	`[ ]`
numbers	`[0-9]+`

(3.1)

l'espressione regolare risultante diventa:

```
([a-z]+|[ ]|[0-9]+)
```

(3.2)

e utilizzando una funzione di split per espressioni regolari, la frase *oggi ho letto 3 libri* viene suddivisa in

un controllo che viene eseguito al momento dell'inserimento dei dati. Altri motori possono usare altri meccanismi.

9. Il metodo reale di parsing e la descrizione dell'espressione regolare dipendono dal linguaggio di programmazione scelto. Qui e in seguito si userà quanto stabilito dalla libreria *re* di Python [Python:re; Kuchling, 2012]. Per una trattazione più generale si veda [Hopcroft, Ullman, 1979].

Tabella 4: Esempio di espressioni regolari per il parsing dell'italiano, in grassetto quella possibile come separatore di periodi. Nella tabella in basso l'output risultante con un brano di esempio. Le notazioni `(?=`x`)`, `(?!`x`)`, `(?<=`x`)` e `(?<!`x`)` sono estensioni di Python (individuano il testo adiacente, rispettivamente come seguito, non seguito, preceduto, non preceduto da x).

space	`[ ]`	*spazio da solo*
dialog mark	`["«»‘“”]`	
hyphen	`[--]`	*tipi diversi di trattino*
ellipses	`...|[.]{2,}`	*tre puntini (unico carattere) o più di due punti consecutivi*
numbers	`[0-9]+`	
brackets	`[(){}\[\]]`	
apostrophe alone	`(?<![a-zA-ZàèìòùáéíóúÀÈÌÒÙÁÉÍÓÚ])['’]`	*apostrofo non preceduto da altri caratteri*
alpha	`[a-zA-ZàèìòùáéíóúÀÈÌÒÙÁÉÍÓÚ]+['’]?`	*caratteri alfabetici eventualmente seguiti da apostrofo*
others	`[^(){}\[\]a-zA-ZàèìòùáéíóúÀÈÌÒÙÁÉÍÓÚ ?,;:!....'’\t\n\r0-9"«»‘“”’--]`	*caratteri che non sono quelli dell'elenco*
tab	`[\t]`	
new line	`[\n\r]`	
period	`(?<![.])[.](?![.])|[?;:!]`	*punto isolato o altri segni di interpunzione forte*
punctuation mark	`[,]`	

Un *alpha*	 *space*	raggio *alpha*	 *space*	di *alpha*	 *space*	sole *alpha*	 *space*	filtrò *alpha*	 *space*
con *alpha*	 *space*	decisione *alpha*	 *space*	dalla *alpha*	 *space*	finestra *alpha*	, *p. mark*	 *space*	invase *alpha*
 space	la *alpha*	 *space*	camera *alpha*	 *space*	783 *numbers*	 *space*	dell' *alpha*	hotel *alpha*	 *space*
e *alpha*	 *space*	si *alpha*	 *space*	andò *alpha*	 *space*	a *alpha*	 *space*	schiantare *alpha*	, *p. mark*
 space	dritto *alpha*	 *space*	dritto *alpha*	, *p. mark*	 *space*	sul *alpha*	 *space*	suo *alpha*	 *space*
occhio *alpha*	 *space*	sinistro *alpha*	. *period*	 *space*	“ *d. mark*	Oh *alpha*	 *space*	no *alpha*	, *p. mark*
” *d. mark*	 *space*	pensò *alpha*	 *space*	Roberto *alpha*	 *space*	deciso *alpha*	 *space*	a *alpha*	 *space*
non *alpha*	 *space*	arrendersi *alpha*	, *p. mark*	 *space*	“ *d. mark*	sono *alpha*	 *space*	stanco *alpha*	, *p. mark*
 space	voglio *alpha*	 *space*	dormire *alpha*	 *space*	ancora *alpha*	. *period*	” *d. mark*		

