UNIVERSITÁ DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Sviluppo di un dizionario matematico nella struttura di WordNet

Tesi di laurea di Stefano Croin

Relatori:

Chiar.mo prof. Carlo Minnaja

Chiar.ma prof. Laura Paccagnella

Anno Accademico 2002/2003

a tutte le persone importanti che ho incontrato nella mia vita

Indice

Introduzione

Gli antropologi hanno stabilito che l'essere umano realizza la sua vita nella relazione con gli altri. Con questo si intende dire che il relazionarsi è una necessità dettata dal proprio essere persone vive e costituisce per questo il fine ultimo dell'esistenza.

Questo fatto è evidenziato da numerosi studi sull'evoluzione umana ed è un bisogno da sempre manifestato attraverso il tentativo continuo di sviluppare una qualche forma di comunicazione con il prossimo. La persona realizza se stessa nel momento in cui riesce a stabilire uno scambio di conoscenze con un'altra persona. La storia insegna come la comunicazione sia stata protagonista di una vera e propria evoluzione caratterizzata da diversi stadi in cui essa avveniva con forme e modalità differenti (pensiamo alla comunicazione gestuale e, successivamente, a quella simbolica). Il punto di arrivo di questo percorso è la definizione e costruzione di un linguaggio costituito da un alfabeto e da regole che guidano la formazione di diversi costrutti semplici o composti quali parole e frasi. Questo linguaggio, accettato e riconosciuto da gruppi di persone, permette loro di comunicare.

La volontà di realizzare macchine che effettuino una qualche forma di

ragionamento divenendo più simili a noi (l'intelligenza artificiale è lo studio delle idee che permettono agli elaboratori elettronici di seguire quelle azioni che fanno apparire intelligenti le persone, P.H. Winston (1977)) porta a pensare di 'insegnare' loro uno strumento di comunicazione che rifletta il linguaggio naturale e cioè quanto permette la comunicazione fra gli esseri umani.

Uno dei passi sulla via del raggiungimento di questo ambizioso fine è costituito dai primi lavori iniziati nel 1985 che avrebbero portato alla realizzazione di WordNet. Come verrà spiegato in maniera più approfondita nel primo capitolo del presente lavoro di tesi, WordNet costituisce un database che è anche un modello della memoria lessicale umana ed è stato costruito da un équipe di psicologi e linguisti guidata da George Miller presso il laboratorio di Scienze Cognitive all'Università di Princeton. In seguito ulteriori passi sono stati compiuti mediante la realizzazione di EuroWordNet, un database formato da diverse parti, fra loro interfacciate per poter costituire un corpo unico, ognuna delle quali relativa a una specifica lingua europea. Un progetto simile, ma con caratteristiche in parte diverse, porta il nome di MultiWordNet. Un ulteriore passo è rappresentato dalla formazione di database appositamente costruiti, anch'essi inglobati mediante interfacce. Per esempio un progetto italiano che ha visto la collaborazione dell'Istituto di Linguistica Computazionale di Pisa, del Consorzio Pisa Ricerche e dell'Istituto per la Ricerca Scientifica e Tecnologica di Trento con fine ultimo la realizzazione di ItalWordNet, un database semantico per il trattamento automatico del linguaggio italiano, ha portato alla costruzione di una sezione interamente dedicata all'economia.

Sulla base di queste esperienze l'obiettivo della presente tesi è costituito dalla realizzazione di un nuovo database relativo al linguaggio di base della matematica italiana con eventuali sviluppi futuri che portino alla creazione di un'interfaccia che ne permetta l'inserimento in ItalWordNet e quindi in EuroWordNet.

Capitolo 1

WordNet

A una prima occhiata WordNet altro non è che un dizionario on-line, cioè un database lessicale che può essere letto tramite un computer. Il primo vantaggio che deriva dall'utilizzo di un calcolatore è una velocità di ricerca superiore a quella di un essere umano a contatto con un dizionario su supporto cartaceo. Tuttavia risulta evidente che utilizzare un elaboratore come semplice 'cercatore di informazione' è non sfruttare a pieno le potenzialità a disposizione. Per questa ragione realizzando WordNet si è ottenuto molto di più di un semplice dizionario on-line ed è stato pensato e costruito un modo completamente nuovo di strutturare le informazioni.

WordNet non è organizzato alfabeticamente, ma su base concettuale. Le informazioni sono memorizzate in base al significato e collegate fra loro per mezzo di una serie di riferimenti che permettono di soddisfare ad una ricerca fornendo non solo la risposta cercata, ma informazioni aggiuntive sia a livello generale che a livello specifico, dettagliato. WordNet costituisce un 'navigatore' all'interno dell'informazione, esso infatti aiuta l'utente nel reperimento guidandolo tanto nella risalita verso la generalità quanto nella discesa ver-

so la particolarizzazione, verso lo specifico. Alla base di tutto questo, come detto precedentemente, vi è un sistema per memorizzare l'informazione che passa da uno schema classico del tipo *{nome, definizione}* ad uno più articolato *{nome, definizione, puntatori}* dove i puntatori sono riferimenti ad altre informazioni contenute nel database.

Possiamo fare un esempio utilizzando WordNet 2.0. Andiamo a cercare la parola `dog` e otteniamo sette possibili significati (successivamente vedremo come sono organizzati tali significati). A questo punto possiamo navigare nell'informazione in vari modi, per esempio spostandoci verso la generalità alla ricerca degli iperonimi ottenendo la seguente *gerarchia*:

dog

⇒ canine

⇒ carnivore

⇒ placental, plac. mammal, eutherian, euth. mammal

⇒ mammal

⇒ vertebrate, craniate

...

Possiamo poi spostarci verso lo specifico ottenendo una diversa gerarchia di cui riportiamo una parte piccola, ma significativa:

dog

⇒ pooch, doggie, doggy, barker, bow-wow

...

⇒ toy-dog, toy

⇒ Chihuahua

⇒ Japanese spaniel

⇒ Maltese dog, Maltese

⇒ Pekinese, Pekingese, Peke

...

...

1.1 Genesi di WordNet

Una nuova scienza, chiamata psicolinguistica, è nata in seguito allo studio del modello della memoria lessicale umana. Questa scienza coinvolge varie discipline di ricerca e non può prescindere da competenze linguistiche di base. Per tali motivi i linguisti sono stati protagonisti di questi studi e hanno portato, nel 1985, alla nascita di WordNet come risultato di un progetto cui hanno lavorato psicologi e linguisti dell'Università di Princeton. L'idea iniziale era di provvedere una ricerca concettuale nei dizionari che potesse costituire una risorsa in più rispetto alla ricerca alfabetica. Col procedere dei lavori le ambizioni diventarono sempre più alte fino a passare da un semplice meccanismo di ricerca alla costituzione di un intero dizionario basato sui principi della psicolinguistica.

In WordNet si notano delle evidenti differenze rispetto ai comuni dizionari. La prima è la divisione del lessico in cinque categorie:

- nomi;
- verbi;

- aggettivi;
- avverbi;
- parole funzione.

Questa divisione è frutto di una diversa organizzazione del lessico a seconda della categoria in cui ci si trova. La struttura con cui i nomi vengono memorizzati è differente da quella con cui si fa lo stesso per gli aggettivi o i verbi. Questa situazione è dovuta alla differenti relazioni con cui gli esseri umani sono portati a collegare i nomi (iperonimia/iponimia) e, per esempio, gli aggettivi (similitudine/antonimia). Il vantaggio di questa suddivisione consiste quindi nella presenza di una organizzazione chiara e consistente all'interno delle diverse categorie. Lo svantaggio è la presenza di una certa ridondanza che sarà evidente anche nel database matematico proposto in questo trattato.

L'obiettivo di WordNet e della presente tesi è di organizzare le informazioni lessicali secondo il significato e non secondo la forma e cercare di farlo in maniera corretta e non ridondante. Per capire come sia possibile far questo è necessario capire la struttura base su cui si fonda WordNet (Miller e Fellbaum, 1991).

1.2 La matrice lessicale

Per organizzare WordNet dobbiamo capire bene quali 'pezzi' sono alla base della nostra costruzione ed è perciò importante associare il giusto significato al nostro elemento base: la parola.

Definizione 1 *La parola è un'associazione fra un concetto lessicale e la stringa che lo rappresenta.*

La domanda che ci poniamo dopo aver definito cos'è per noi una parola è la seguente: qual è la natura e l'organizzazione dei concetti lessicali che le parole esprimono? La risposta più immediata è che tale organizzazione dovrà riflettere la presenza di due elementi in una parola: il concetto lessicale o significato ('word meaning') e la stringa o forma ('word form').

Una prima osservazione ci porta a stabilire che la relazione fra le forme di parola e i significati è del tipo 'molti a molti', infatti ad una stringa possono corrispondere più significati (polisemia) e ad un significato possono essere associate più forme (sinonimia). Da questa importante constatazione nasce il concetto di matrice lessicale rappresentato nella tabella 1.1.

significato	forma di parola				
	F_1	F_2	F_3	$\cdots$	F_n
M_1	$E_{1,1}$	$E_{1,2}$			
M_2		$E_{2,2}$			
M_3			$E_{3,3}$		
$\cdots$				$\cdots$	
M_m					$E_{n,m}$

Tabella 1.1: Matrice lessicale

Risulta evidente come polisemia e sinonimia siano aspetti complementari della stessa mappa e, come vedremo, risulteranno fondamentali per l'organizzazione delle parole in WordNet.

1.3 Forma e significato, come rappresentarli?

Dalle considerazioni della sezione precedente si intuisce come sarà necessario rappresentare in WordNet due diverse relazioni:

- relazioni **semantiche** fra i significati;
- relazioni **lessicali** fra le forme.

Questi due concetti vengono utilizzati per memorizzare le parole e collegarle fra loro. Il metodo con cui ciò avviene dipende dalla teoria scelta per l'organizzazione della base di dati. Esistono infatti due teorie: **costruttiva** e **differenziale**.

Secondo la prima teoria la rappresentazione di un concetto deve presentare elementi a sufficienza per la costruzione del concetto stesso. Per identificare e caratterizzare un significato deve essere portato un numero di informazioni sufficiente non solo a distinguerlo da altri possibili concetti lessicali, ma a definirlo in maniera corretta. Facciamo un esempio considerando una parola che abbia più di un significato (polisemia): parco. Tale forma può assumere due significati:

1. terreno boscoso e piuttosto esteso, spesso recintato ed adibito a usi particolari;
2. sobrio, frugale, parsimonioso.

Se definendo il parco (riferito al primo significato) dicessimo che può essere recintato, lo differenzieremmo senz'altro dal secondo, in quanto la parola recintato non ha alcun senso riferita al significato sobrio, ma non potremmo risalire al senso perché un campo o un cortile può essere recintato senza

essere un parco. In una definizione costruttiva dobbiamo fornire abbastanza informazioni da poter *costruire* il significato del termine esaminato.

Invece la seconda teoria, più semplice, si preoccupa unicamente della presenza di conoscenze che permettano di *distinguere* due elementi differenti. In questo senso viene chiamata differenziale. L'idea è quella di sfruttare quanto espresso dalla matrice lessicale per permetterci di separare i vari significati (ovviamente la differenza delle forme comporta un immediata distinzione fra esse). In pratica si tratta di rappresentare un significato M_x per mezzo dell'insieme delle forme che lo supportano $F_1, F_2, \ldots, F_n$. Così facendo otteniamo per ogni significato un insieme di forme di parola in relazione di sinonimia (hanno tutte lo stesso significato); tale insieme è detto **synset** dalla contrazione di synonym set (insieme di sinonimi) ed è rappresentato dalla stringa $\{x_0, x_1, \ldots, x_n\}$ dove $x_0, \ldots, x_n$ sono gli elementi dell'insieme. La cosa importante è data dal fatto che i synset non spiegano cosa sono i concetti, ma affermano che i concetti esistono.

1.4 Le relazioni su cui si basa WordNet

WordNet è una base di dati che si poggia, come abbiamo visto e come sarà più chiaro in seguito, sull'associazione fra concetti, ma queste associazioni vengono fatte studiando le relazioni lessicali e semantiche esistenti. Per questo esaminiamo alcune importanti relazioni:

- sinonimia;
- iponimia/iperonimia;
- meronimia/olonimia;

- antonimia;
- relazioni morfologiche.

Abbiamo preferito mantenere distinto lo studio di sinonimia e antonimia per il diverso uso che viene fatto di queste due relazioni nella costruzione di WordNet.

1.4.1 Sinonimia, il concetto alla base dei synset

La prima relazione importante al fine dell'organizzazione di WordNet è quindi la relazione di sinonimia. Questa relazione ci permette non tanto di legare diversi significati, ma di rappresentarli. Ogni concetto è identificato da un synset, insieme costruito unendo elementi fra loro sinonimi.

Cerchiamo di definire cos'è la sinonimia:

Definizione 2 *Due concetti sono fra loro sinonimi se la sostituzione di uno con l'altro non altera il valore di verità della frase in cui tale sostituzione avviene.*

Questa è una definizione molto stringente e, in base ad essa, ben pochi termini si possono considerare sinonimi. Se la tenessimo presente per la costruzione del nostro dizionario non riusciremmo ad ottenere che synset formati da una o due forme di parola. Per questo si è scelto, per la realizzazione di WordNet, di adottare un concetto di sinonimia più adatto allo scopo ridefinendo la sinonimia come segue:

Definizione 3 *Due concetti sono fra loro sinonimi in un determinato contesto linguistico C se la sostituzione di uno con l'altro non altera il valore di verità della frase in tale contesto linguistico.*

Oltre a basarsi su questa nuova definizione ci si accontenta, nella definizione dei synset, anche della semplice similarità perché in molti casi non si può considerare la sinonimia come un fatto discreto, vero o falso, come le definizioni date sembrano affermare. Quello che vogliamo è ottenere degli insiemi di forme di parole che ci permettano di distinguere un significato da un altro e per raggiungere questo scopo la similarità ci basta.

Cerchiamo di chiarire con un esempio. La parola *tree*, (*albero*), ha un duplice significato: possiamo intendere l'albero inteso come pianta oppure un diagramma ad albero. Vengono quindi creati due synset per ditinguere i due concetti:

- {tree, woody plant, ligneous plant}
- {tree, plan figure, two-dimensional figure}

Nel secondo synset vediamo che la parola *albero* non è assolutamente sostituibile da *figura piana*, perché esistono *figure piane* diverse dall'*albero*, ma il nostro scopo è ottenere una distinzione fra i due significati e questo riesce benissimo grazie alla similarità fra *albero* e *figura piana*.

1.4.2 Iponimia

La relazione di iponimia/iperonimia è una delle più importanti presenti in WordNet ed è detta anche relazione di specializzazione/generalizzazione. Si dice che un concetto (rappresentato da un synset $\{x_0, x_1, \ldots, x_n\}$) è un iponimo di $\{y_0, y_1, \ldots, y_n\}$ se si può rispondere affermativamente alla domanda $\{x_0, x_1, \ldots, x_n\}$ è un tipo di $\{y_0, y_1, \ldots, y_n\}$?. In questo caso si dice che $\{x_0, x_1, \ldots, x_n\}$ è un iperonimo di $\{y_0, y_1, \ldots, y_n\}$. Questa relazione è anti-

simmetrica e transitiva ed è rappresentata in WordNet da due puntatori fra i synset coinvolti.

Si dice che la struttura derivante dall'applicazione della relazione di iponimia/iperonimia ai synset è gerarchica in quanto da un concetto padre derivano più concetti figli che condividono delle caratteristiche base con il padre, ma presentano delle peculiarità proprie che li rendono particolari. Questa gerarchia è molto comoda per la costruzione di una base di dati per due motivi principali:

- permette di ridurre la ridondanza dei dati in quanto le caratteristiche condivise dal padre e da tutti i figli vengono immagazzinate solo per il padre;
- agevolano la ricerca permettendo di passare facilmente dalla generalità alla specificità rispettivamente risalendo o scendendo le gerarchie presenti.