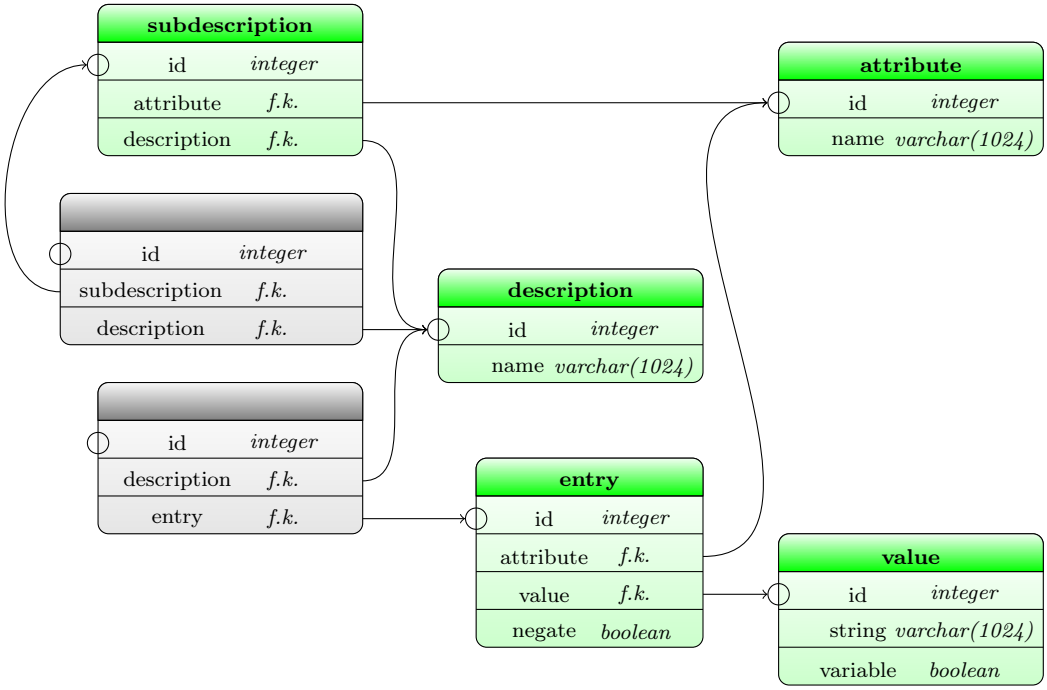


Figura 4: Tabelle di base: descrizioni. Le frecce indicano una foreign key (f.k.). In grigio le tabelle di relazione per le relazioni molti a molti. Una descrizione è una collezione di coppie attributo/valore o attributo/descrizione: quindi viene rappresentata come una tabella (*description*) con due relazioni molti a molti, rispettivamente con una tabella che unisce attributi e valori (*entry*) e una tabella che unisce attributi e descrizioni (*subdescription*). I valori (tabella *value*) possono essere costanti (campo *variable* posto a 'false') o variabili (campo *variable* posto a 'true'). All'interno di una descrizione possono apparire anche come negati (campo *negate* della tabella *entry*).

```
[ 'oggi' , ' ' , 'ho' , ' ' , 'letto' , ' ' , '3' , ' ' , 'libri' ].
```
(3.3)

A questo punto un successivo confronto indivduerà '3' come un token di tipo *numbers*, 'oggi', 'ho', 'letto' e 'libri' come token di tipo *alpha* e gli altri come token di tipo *space*.

Da notare che con le espressioni regolari definite in (3.1), la frase *oggi, come di consuento, ho letto 3 libri*, produce un token aggiuntivo, ',', che verrà marcato come *sconosciuto* in quanto non previsto da nessuna regola.[10] L'utente dovrà quindi inserire una nuova espressione regolare, per esempio:

```
punctuation    [.:;,?!]
```
(3.4)

che consente al programma di individuare il tipo giusto per il token ','.

3.2. Descrizioni

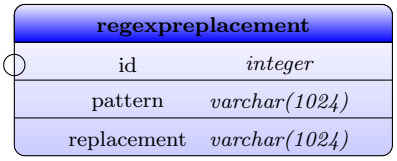
La rappresentazione di una descrizione (v. sezione 2.2) nel database è riportata in figura 4. Una

10. La funzione di split, qui considerata come esempio di metodo di parsing, suddivide una stringa basandosi su un'espressione regolare data come delimitatore. Ossia, data l'espressione regolare `a+`, suddivide la stringa *defaabaghj* in `['def','b','ghj']`. Se si usano le parentesi, ossia `(a+)`, ritorna anche le occorrenze della stringa, ossia: `['def','aa','b','a','ghj']`, che è il risultato che ci interessa [Python:re].

descrizione è una collezione di coppie attributo/valore o attributo/descrizione: quindi viene rappresentata come una tabella (*description*) con due relazioni molti a molti, rispettivamente con una tabella che unisce attributi e valori (*entry*) e una tabella che unisce attributi e descrizioni (*subdescription*). I valori (tabella *value*) possono essere costanti (campo *variable* posto a 'false') o variabili (campo *variable* posto a 'true'). All'interno di una descrizione possono apparire anche come negati (campo *negate* della tabella *entry*).

4. Tabelle morfologiche

Anche in questo caso, come meccanismo base abbiamo usato le espressioni regolari, elencate nella tabella delle sostituzioni (v. figura 5). Ogni sostituzione è definita da due campi, un'espressione regolare (*pattern*) e la stringa che definisce la sostituzione (*replacement*).



pattern	*replacement*			
`(.*)`	`\1iamo`	(am)	(am)iamo	amiamo
`(.*)([bcdfglmnpqrstvz])`	`\1\2\2ia`	(sa)(p)	(sa)(p)(p)ia	sappia

Figura 5: Morfologia: sostituzioni basate su espressioni regolari. A lato degli esempi.

Una radice tematica è l'unione di una radice lessicale e di una regola di derivazione, mentre una parola è l'unione di una radice tematica con una regola di flessione (per la composizione vedi sezione 4.3). Per rappresentarle sono sufficienti tre tabelle, quella delle radici lessicali (*root*), quella delle regole di derivazione (*derivation*) e quella delle regole di flessione (*inflection*): le tabelle delle radici tematiche (*stem*) e delle parole (*word*) possono essere calcolate in base alle caratteristiche delle altre tre. Per questioni di prestazione, è prevista anche una tabella di cache (*wordcache*), che è quella usata dalla componente di analisi dei testi.

Una tabella a parte è quella delle non-parole (*notword*), ossia un elenco di token che non devono essere analizzati morfologicamente. Questa tabella comprende spazi, tabulazioni, a capo, punteggiatura, ecc. (v. figura 7).

4.1. Derivazione

Perché una regola di derivazione d e una radice lessicale r possano dar vita a una radice tematica s devono essere compatibili, ossia dev'essere possibile applicare d a r. Le caratteristiche che definiscono la compatibilità sono state divise in tre parti: la categoria (verbale, nominale, ecc.), il tema e la descrizione della radice.

La categoria di r, $r.pos$, è definita dal campo *part of speech* e dev'essere uguale al campo *root part of speech* di d, $d.rpos$:

$$r.pos = d.rpos\,. \tag{4.1}$$

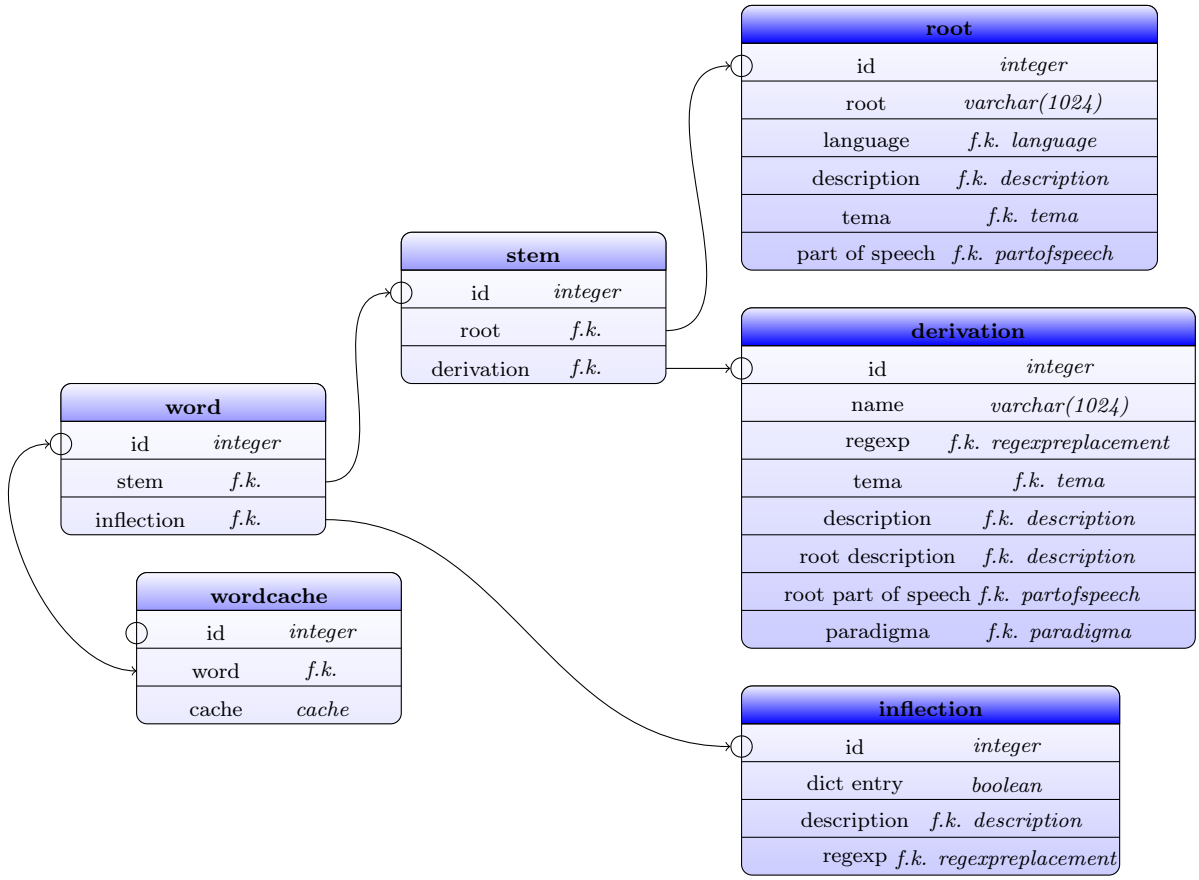


Figura 6: Morfologia: relazioni tra radici lessicali (*root*), radici tematiche (*stem*) e parole (*word*). In colore più chiaro le tabelle calcolate.

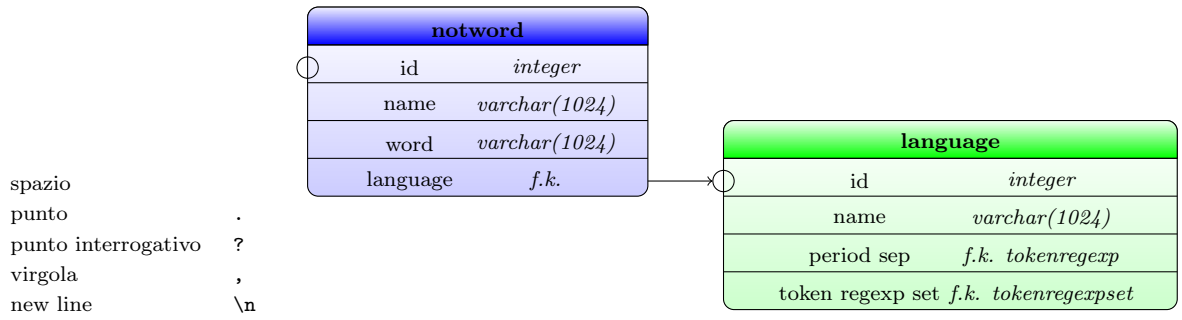


Figura 7: Morfologia: non-parole. A lato degli esempi.

Entrambi sono un collegamento alla tabella *partofspeech*, dove l'utente definisce le parti del discorso che gli interessano.

Il tema è una versione semplificata di descrizione, dove gli attributi possono essere associati solo a valori costanti e rappresentano tipi di derivazioni possibili (v. figura 8). Perché il tema di r, $r.tema$, sia compatibile con quello di d, $d.tema$, bisogna che:

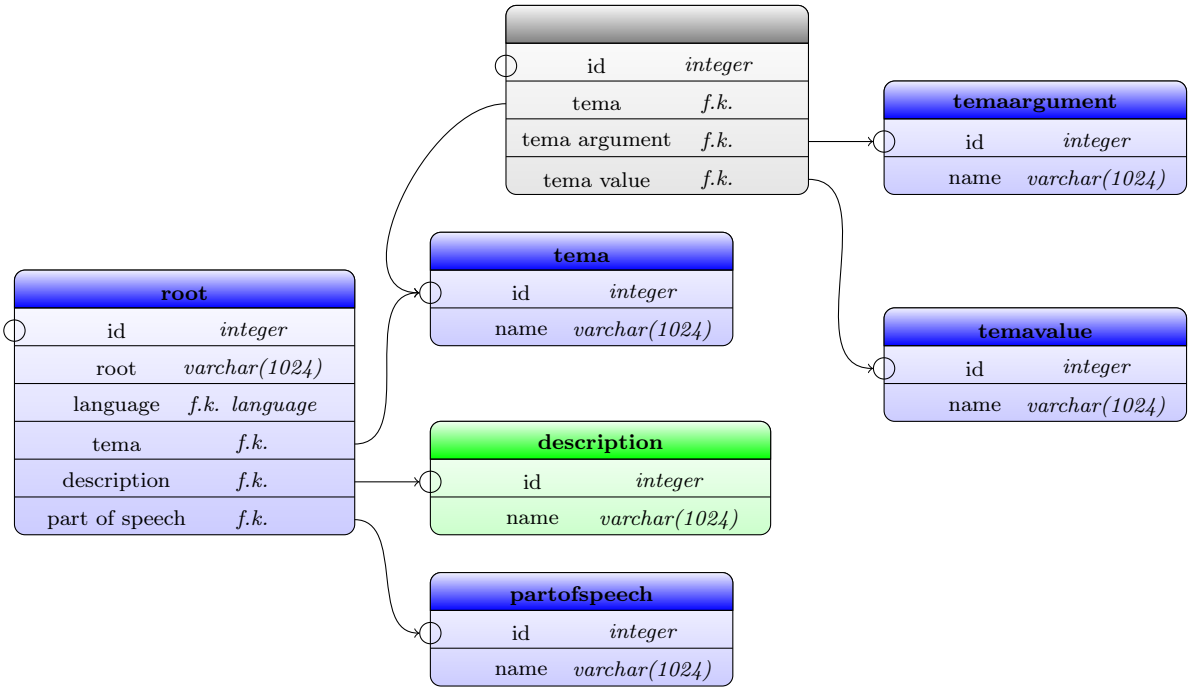


Figura 8: Morfologia: radici.

$$d.tema \sqsubseteq r.tema\,. \tag{4.2}$$

Ad esempio, il tema della radice lessicale 'bell⊢' può essere descritto come:

$$\begin{bmatrix} \text{base} & = & \text{-o} \\ \text{nome di qualità} & = & \text{-ezza} \end{bmatrix} \tag{4.3}$$

mentre quello di 'amic⊢':

$$\begin{bmatrix} \text{base} & = & \text{-o} \\ \text{nome di qualità} & = & \text{-izia} \end{bmatrix} \tag{4.4}$$

La regola di derivazione che ricava le radici tematiche 'bell-' e 'amic-', avrà come tema:

$$\begin{bmatrix} \text{base} & = & \text{-o} \end{bmatrix} \tag{4.5}$$

mentre quella che ricava 'bellezz-' avrà:

$$\begin{bmatrix} \text{nome di qualità} & = & \text{-ezza} \end{bmatrix} \tag{4.6}$$

e quella che ricava 'amicizi-':

$$\begin{bmatrix} \text{nome di qualità} & = & \text{-izia} \end{bmatrix} \tag{4.7}$$

Infine è necessario che la descrizione di r, $r.desc$, sussuma quella richiesta per la radice da d, $d.rdesc$ (campo *root description*):

$$d.rdesc \sqsubseteq r.desc\,. \tag{4.8}$$

Ad esempio, se la radice lessicale 'poet⊢' ha i seguenti parametri:

$$\left\{ \begin{array}{lcl} \text{categoria} & = & \text{nome} \\ \text{tema} & = & \left[\begin{array}{lcl} \text{base} & = & \text{-a} \\ \text{femminile} & = & \text{-essa} \end{array} \right] \\ \text{descrizione} & = & \left[\begin{array}{lcl} \text{genere} & = & \text{maschile} \end{array} \right] \end{array} \right\} \tag{4.9}$$

è compatibile con i seguenti parametri di una derivazione che genera nomi maschili in -a:

$$\left\{ \begin{array}{lcl} \text{categoria della radice} & = & \text{nome} \\ \text{tema} & = & \left[\begin{array}{lcl} \text{base} & = & \text{-a} \end{array} \right] \\ \text{descrizione della radice} & = & \left[\begin{array}{lcl} \text{genere} & = & \text{maschile} \end{array} \right] \end{array} \right\} \tag{4.10}$$

o con i seguenti parametri di una derivazione che genera nomi femminili in -essa:

$$\left\{ \begin{array}{lcl} \text{categoria della radice} & = & \text{nome} \\ \text{tema} & = & \left[\begin{array}{lcl} \text{femminile} & = & \text{-essa} \end{array} \right] \\ \text{descrizione della radice} & = & [] \end{array} \right\} \tag{4.11}$$

Oltre ai parametri necessari a selezionare le radici, una regola di derivazione contiene anche:

- una sostituzione basata su un'espressione regolare $d.regsub$ (*regsub*) che indica come modificare la radice lessicale per ottenere quella tematica (v. p. 16 e figura 5);
- una descrizione $d.desc$ (*description*);
- un paradigma $d.paradigma$ (*paradigma*), che contiene le regole di flessione che si applicano alla radice tematica risultante (v. sezione 4.2).

La componente che genera la tabella delle radici tematiche (*stem*) fa un prodotto cartesiano tra le radici e le derivazioni, registrando solo quelle compatibili. Una radice tematica è rappresentata solo da un collegamento a una radice lessicale e a una regola di derivazione. Le sue proprietà discendono da quella della derivazione, in particolare:

- la descrizione è l'unificazione tra quella della radice e quella della derivazione;
- il paradigma è quello della derivazione;
- la parte del discorso è quella del paradigma;
- la forma è il risultato dell'applicazione della sostituzione della derivazione (campo *regsub* della tabella *derivation*) alla radice lessicale (campo *root* della tabella *root*).

Riassumendo, date la radice lessicale r:

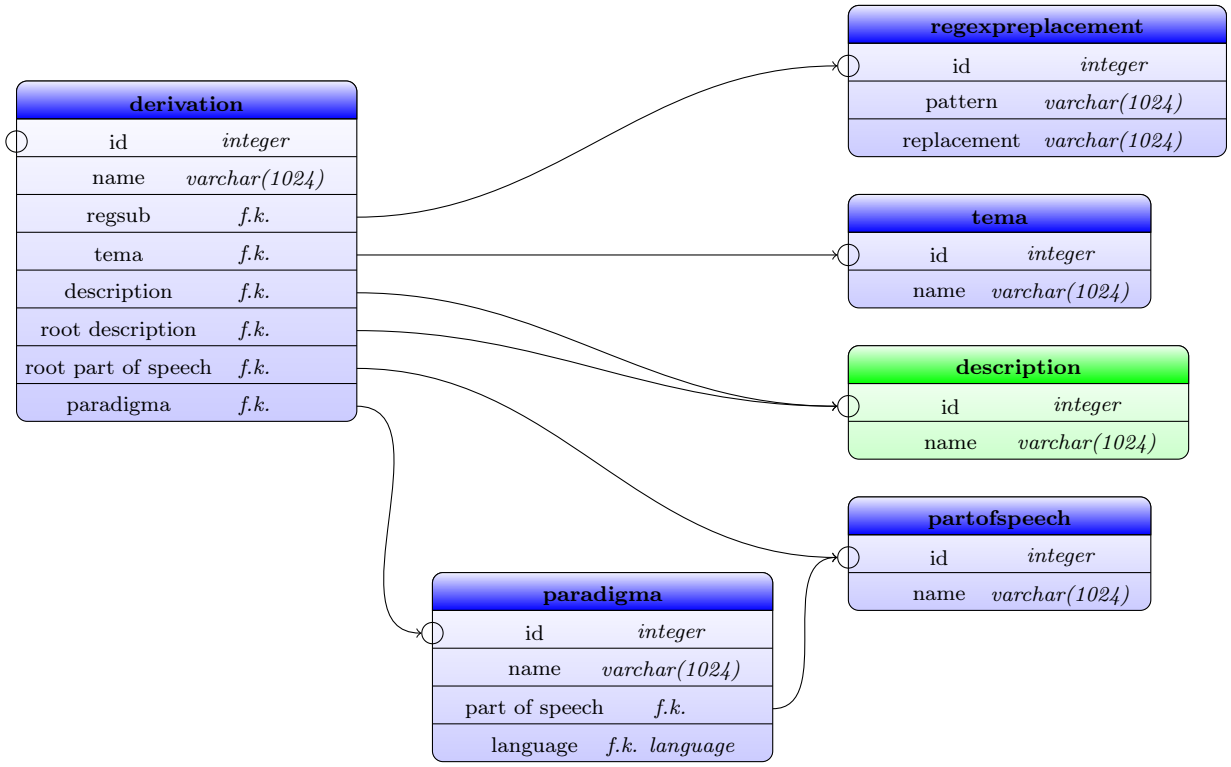


Figura 9: Morfologia: derivazione.

$$r = \begin{cases} r.root & \text{(radice)} \\ r.pos & \text{(categoria)} \\ r.tema & \text{(tema)} \\ r.desc & \text{(descrizione)} \end{cases} \tag{4.12}$$

e la regola di derivazione d:

$$d = \begin{cases} d.rpos & \text{(categoria della radice)} \\ d.tema & \text{(tema)} \\ d.rdesc & \text{(descrizione della radice)} \\ d.desc & \text{(descrizione)} \\ d.regsub & \text{(sostituzione)} \\ d.paradigma & \text{(paradigma)} \end{cases}, \tag{4.13}$$

d è applicabile a r se e solo se:

$$\begin{aligned} d.rpos &= r.pos \\ d.tema &\sqsubseteq r.tema \\ d.rdesc &\sqsubseteq r.desc \end{aligned} \tag{4.14}$$

e il risultato è:

$$r \overset{d}{\twoheadrightarrow} s = \begin{cases} s.stem & = & d.regsub \star r.root \\ s.tema & = & r.tema \\ s.desc & = & r.desc \cup d.desc \\ s.paradigma & = & d.paradigma \\ s.pos & = & d.paradigma.pos \end{cases} \tag{4.15}$$

4.2. Flessione

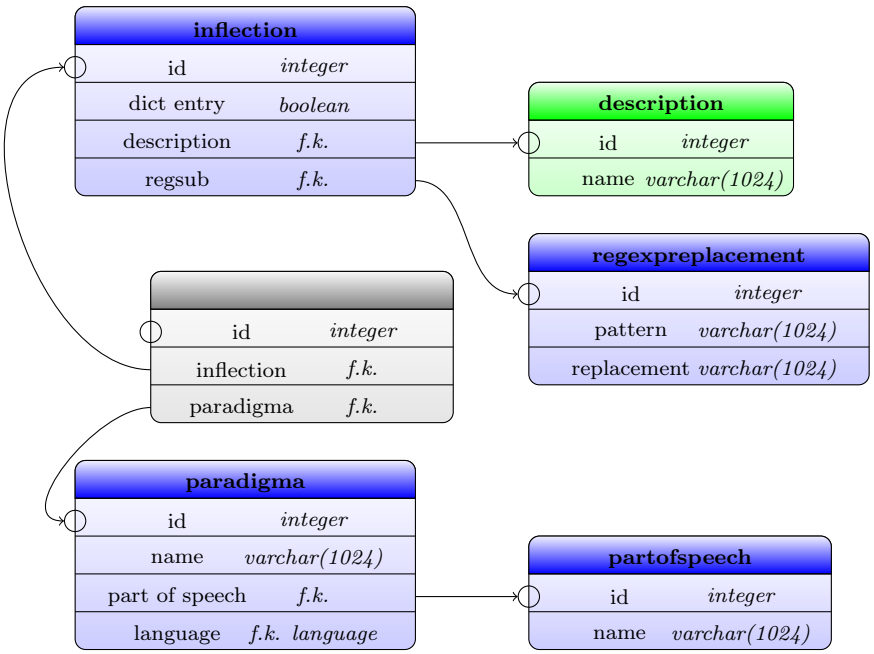


Figura 10: Morfologia: flessione.

Una volta associato, tramite una derivazione, un paradigma a una radice tematica, per ottenere le parole è sufficiente applicare tutte le regole di flessione contenute nel paradigma (v. figura 10), ossia, registrare nella tabella *word* i riferimenti alle radici tematiche e alle regole di flessione.

Una regola di flessione i contiene:

- una sostituzione basata su un'espressione regolare $i.regsub$ (*regsub*) che indica come modificare la radice tematica per ottenere una parola (v. p. 16 e figura 5);
- una descrizione $i.desc$ (*description*);
- un campo booleano $i.vocediz$ (*dict entry*), che indica se la parola risultante è una voce del dizionario per la sua radice tematica.