Questa relazione è alla base della strutturazione dei nomi.

1.4.3 Meronimia

La relazione di meronimia/olonimia è chiamata anche con il nome 'parte di' o 'parte/tutto'. Si dice che un synset $\{x_0, x_1, \ldots, x_n\}$ è meronimo di un synset $\{y_0, y_1, \ldots, y_n\}$ se si può rispondere affermativamente alla domanda '$\{x_0, x_1, \ldots, x_n\}$ è una parte di $\{y_0, y_1, \ldots, y_n\}$?'. Questa relazione è pure antisimmetrica e transitiva ed è rappresentata in WordNet da puntatori fra synset diversi.

Anche questa relazione è semantica e porta alla costituzione di una gerarchia. È tuttavia necessario considerare che esistono tipi diversi di rapporti

parte/tutto e, se questi non vengono considerati in tale diversità, si perde la proprietà transitiva. Spieghiamolo con un esempio: il *piede* è parte di un *giocatore di calcio* e quest'ultimo è parte di una *squadra*, ma non è corretto dire che il *piede* è parte di una *squadra*. Questo perché il rapporto fra il *piede* e il *giocatore di calcio* è un rapporto di tipo componente/oggetto mentre il rapporto fra il *giocatore di calcio* e *squadra* è del tipo membro/insieme. Si possono individuare sette diversi tipi della relazione parte/tutto e precisamente:

1. componente/oggetto: i componenti sono parte integrante di un oggetto, possono essere separati da esso e rispetto allo stesso hanno una funzione specifica (gambe/sedia);

2. membro/insieme: è caratterizzata dalla relazione di appartenenza presente in matematica e non solo, un membro è separabile dal tutto (giocatore/squadra);

3. porzione/massa: la massa costituisce un tutto omogeneo di cui le porzioni fanno parte e possono esserne separate (fetta/torta);

4. materiale/oggetto: è la relazione fra il tutto e i materiali presenti in esso, i materiali non sono separabili e non hanno una funzione specifica rispetto all'oggetto (fibra di carbonio/bicicletta);

5. caratteristica/attività: una caratteristica descrive una fase dell'attività intesa come processo, ogni fase è funzionale al completamento dell'attività (progettazione concettuale/realizzazione di una base di dati);

6. posto/area: è una relazione spaziale fra due aree fisiche, ogni posto è dello stesso tipo dell'area e non può esserne separato (Italia/Europa);

7. fase/processo: la fase indica un periodo temporale che fa parte dell'evoluzione del processo e non è da quest'ultimo separabile (adolescenza/crescita).

Di questi tipi di relazione in WordNet vengono considerati i tre seguenti che hanno permesso di costruire gerarchie sufficientemente significative:

1. componente/oggetto;

2. membro/insieme;

3. materiale/oggetto.

Statisticamente si è visto che il primo di questi tre tipi risulta essere presente con maggior frequenza.

1.4.4 Antonimia

Questa relazione è solo apparentemente semplice, in realtà è molto difficile da definire. Infatti l'antonimo di un concetto x a volte è ¬x, ma non sempre (il simbolo ¬ rappresenta la negazione). L'esempio più ricorrente è quello di *ricco* e *povero*. Le due parole sono antonime, ma la negazione di *ricco*, cioè *non ricco* non è un antonimo della parola di partenza. Per questo la relazione di antonimia non è una semplice relazione simmetrica come sembrerebbe. Per renderla consistente essa viene considerata come una relazione sulle forme di parola e non sui significati, quindi risultano antonime solo le forme di parola che lo sono per 'costruzione'.

1.4.5 Relazioni morfologiche

Di grande importanza sono le relazioni morfologiche che sono basate sulle forme di parola e che diventano fondamentali per evitare inutili ridondanze nella base di dati e permettere allo stesso tempo una certa libertà di ricerca all'utente. Un esempio può essere illuminante. Supponiamo che l'utente intenda cercare la parola *alberi* e la inserisca. Di certo non è memorizzata nella nostra base di dati tanto la parola *albero* che i suoi plurali perché sarebbe un'inutile ridondanza. *Albero* e *alberi* sono in relazione morfologica ed è importante studiare un metodo per ricondurre queste due parole allo stesso synset.

1.5 WordNet e le categorie lessicali

Discutendo i principi alla base della costruzione di WordNet abbiamo evidenziato come sia necessario in diverse situazioni fare una separazione fra le diverse categorie lessicali. Alla base di questa scelta ci sono varie motivazioni che possono essere riassunte con le seguenti considerazioni:

- la rappresentazione dei concetti avviene per mezzo dei synset, cioè insiemi di sinonimi, ma la sinonimia è una relazione che non può essere definita trasversalmente alle diverse categorie;
- all'interno delle diverse categorie lessicali è presente un'organizzazione gerarchica distinta, basata su relazioni base diverse a seconda della categoria considerata.

Vediamo ora come si procede all'organizzazione dei synset all'interno di ogni categoria con particolare interesse per i nomi.

1.5.1 Nomi

I nomi sono organizzati secondo una struttura gerarchica con ereditarietà in base alla relazione di iponimia/iperonimia. Tale struttura è ad albero e questo permette di eliminare il rischio delle cosiddette definizioni circolari (la parola W_a è definita in base alla parola W_b e viceversa). La relazione di iponimia è quella che caratterizza la formazione dell'albero e, perciò, avremo un maggior livello di generalità risalendo l'albero, mentre il bisogno di specificità è soddisfatto ai livelli più bassi dell'albero stesso. Si è soliti distinguere un 'basic level' al di sopra del quale le definizioni sono generali, mentre al di sotto diventano particolareggiate. Ogni synset è collegato con dei puntatori tanto ai suoi iponimi quanto ai suoi iperonimi.

Gerarchie di questo tipo sono molto utilizzate in database di grosse dimensioni. Il vantaggio è la possibilità di risparmiare spazio evitando le ridondanze (cfr. 1.4.2).

Un modo di costruire l'albero è considerare tutti i nomi appartenenti ad un'unica gerarchia. Per far questo è necessario porre come radice qualcosa di molto astratto e da essa partire (vedi figura 1.1).

All'atto pratico questo modo di procedere genera numerosi concetti che portano con sé pochissima informazione semantica. Per questo si è attuata una scelta alternativa: partizionare l'insieme dei nomi per mezzo di un insieme di synset primitivi, ognuno dei quali radice unica di una gerarchia indipendente. Questi synset corrispondono a campi semantici distinti. Un altro vantaggio conseguente alla loro creazione è la minor dimensione dei singoli 'lexicographers files', cioè dei file contenenti i synset. Le 25 radici alla base dei nomi in WordNet sono riportate in tabella 1.2. Non è facile

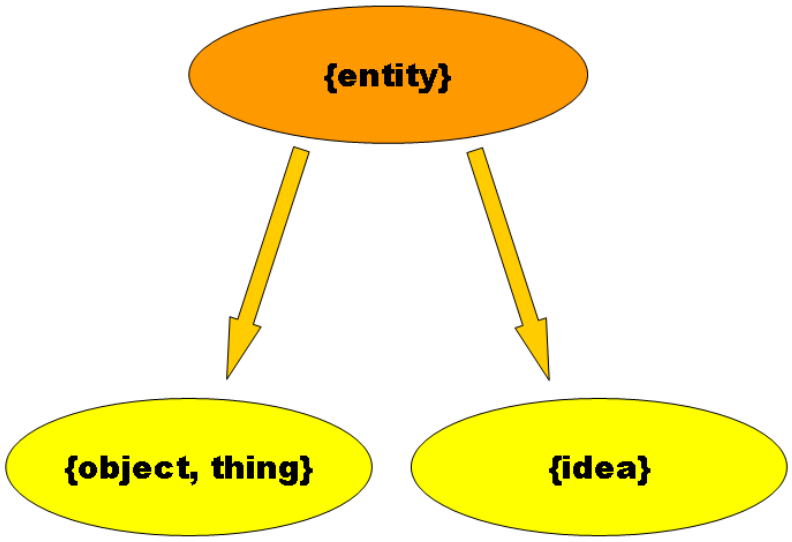

Figura 1.1: Nomi: possibile radice di un'unica gerarchia

esplicitare dei synset di partenza con la proprietà che ogni parola del vocabolario trovi posto in una gerarchia discendente da una di queste radici. Le soluzioni possono essere molteplici ed è difficile poter affermare di aver risolto il problema in modo ottimo. Per questo è giusto dire che i synset che rappresentano le 25 radici sono stati eletti a quel ruolo per scelta, in modo da creare dei nodi di partenza da cui le gerarchie potessero essere sviluppate. È doveroso sottolineare che tali gerarchie non sono mutuamente esclusive, vale a dire che sono presenti numerosi riferimenti incrociati fra significati presenti in diverse gerarchie. Ad esempio consideriamo la parola *man* (*uomo*), essa appartiene alla gerarchia che ha come radice *{person, human being}*, ma costituisce pure un iponimo del termine *essere vivente* che è presente nella gerarchia originata da *{natural object}*. In questo caso è presente una re-

{art, action, activity}	{natural object}
{animal, fauna}	{natural phenomenon}
{artifact}	{person, human being}
{attribute, property}	{plant, flora}
{body, corpus}	{possession}
{cognition, knowledge}	{process}
{communication}	{quantity, amount}
{event, happening}	{relation}
{feeling, emotion}	{shape}
{food}	{state, condition}
{group, collection}	{substance}
{location, place}	{time}
{motive}	

Tabella 1.2: Lista dei 25 synset alla base di WordNet

lazione fra synset appartenenti a differenti gerarchie confermando che esse non sono mutuamente esclusive.

È interessante notare come si possano trovare delle relazioni fra gli stessi synset radice, in conseguenza di ciò è stato creato un piccolo 'top-file' che le esplicita. In figura 1.2 è rappresentata la relazione fra 7 dei 25 insiemi iniziali.

Vista la struttura gerarchica governata dalla relazione di iponimia le differenze fra i vari significati sono formalizzate esplicitando le caratteristiche che distinguono un concetto da un altro. Tali caratteristiche distintive, che vanno trattate separatamente, sono di tre tipologie:

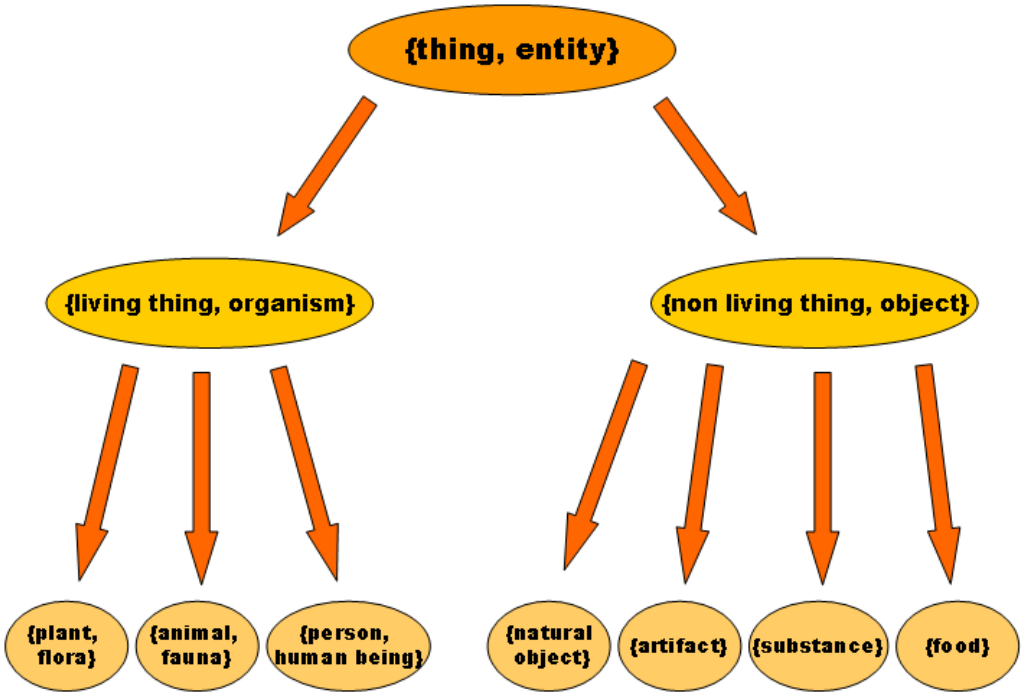

Figura 1.2: Relazioni fra 7 synset di partenza

- attributi;
- parti (meronimia);
- funzioni.

È importante notare che, di queste tre, solo la caratteristica denominata 'parti' è rappresentata da altri nomi, mentre gli 'attributi' sono aggettivi e le 'funzioni' sono verbi. Questo ci fa dedurre che solo la meronimia può essere descritta da puntatori interni alla categoria nomi, mentre le altre due caratteristiche possono essere esplicitate solo con puntatori verso categorie distinte. Questo non è stato realizzato nelle prime versioni di WordNet però è

giusto tenerne conto perché fornisce potenzialità molto elevate. Ad esempio sarebbe interessante poter collegare il termine matematico *integrale* con il verbo associato *integrare* e con gli aggettivi *definito* e *indefinito*.

Per quanto riguarda la meronimia essa è rappresentata senza difficoltà grazie a puntatori verso altri nomi. È necessario evidenziare che vi sono tre tipi di puntatori in quanto, cfr. 1.4.3, vengono prese in considerazione tre diversi tipi di relazioni.

Un'ultima relazione non necessaria all'organizzazione dei nomi, ma che può essere utile, è l'antonimia. Essa viene inclusa in WordNet rimanendo opzionale come vedremo studiando l'organizzazione fisica di WordNet (cfr. 1.6).

La rete di nomi che si ottiene considerando le tre relazioni semantiche definite (iperonimia, meronimia, antonimia) è una rete piuttosto connessa (vedi figura 1.3).

1.5.2 Aggettivi

In WordNet vengono distinti aggettivi di tre diverse tipologie:

- aggettivi descrittivi;
- aggettivi relazionali;
- aggetivi 'reference-modifying'.

Un aggettivo descrittivo associa un valore a un attributo del nome cui esso è associato. Pensiamo per esempio a *casa grande*: *grande* è un aggettivo descrittivo riferito all'attributo 'dimensione' del nome *casa*. Questa tipologia di aggettivi non è organizzata secondo un albero o una gerarchia come per i

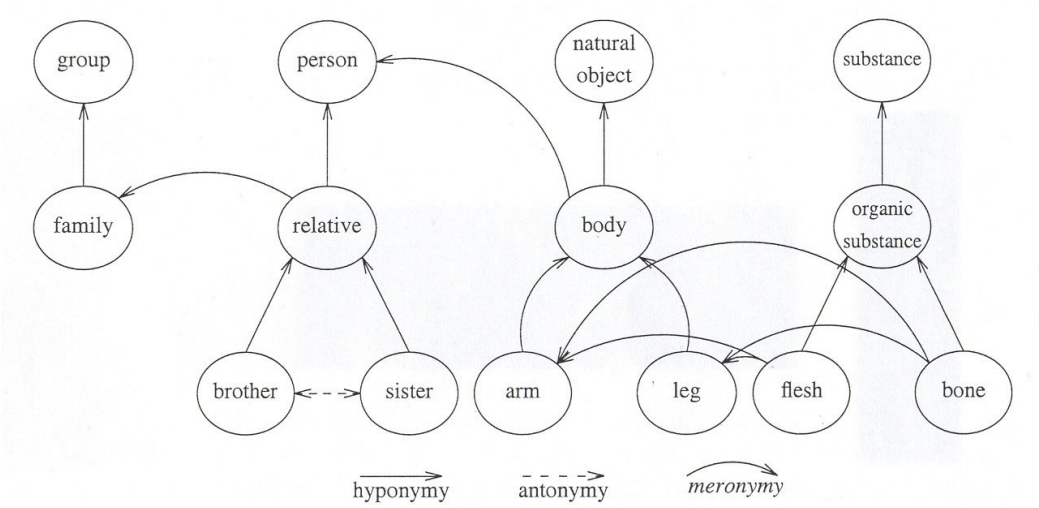

Figura 1.3: Rappresentazione della rete con le tre relazioni semantiche (iperonimia, meronimia, antonimia) su una varietà linguistica di esempio

nomi, ma per mezzo di uno spazio dimensionale di cui sono noti gli estremi. In riferimento all'attributo 'grandezza' viene definita una scala di valori (spazio unidimensionale) estesa, ad esempio, fra gli estremi *infinitesimo* ed *enorme*. La relazione fondamentale per la definizione di queste scale è l'antonimia (cfr. 1.4.4).