Le proprietà di una parola discendono sia da quelle della radice tematica, che da quelle della regola di flessione, in particolare:

- la descrizione è un'unione delle due descrizioni;
- la parte del discorso è quella della radice tematica;

- ◇ la forma è il risultato dell'applicazione della sostituzione della flessione (campo *regsub* della tabella *inflection*) alla radice tematica.

Quindi, date la radice tematica s:

$$s = \begin{cases} s.stem & \text{(radice)} \\ s.desc & \text{(descrizione)} \\ s.pos & \text{(parte del discorso)} \\ s.paradigma & \text{(paradigma)} \\ s.tema & \text{(tema)} \end{cases} \tag{4.16}$$

e la regola di flessione i:

$$i = \begin{cases} i.desc & \text{(descrizione)} \\ i.regsub & \text{(sostituzione)} \\ i.vocediz & \text{(è voce del dizionario?)} \end{cases}, \tag{4.17}$$

i è applicabile a s se e solo se:

$$i \in s.paradigma \tag{4.18}$$

e il risultato è:

$$s \xrightarrow{i} w = \begin{cases} w.word & = & i.regsub \star s.stem \\ w.desc & = & s.desc \cup i.desc \\ w.vocediz & = & s.vocediz \\ w.tema & = & s.tema \\ w.pos & = & s.pos \end{cases} \tag{4.19}$$

La componente che aggiorna le parole si preoccupa anche di precalcolarle (tramite derivazione e flessione) e di inserirle nella tabella di cache (*wordcache*), assieme a un riferimento uno a uno verso la tabella delle parole (*word*).

4.3. Composizione

La composizione è una situazione particolare: ad ogni parola composta, che appare come un singolo token nell'analisi morfologica, corrispondono più parole, che vengono passate al livello successivo come fossero parole distinte. Ossia, mentre la parola 'casa' al termine dell'analisi morfologica darà il singolo token 'casa', la parola composta 'del' produrrà due token, 'di' e 'il'.

Una parola composta c si ottiene applicando una composizione f a una sequenza parole $(w_1, .., w_n)$, specificate in un certo ordine (v. figura 11):

$$(w_1, \cdots, w_N) \overset{f}{\rightrightarrows} c. \tag{4.20}$$

Una composizione è formata da più regole di composizione f_n, ognuna delle quali si applica alla corrispondente parola w_n (v. figura 12):

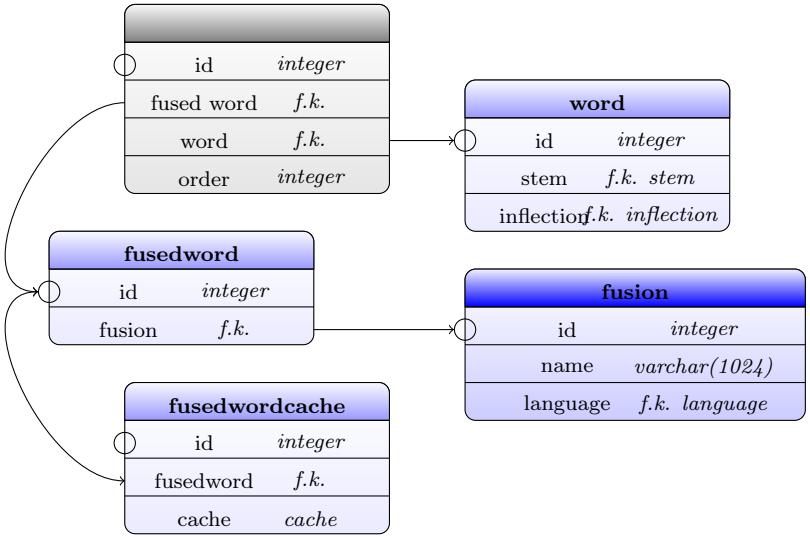


Figura 11: Morfologia: relazione tra parole (*word*) e parole composte (*fusedword*). In colore più chiaro le tabelle calcolate e in grigio le tabelle di relazione per relazioni molti a molti (in questo caso calcolate).

$$\begin{pmatrix} w_1 & \overset{f_1}{\rightrightarrows} & c_1 \\ w_2 & \overset{f_2}{\rightrightarrows} & c_2 \\ \vdots & \vdots & \vdots \\ w_N & \overset{f_N}{\rightrightarrows} & c_N \end{pmatrix}. \tag{4.21}$$

La parola composta è ottenuta poi per concatenazione:

$$c = \sum_{n=1}^{N} c_n. \tag{4.22}$$

La compatibilità tra una parola w_n e una regola f_n è del tutto analoga a quella tra una radice lessicale e una derivazione.

Quindi, data una lista di parole $(w_1, \cdots, w_N)$, in cui ogni w_n è:

$$w_n = \begin{cases} w_n.word & \text{(parola)} \\ w_n.desc & \text{(descrizione)} \\ w_n.pos & \text{(parte del discorso)} \\ w_n.vocediz & \text{(è voce del dizionario?)} \\ w_n.tema & \text{(tema)} \end{cases} \tag{4.23}$$

e una composizione $f = (f_1, \cdots, f_M)$, in cui ogni f_n è:

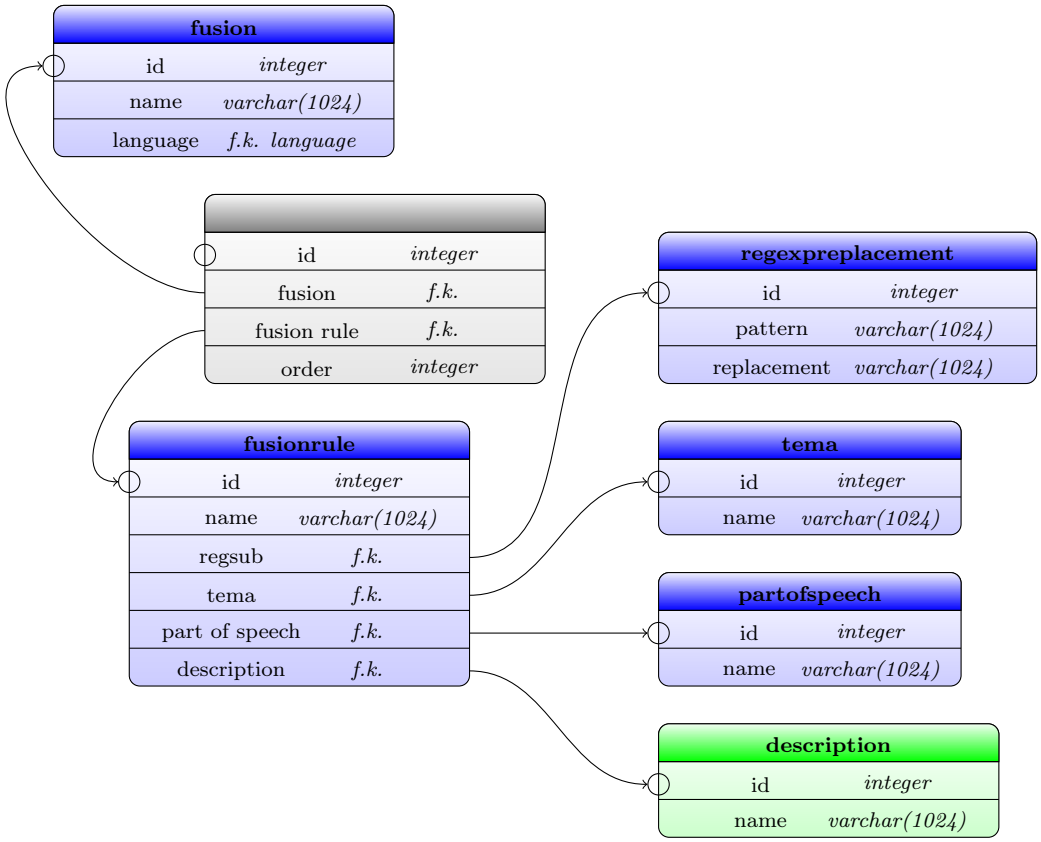


Figura 12: Morfologia: composizione. In grigio le tabelle di relazione per relazioni molti a molti.

$$f_n = \begin{cases} f_n.regsub & \text{(sostituzione)} \\ f_n.desc & \text{(descrizione)} \\ f_n.tema & \text{(tema)} \\ f_n.pos & \text{(parte del discorso)} \end{cases}, \tag{4.24}$$

f è applicabile a $(w_1, \cdots, w_N)$ se e solo se $N = M$ e per ogni n:

$$\begin{aligned} f_n.pos &= w_n.pos \\ f_n.tema &\sqsubseteq w_n.tema \\ f_n.desc &\sqsubseteq w_n.desc \end{aligned} \tag{4.25}$$

e il risultato è:

$$(w_1, \cdots, w_N) \overset{f}{\rightrightarrows} c = \begin{cases} c.comp &= \sum_{n=1}^{N} f_n.regsub \star w_n.word \\ c.words &= (w_1, \cdots, w_N) \\ c.desc &= (w_1.desc, \cdots, w_N.desc) \\ c.pos &= (w_1.pos, \cdots, w_N.pos) \end{cases} \tag{4.26}$$

4.4. Il parsing morfologico

Il parsing, di cui un esempio di output è in tabella 5, viene eseguito ricercando ogni token prodotto dal parsing preliminare (v. sezione 3.1) nelle tabelle:

1. delle non-parole (*notword*);
2. delle parole (*wordcache*);
3. delle parole composte (*fusedwordcache*).

A parte nel caso delle non-parole, in cui esiste una sola tabella, e ogni oggetto produce un solo token, l'analisi viene eseguita risalendo le relazioni che partono dagli oggetti di cache (v. figura 6 e figura 11), in modo da ricostruire i parametri delle parole da analizzare come visto nelle sezioni precedenti.

Per ogni token p, può succedere che:

1. p non sia trovato in nessuna tabella: verrà segnalato all'utente la mancanza di regole per quella parola;
2. p corrisponda a un solo record della tabella *wordcache* (il caso di 'panchina' in tabella 5): produrrà un token;
3. p corrisponda a più record, tutti dalla tabella *wordcache* (il caso di 'la' in tabella 5): produrrà un solo token, ma ci saranno due possibili risultati di analisi (segnalando quindi un'ambiguità che dovrà essere risolta nei passaggi successivi);
4. p corrisponda a un solo record della tabella *fusedwordcache* (il caso di 'della' in tabella 5): produrrà più token, ma un solo risultato;
5. p corrisponda a più record, almeno uno dei quali della tabella *fusedwordcache* (il caso di 'dell'' in tabella 5): produrrà più token e più risultati.

Tabella 5: Esempio di analisi morfologica della frase *la panchina della fermata dell'autobus*. Gli articoli determinativi sono rappresentati come forme flesse della radice 'il-', con regole di sostituzione banali (ad esempio, per il femminile 'la' la regola è: pattern=`(.*)`, replacement=`la`). Da notare che non interpreta correttamente 'fermata': questo perché nella grammatica definita esiste la derivazione che genera il verbo 'ferm-' a partire dalla radice lessicale 'ferm⊢', ma non quella che genera il nome 'fermat-' dalla stessa radice.

parola	*parte del discorso*		*radice tematica*	*descrizione*		
la	articolo		il-	definitezza	=	definito
				numero	=	singolare
				genere	=	femminile
	pronome		lei-	categoria	=	personale
				persona	=	III
				genere	=	femminile
				numero	=	singolare
				caso	=	accusativo (debole)
	non parola		spazio			
panchina	nome		panchin-	genere	=	femminile
				numero	=	singolare
	non parola		spazio			
della	composta	preposizione di	di-			
		articolo la	il-	definitezza	=	definito
				numero	=	singolare
				genere	=	femminile
	non parola		spazio			
fermata	verbo		ferm-	genere	=	femminile
				numero	=	singolare
				deverbale	=	aggettivo verbale
				tempo	=	passato
				modo	=	participio
	non parola		spazio			
dell'	composta	preposizione di	di-			
		articolo l'	il-	definitezza	=	definito
				numero	=	singolare
				genere	=	maschile
	composta	preposizione di	di-			
		articolo l'	il-	definitezza	=	definito
				numero	=	singolare
				genere	=	femminile
	non parola		spazio			
autobus	nome		autobus-	genere	=	maschile
				numero	=	singolare
	nome		autobus-	genere	=	maschile
				numero	=	plurale