Gli aggettivi relazionali sono aggettivi che derivano dai nomi (ad esempio *atomico* deriva da *atomo*). Essi non si riferiscono ad una proprietà del nome che modificano e non sono quindi associati ad un attributo. Aggettivo relazionale significa qualcosa del tipo 'relativo/pertinente a, associato con' un nome. Essi inoltre non sono organizzati come gli aggettivi descrittivi perché per essi non è significativa la relazione di antonimia. Questi aggettivi sono mantenuti in una lista distinta (vedi tabella 1.3) e puntano al nome da cui derivano (puntatori trasversali alle categorie).

...	...
atomico	atomo
fraterno	fratello
dentale	dente
musicale	musica
criminale	crimine
...	...

Tabella 1.3: Lista di aggettivi relazionali

Per quanto riguarda l'ultima tipologia (gli aggettivi 'reference-modifying') possiamo dire che un numero piuttosto piccolo di aggettivi, comprendente *former* (*precedente*) e *alleged* (*fatto supposto vero senza che vi sia una dimostrazione di esso*), costituiscono una classe chiusa che viene mantenuta separata dagli altri aggettivi. Spieghiamo che tipologia di aggettivi costituiscono con un esempio, consideriamo la frase seguente: 'Ieri ho incontrato un mio *vecchio* amico'. L'aggettivo *vecchio* può avere due collocazioni:

- è riferito alla persona amica e indica che tale persona è anziana ('referent-modifyng', modifica il referente);
- è riferito all'amicizia indicando che è di lunga data ('reference-modifyng').

In quest'ultimo caso abbiamo la tipologia di aggettivi in esame, cioè quelle parole che modificano la 'referenza' che passa dall'amico all'amicizia. Un altro esempio: se dico 'il preside precedente' modifico la persona cui mi riferisco passando dal preside a colui che lo precedeva. Da ultimo possiamo aggiungere che spesso questi aggettivi funzionano come avverbi (esempio: *my*

former teacher e *he was formerly my teacher*), che sono sempre attributivi e mai predicativi e che spesso hanno antonimi diretti.

1.5.3 Verbi

I verbi, compresi i 'phrasal verbs' (frasi verbali come, ad esempio, *stare per*, *accingersi a*, ...), hanno in WordNet un'organizzazione gerarchica. Essi sono anzitutto divisi in due categorie:

- verbi che denotano azioni o eventi;
- verbi che denotano stati.

I primi sono a loro volta suddivisi in quattordici tipologie corrispondenti a quattordici diversi gruppi semantici. Per quanto riguarda i secondi, essi costituiscono un raggruppamento unico.

Le relazioni in base a cui sono organizzati i verbi sono ancora due:

- implicazione;
- opposizione.

Vengono considerate quattro diverse implicazioni a seconda del rapporto temporale fra il verbo implicante e il verbo implicato, in particolare se c'è inclusione temporale fra i due si parla di co-occorrenza o troponimia (ad esempio: *zoppicare* implica *camminare*) e questa relazione corrisponde a quella che per i nomi è detta iponimia (cfr. 1.4.2). In pratica si dice che un verbo è troponimo di un altro se si può dire che il primo verbo è un 'modo' dell'altro. Questa relazione è particolarmente importante perché permette la costituzione di una struttura gerarchica che è tuttavia diversa da quella per

i nomi in quanto all'interno di un campo semantico non tutti i verbi possono essere riuniti sotto un unico nodo. Per quanto riguarda l'opposizione va evidenziato che viene codificata solamente l'opposizione lessicale, non quella semantica (cfr. 1.4.4).

Per quanto riguarda l'informazione associata ad ogni verbo in WordNet possiamo dire che ne viene specificata la struttura predicato-argomento: sono attribuiti ruoli tematici ai nomi che fungono da argomenti del verbo e vengono specificate le proprietà semantiche riguardanti le classi di nomi che possono costituire l'argomento per un verbo. Infine, per ogni verbo del synset, vengono descritte una o più strutture (frames) che specificano le caratteristiche di sottocategorizzazione dei verbi (indicando le frasi in cui possono comparire).

1.5.4 Avverbi

Gli avverbi sono memorizzati in maniera molto semplice in quanto non è presente né una struttura gerarchica (come per i nomi e per i verbi) né un'organizzazione in gruppi (come per gli aggettivi). Per ogni avverbio sono indicati i sinonimi (cfr. 1.4.1) e gli antonimi (cfr. 1.4.4) e, quando derivano da un aggettivo, questo viene specificato.

Gli avverbi costituiscono una categoria molto eterogenea (anche per la presenza di numerosissime locuzioni avverbiali), ma non è mai stato fatto alcun tentativo allo scopo di suddividerli in gruppi distinti.

1.6 WordNet: l'organizzazione fisica

WordNet si può definire un database lessicale dotato di particolari software di ricerca. Dal punto di vista del disegno e dell'implementazione di tale database si possono distinguere tre grandi fasi:

1. scrittura dei 'source files', file testuali che contengono i dati lessicali;
2. creazione di un set di programmi software che permettano l'archiviazione dei source file in una struttura dati e la successiva ricerca delle informazioni all'interno di tale struttura;
3. realizzazione di un ambiente grafico che permetta l'interazione dell'utente con il sistema intesa come possibilità di fare ricerche e ottenere risultati.

1.6.1 I source file

I source file non sono altro che file puramente testuali contenenti i synset e quindi tutte le informazioni che permettono di inserire significati e collegamenti fra essi nella base di dati. Essi sono scritti dai lessicografi e sono il frutto di un'analisi molto approfondita che tiene conto di numerose relazioni lessicali e semantiche. Questi file sono la base su cui WordNet è costruito.

Sono presenti source file distinti per le diverse categorie lessicali (cfr. 1.5), e ogni categoria lessicale comprende generalmente più di un file (in tabella 1.4 sono riportati numerosi source file relativi alle diverse categorie). I nomi di questi file sono della forma:

pos.suffix

adj.all	'clusters' per tutti gli aggettivi
adj.pert	aggettivi relazionali
adv.all	avverbi
noun.Tops	radici della categoria nomi
noun.act	nomi che denotano atti o azioni
noun.animal	nomi che denotano animali
noun.artifact	nomi che denotano oggetti fatti dall'uomo
noun.attribute	nomi che denotano attributi
...	...
noun.event	nomi che denotano eventi naturali
noun.feeling	nomi che denotano sensazioni ed emozioni
noun.food	nomi che denotano cibi e bevande
...	...
verb.body	verbi attinenti al corpo
verb.change	verbi attinenti a grandezze e loro variazioni
...	...
verb.wheater	verbi attinenti espressioni climatiche
adj.ppl	aggettivi participi

Tabella 1.4: Divisione dei source file per campo semantico

dove *pos* può essere *noun*, *verb*, *adj* o *adv*; mentre *suffix* caratterizza la categoria tematica dei significati presenti in quel determinato file (per esempio in tabella sono presenti *event*, *feeling*, *food*).

Essi sono costruiti su due elementi base: **forme di parola** (word forms) e **significati** (word meanings). Le forme di parola sono costituite dalle rappresentazioni ortografiche delle parole, i significati sono i nostri 'synonym sets' (synset). Ad esempio *car* costituisce una forma di parola in quanto consiste in una rappresentazione ortografica. Il significato italiano *automobile* è invece rappresentato da un insieme di sinonimi, cioè un synset: *car, auto, automobile, machine, motorcar*.

Per ogni forma di parola sono memorizzate in WordNet quattro informazioni:

- rappresentazione ortografica (se la forma di parola è una collocazione, quindi è una forma costituita da più parole, queste vengono unite per mezzo del simbolo underscore), per esempio: *car*, *motor_vehicle*;
- categoria lessicale (si evince dal file cui appartiene), per esempio la parola *car*, essendo un nome (in inglese *noun*), apparterrà ad un file denominato *noun.suffix*;
- campo semantico (si deduce dal synset cui appartiene), ad esempio la forma *man* e i synset ad essa relativi saranno contenuti nel file *noun.person* (per quanto riguarda i nomi dei source file e il campo semantico ad essi associato rimandiamo il lettore alla tabella 1.4);
- indice di familiarità.

È opportuno fare qualche considerazione per quanto concerne l'indice di familiarità. Esso costituisce un numero che caratterizza la 'familiarità' di una parola rispetto ad un'altra. Tale parametro condiziona alcune variabili quali, ad esempio, la velocità di lettura e di comprensione. Un buon metodo per descrivere l'indice consiste nel calcolare la frequenza di ogni parola in un testo di riferimento. Quest'operazione è tuttavia non adeguata ad un database esteso come WordNet. Si può affermare (Zipf, 1945) che la frequenza delle occorrenze delle parole in un testo e la polisemia sono correlate. Con questo intendiamo che, in media, le parole più utilizzate in un testo sono quelle che hanno un maggior numero di significati diversi in un dizionario. Per questo in WordNet si associa ad una forma di parola un numero che descrive i diversi sensi che una forma di parola assume (se vogliamo un 'indice di polisemia'). Questa misura è di facile realizzazione: basta prendere in esame un dizionario on-line e associare un indice pari a 0 alle parole che non compaiono e un indice pari a 1 o maggiore in accordo al numero di significati diversi associati ad esse dal dizionario. In WordNet ad ogni forma di parola[1] è associato un indice intero pari al numero di significati che una forma può assumere quando è utilizzata come nome, verbo, aggettivo o avverbio nel 'Dictionary of the English Language' di Collins). L'utilità di tale indice è evidente nell'esempio seguente in cui riportiamo le gerarchie che risalgono dal termine *bronco* alla radice attraverso gli iperonimi, prima considerando ogni livello dell'albero, poi tenendo conto solamente dei termini che hanno indice di familiarità superiore a 1. Solitamente le forme con familiarità 0 o 1 costituiscono termini

[1] WordNet diverrà una 'simulazione' ancora migliore della memoria umana nel momento in cui si riuscirà ad associare un indice di familiarità valido alla coppia forma-significato anziché solamente alla forma di parola.

molto tecnici e poco utilizzati. Nel primo caso otteniamo:

bronco
⇒ mustang
⇒ pony
⇒ horse
⇒ equine
⇒ odd-toed ungulate
⇒ placental mammal
⇒ mammal
⇒ vertebrate
⇒ chordate
⇒ animal
⇒ organism
⇒ entity

Nel secondo abbiamo invece:

bronco
⇒ pony
⇒ horse
⇒ animal
⇒ organism
⇒ entity

Questa seconda catena è più vicina a quanto un utente si può attendere e omette tutti quei termini specialistici che spesso non sono di interesse.

Definite le forme di parola è necessario descrivere i puntatori che esprimeranno le relazioni lessicali e semantiche tra forme di parola presenti in synset distinti. I puntatori lessicali esprimono relazioni tra forme di parola e quindi interessano solamente una forma del synset di partenza e una del synset di arrivo (pensiamo ad esempio agli aggettivi relazionali che sono collegati alla forma di parola che rappresenta il nome da cui derivano: *atomic* da *atom* o *fraternal* da *brother*. I puntatori semantici individuano relazioni fra significati e quindi interessano interi synset. Un puntatore fra due synset viene specificato definendo nel synset di partenza una forma di parola del synset di arrivo seguita da una virgola e da un simbolo che specifica il tipo di puntatore (e quindi il tipo di relazione fra i due synset: iponimia, iperonimia, ...). Pensiamo ad esempio al synset *{collie dog1,@ ...}* in cui il synset che rappresenta il cane di razza collie è collegato al suo iponimo. I puntatori sono quindi della forma:

forma di parola, simbolo puntatore

La base di dati che andremo a costruire nel corso della tesi utilizzerà gli stessi simboli di WordNet presenti nelle tabelle 1.5(avverbi), 1.6(aggettivi), 1.7(verbi), 1.8(nomi).

Molti puntatori godono della proprietà simmetrica e questo comporta un collegamento doppio fra due synset, tali puntatori sono riportati con il proprio corrispondente in tabella 1.9.

!	antonimia
\	deriva dall'aggettivo
;c	dominio del synset - CATEGORIA
;r	dominio del synset - REGIONE
;u	dominio del synset - USO

Tabella 1.5: Simboli dei puntatori per la categoria avverbi

!	antonimia
&	similarità
<	participio del verbo
\	deriva dal nome
=	attributo
⋀	vedi anche
;c	dominio del synset - CATEGORIA
;r	dominio del synset - REGIONE
;u	dominio del synset - USO

Tabella 1.6: Simboli dei puntatori per la categoria aggettivi

!	antonimia
@	iperonimia
$\sim$	iponimia
*	assegnazione
$>$	causa
$\wedge$	vedi anche
$	gruppo verbale
$+$	derivato
;c	dominio del synset - CATEGORIA
;r	dominio del synset - REGIONE
;u	dominio del synset - USO

Tabella 1.7: Simboli dei puntatori per la categoria verbi

I source file contengono svariate righe, ogni riga rappresenta un synset il quale è terminato dal carattere *newline*. La sintassi di un synset è diversa a seconda della categoria lessicale in cui ci muoviamo. Per i nomi abbiamo:

{parola[2]*-puntatori-(glossa)*[3]*}*

I puntatori che seguono le forme di parola in un synset rappresentano relazioni semantiche tra quel synset e il synset obiettivo.

Per i verbi la sintassi del synset base è la seguente:

[2]da qui in avanti scriveremo 'parola' anziché 'forma di parola' per migliorare la leggibilità

[3]le parentesi tonde indicano qualcosa di opzionale

!	antonimia
@	iperonimia
~	iponimia
#m	olonimia (membro/insieme)
#s	olonimia (materiale/oggetto)
#p	olonimia (componente/oggetto)
%m	meronimia (membro/insieme)
%s	meronimia (materiale/oggetto)
%p	meronimia (componente/oggetto)
=	attributo
+	derivato
;c	dominio del synset - CATEGORIA
-c	membro di questo dominio - CATEGORIA
;r	dominio del synset - REGIONE
-r	membro di questo dominio - REGIONE
;u	dominio del synset - USO
-u	membro di questo dominio - USO

Tabella 1.8: Simboli dei puntatori per la categoria nomi

puntatore	**simmetrico**
antonimo	antonimo
iponimo	iperonimo
iperonimo	iponimo
olonimo	meronimo
meronimo	olonimo
similare	similare
attributo	attributo
gruppo verbale	gruppo verbale
derivato	derivato
dominio del synset	membro di questo dominio

Tabella 1.9: Puntatori che godono della proprietà simmetrica

{forme di parola-puntatori-frames-(glossa)}

Per gli aggettivi abbiamo una sintassi che prevede la loro organizzazione in cluster contenenti uno o più synset principali e dei synset satellite opzionali.