Note e convenzioni

Simboli:

- $\sqsupseteq$ sussume;
- $\sqsubseteq$ è sussunto da;
- $\cup$ unificazione;
- $\rightarrow$ una produzione di una grammatica BNF;
- $\rightarrowtriangle$ una flessione (es. un plurale da un singolare);
- $\twoheadrightarrow$ una derivazione (es. un verbo da un nome);
- $\rightrightarrows$ una composizione (es. una preposizione articolata);
- $s \star t$ indica l'applicazione della sostituzione s (definita come coppia espressione regolare/sostituzione) alla stringa t;
- $\sum$ nel caso di stringhe indica concatenazione;
- $x\vdash$ indica che x è una radice lessicale;
- $x\text{-}$ indica che x è una radice tematica;

Bibliografia

[DT, b. 9 n. 1099] Archivio di Stato di Venezia, *Documenti turchi*, b. 9 n. 1099, "Caʿfer Paşa ex-Beylerbeyi di Cipro al Doge di Venezia", Costantinopoli, sd (gennaio 1601).

[Aho, Sethi, Ullman, 1986] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, *Compilers. Principles, Techniques, and Tools*, Boston, Addison-Wesley, 1986.

[Albano, Ghelli, Orsini, 1997] Antonio Albano, Giorgio Ghelli, Renzo Orsini, *Basi di dati relazionali e a oggetti*, Bologna, Zanichelli, 1997.

[Cecchetto, 2002] Carlo Cecchetto, *Introduzione alla sintassi. La teoria dei principi e dei parametri*, Milano, LED - Edizioni Universitarie di Lettere Economia Diritto, 2002.

[Chomsky, 1965] Noam Chomsky, *Aspects of the Theory of Syntax*, Cambridge, Mass., MIT Press, 1965.

[Chomsky, 2002] Noam Chomsky, *Syntactic Structures*, Berlin, Mouton de Gruyer, 2002, (II ed.).

[Del Monte, 2008] Rodolfo Del Monte, *Computational Linguistic Text Processing. Lexicon, Grammar, Parsing and Anaphora Resolution*, New York, Nova Science Publishers, 2008.

[Django] *Django*, Django Software Foundation, 2005-2012.
https://www.djangoproject.com/ (ultima visita: 25 agosto 2012)

[Fenstad, Langholm, Vestre, 1992] Jens Eric Fenstad, Tore Langholm, Espesn Vestre, "Representations and Interpretations", in *Computational linguistics and formal semantics*, a cura di Michael Rosner, Roderick Johnson, Cambridge, UK, Cambridge University Press, 1992, pp. 31-96.

[Hopcroft, Ullman, 1979] John E. Hopcroft, Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Boston, Addison-Wesley, 1979.

[Kay, 1992] Martin Kay, "Unification", in *Computational linguistics and formal semantics*, a cura di Michael Rosner, Roderick Johnson, Cambridge, UK, Cambridge University Press, 1992, pp. 1-30.

[Kieffer, Bianchi, 1835a] J. D. Kieffer, T. X. Bianchi, *Dictionnaire Turc-Français à l'usage des agents diplomatiques et consulaires, des commerçants, des navigatuers et autres voyageurs dans le levant. Tome premier*, Paris, Imprimerie Royale, 1835.

[Kuchling, 2012] A.M. Kuchling, *Regular Expression HOWTO*, Python Software Foundation, 2012.
http://docs.python.org/howto/regex.html (ultima visita: 26 agosto 2012)

[Longo, 2012] Andrea Longo, *Realizzazione di Definite Clause Grammar in Java tramite JSetL*, tesi di laurea, Università degli Studi di Parma, Facoltà di Scienze Matematiche, Fisiche e Naturali, Corso di Laurea in Informatica, a.a. 2010/2011, rel. Gianfranco Rossi, Parma, 2012.

[Meninski, 1680da] Francisci à Mesgnien Meninski, *Thesaurus Linguarum Orientalium Turcicæ, Ara-*

bicæ, Persicæ. Lexicon Turcico-Arabico-Persicum. I. أ-خ, Viennæ, sumptibus auctoris, 1680, ristampa anastatica, Simurg, İstanbul, 2000.

[PostgreSQL] *PostgreSQL*, The PostgreSQL Global Development Group, 1996-2012. http://www.postgresql.org (ultima visita: 25 agosto 2012)

[Python] *Python Programming Language*, Python Software Foundation, 1990-2012. http://www.python.org/ (ultima visita: 25 agosto 2012)

[Python:re] *The Python Standard Library. 7.2 re — Regular expression operations*, Python Software Foundation, 1990-2012. http://docs.python.org/library/re.html (ultima visita: 26 agosto 2012)

[Redhouse, 1997] *Redhouse Türkçe/Osmanlıca-İngilizce Sözlük*, a cura di V. B. Alkım, N. Antel, R. Avery, J. Eckmann, S. Hurı, F. İz, M. Mansuroğlu, A. Tietze, İstanbul, SEV Matbaacılık ve Yayıncılık A. Ş., 1997.

[Simone, 2008] Raffaele Simone, *Fondamenti di linguistica*, Roma-Bari, Editori Laterza, 2008.

[Tomita, 1985] Masaru Tomita, "An efficient context-free parsing algorithm for natural languages", in *Proceedings of the 9th international joint conference on Artificial intelligence - Volume 2*, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 1985, pp. 756-764.

[Tomita, 1987] Masaru Tomita, "An efficient augmented-context-free parsing algorithm", *Comput. Linguist.*, vol. 13, n. 1-2, jan, 1987, pp. 31-46.

Indice