[
synset principale
[synset satelliti]
[-]
[synset principale/satelliti opzionali]
]

Come è evidente ogni cluster è racchiuso fra parentesi quadre e può essere composto da una o più parti. I synset satelliti rappresentano aggettivi simili a quello principale, essi possono anche essere omessi, ma, se presenti, sono in relazione di similarità con il synset di riferimento. Interi synset possono essere commentati racchiudendoli fra parentesi tonde ().

Ogni synset ha almeno una forma di parola, tali forme di parola devono comparire subito dopo la parentesi graffa di apertura. Esse possono avere o meno un puntatore. Se non ha il puntatore la forma di parola è del tipo:

$$parola[(marker)][lexid]$$

La forma può contenere caratteri minuscoli o maiuscoli, una collocazione deve avere i singoli termini collegati per mezzo di underscores. Il valore *lexid* è un intero compreso fra 1 e 15 che serve a descrivere il numero di significati diversi associati ad una parola (vedi indice di familiarità a pagina 34). Il valore di default 0 viene omesso, quindi se una forma di parola ha un unico significato il *lexid* viene omesso.

Se il puntatore è presente la sintassi diventa la seguente:

$$[parola[(marker)][lexid], puntatori]$$

Questo è detto 'word/pointer set' e, in questo caso, la relazione coinvolge unicamente la parola e non il synset (relazione lessicale); se il puntatore è

invece specificato all'esterno del word/pointer set la relazione è applicata a tutte le forme di parola del synset e quindi riguarda i significati (relazione semantica).

La sintassi per specificare un puntatore è:

$[pos.suffix :]parola[lexid], simbolo$[4]

oppure:

$[pos.suffix :]parola[lexid]$-$parola[lexid], simbolo$[5]

Nella seconda forma la prima parola è intesa nel synset principale, la seconda in un synset satellite; *pos.suffix* indica il file in cui si trova la forma di parola cui ci si riferisce.

Per i verbi la sintassi viene così estesa:

$[parola, [puntatori] frames]$

Una lista di frame (indicata da frames) è così inserita:
$frames : framenumber[, framenumber \ldots]$

Un'ultima osservazione riguarda la glossa: essa rappresenta una definizione di massima del significato rappresentato dal synset ed è molto utile all'utente che accede a WordNet, ma non ha alcun ruolo al fine dell'organizzazione dei

[4]simbolo del puntatore
[5]simbolo del puntatore

dati. Essa è una stringa che deve essere contenuta fra parentesi tonde.

Concludiamo con alcuni esempi di synset (realmente presenti in WordNet) per le diverse categorie:

- nomi:

  {collie dog1,@ (large multi-colored dog with pointy nose)}

- verbi:

  {interpret construe understand,@ frames: 8}

- aggettivi:

  [

  {[HOT, COLD,!] lukewarm(a), TEPID,∧ warm,& (hot to the touch)}

  {warm, }

  -

  {[COLD, HOT,!] frigid, freezing,& (cold to the touch)}

  {freezing, }

  ]

  }

- avverbi:

  {[badly, adj.all:bad,/ well,!] ill, ('He was badly prepared')}

Capitolo 2

MultiWordNet e EuroWordNet

Una volta realizzato, WordNet ha dimostrato di essere un supporto valido per la ricerca di informazioni e non un semplice dizionario informatico. Per le potenzialità espresse e per le applicazioni, anche commerciali, che questa base di dati ha dimostrato di poter soddisfare si è pensato ad un progetto più ambizioso: un WordNet multilingue e multitematico. L'obiettivo è diventato costruire un sistema governato dai principi di WordNet che riunisse la conoscenza delle diverse lingue e dei diversi ambiti tematici. Gli approcci al fine di raggiungere l'obiettivo sono stati due fra loro distinti:

- MultiWordNet;
- EuroWordNet.

2.1 MultiWordNet

MultiWordNet è un progetto attualmente in fase di sviluppo da parte del Centro per la Ricerca Scientifica e Tecnologica dell'Istituto Trentino di Cul-

tura (ITC irst). Esso prevede la realizzazione di un lessico generico su larga scala per l'italiano basato sulla versione inglese di WordNet.

Più propriamente l'obiettivo del progetto MultiWordNet è creare un oggetto multilingue in cui siano considerate tanto le relazioni intralinguistiche interne alla rappresentazione delle diverse lingue, quanto le relazioni interlinguistiche che costituiscono il collegamento fra lingue diverse. Per rendere questo possibile è necessario ipotizzare che le strutture concettuali di livello lessicale presenti nelle diverse lingue siano confrontabili e, almeno in parte, sovrapponibili. In base a questa assunzione possiamo costruire una matrice lessicale tridimensionale che, sull'esempio della matrice lessicale bidimensionale (cfr. 1.2), ci indica come strutturare MultiWordNet. La struttura,

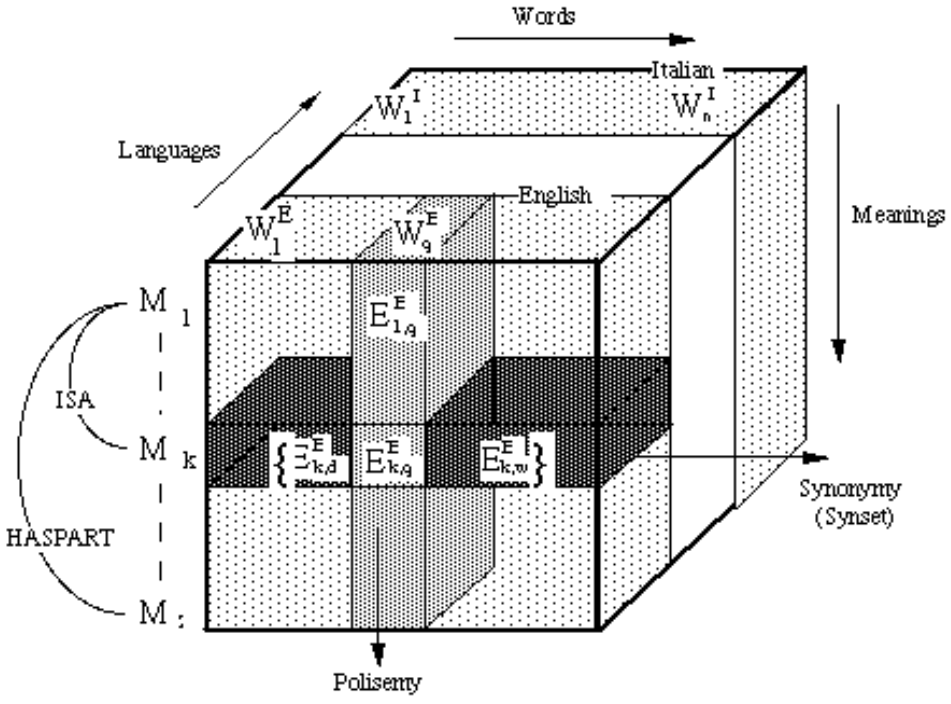

Figura 2.1: Matrice lessicale multilingue

presente in figura 2.1, è costruita per mezzo di una successione di matrici lessicali bidimensionali in cui la dimensione orizzontale presenta le forme di parola $F_1, F_2, \ldots$ che esprimono un dato significato M (sinonimia), mentre la dimensione verticale presenta i diversi significati $M_1, M_2, \ldots$ corrispondenti ad una determinata forma di parola F (polisemia). Su ogni strato verticale è presente la matrice lessicale riferita ad una lingua (in primo piano è presente la lingua inglese, più avanti è riportato il livello della lingua italiana). L'assunzione di cui sopra sulla confrontabilità delle strutture concettuali presenti nelle diverse lingue è fondamentale perché la struttura presentata prevede che i synset di lingue diverse riferiti allo stesso significato siano fra loro corrispondenti. Questo comporta il 'parallelismo' dei vari strati e la corrispondenza non solo dei synset, ma anche delle relazioni semantiche fra essi. Questa assunzione è valida, anche se non sempre corretta, perché è intuitivo come, per culture affini, sia simile il modo di comunicare e di mettere in relazione significati equivalenti. Si può parlare di parallelismo di significati, ma non di forme. Basti pensare all'esempio della parola inglese *pet* di cui non esiste il corrispondente italiano in quanto con *pet* vengono identificati solo cani e gatti domestici mentre l'italiano *animale domestico* individua, oltre a cani e gatti, altri animali quali canarini, criceti, ... Da un punto di vista lessicale, invece, diverse lingue hanno relazioni che si discostano e quindi esse sono mantenute indipendenti in ogni strato verticale e sussistono per la lingua in osservazione. Da un punto di vista pratico questo approccio è molto utile perché permette di costruire i synset di tutte le lingue sulla base di WordNet inglese, e quindi con un insieme di synset e relazioni fra essi già testato e significativo.

A livello di struttura MultiWordNet è organizzato come un insieme di reti monolingue messe assieme per mezzo di una gerarchia comune (denominata WN-comune). Questa costituisce lo scheletro concettuale su cui sono attaccati i diversi livelli corrispondenti alle varie lingue. WN-comune ricalca, nella sostanza, WordNet inglese tranne che in alcune peculiarità.[1] I livelli monolingue hanno una certa indipendenza; come già detto, essa è totale nella rappresentazione delle relazioni lessicali della lingua considerata. Per quanto riguarda le relazioni semantiche esse vengono ereditate dalla gerarchia comune se compatibili, altrimenti vengono espresse autonomamente, ma mantenendo comunque una corrispondenza fra i nodi della gerarchia monolingue e i nodi di WN-comune. Tutto questo è schematizzato nella figura 2.2. Come si vede nella figura i livelli monolingue sono costituiti da due parti:

- la parte corrispondente alla gerarchia WN-comune;
- la parte differente dalla gerarchia WN-comune.

La prima parte contiene solo synset e puntatori alla gerarchia comune. La seconda contiene un numero maggiore di informazioni perché è indipendente dalla gerarchia comune (questa parte viene anche detta WN-differenza).

2.2 EuroWordNet

Lo scopo di EuroWordNet è lo stesso di MultiWordNet, quanto cambia è l'approccio. In EuroWordNet non è presente alcun tipo di gerarchia comune, ma

[1] Quando WordNet inglese si comporta in maniera diversa dalla maggior parte delle altre lingue si è scelto di inserire nella gerarchia comune il comportamento più diffuso. Questo avviene tuttavia in poche circostanze.

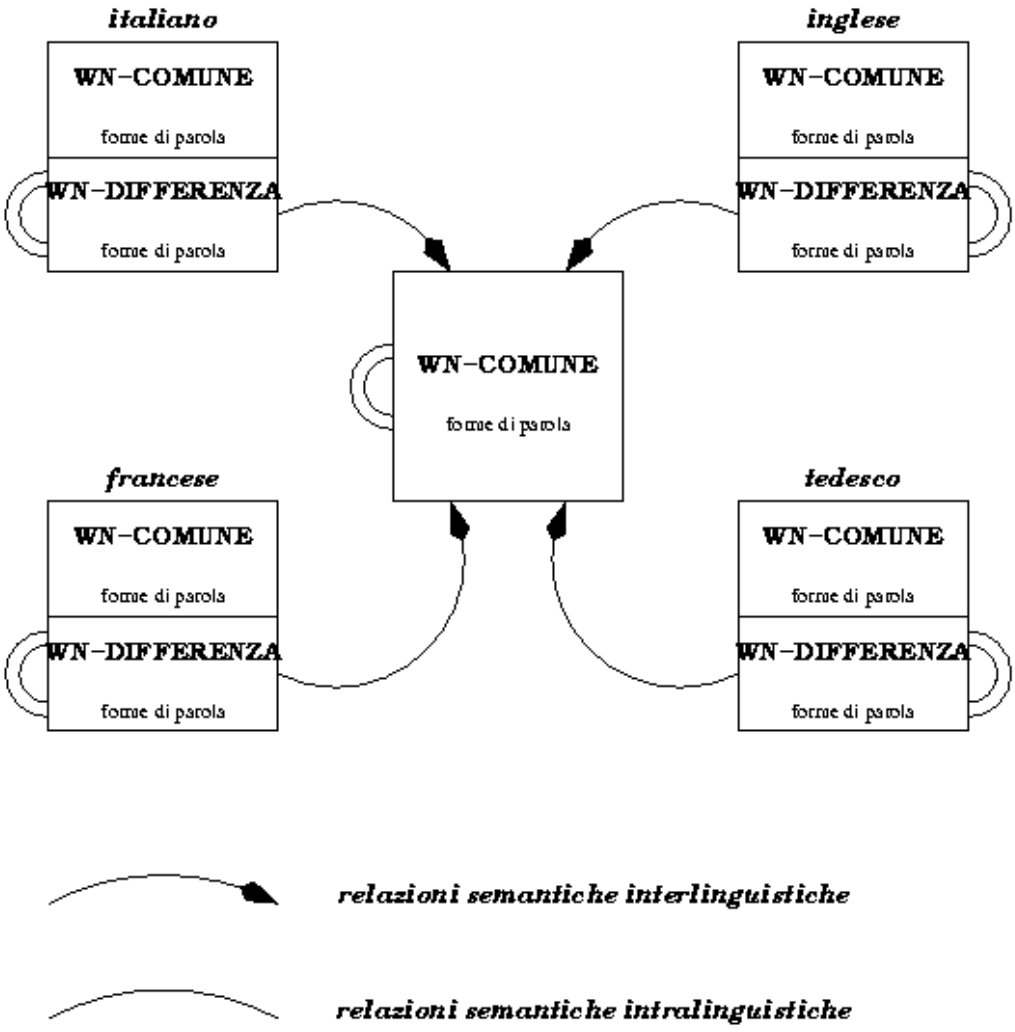

Figura 2.2: Gerarchia comune e gerarchie monolingue

le singole reti monolingue sono costruite indipendentemente l'una dall'altra e trattate come sistemi autonomi riferiti ad un'unica lingua.

Un vantaggio indubbio è la maggior autonomia delle singole lingue che perdono ogni dipendenza da WordNet inglese e sono quindi libere di contenere le proprie peculiarità. Lo svantaggio è la maggior difficoltà di integrazione dei diversi sistemi che non sono più legati per mezzo di puntatori ad una struttura o ad uno scheletro comune.

In EuroWordNet sono presenti due tipologie di moduli: quelli dipendenti da una specifica lingua e quelli indipendenti dalla lingua. Per quanto riguarda i primi essi costituiscono un insieme di synset riferiti ad una lingua specifica e ricalcano, sostanzialmente, la struttura di WordNet inglese con qualche piccola innovazione.[2] I moduli indipendenti dalla lingua consistono di tre elementi:

1. Inter Lingual Index (ILI);
2. Top Concept Ontology (TCO);
3. Domain Ontology (DO).

L'ILI è l'insieme di tutti i concetti e significati presenti nei vari moduli relativi alle diverse lingue[3]. Questa interfaccia permette di collegare tutte le reti monolingue perché ogni synset di tali reti deve avere almeno un puntatore a un record dell'ILI e i synset delle diverse lingue collegati allo stesso

[2]Sono state aggiunte due relazioni riferite ai casi più delicati di sinonimia (Near-Synonymy) e antonimia (Near-Antonymy) e due nuovi tipi di meronimia (porzione/massa e posto/area), cfr. 1.4.3.

[3]Le lingue rappresentate in EuroWordNet sono l'inglese, l'olandese, il tedesco, lo spagnolo, il francese, l'italiano, il ceco e l'estone.

record sono considerati equivalenti. Il suo ruolo è diverso da quello svolto da WN-comune in MultiWordNet in quanto quest'ultimo conteneva solo una parte dei synset del database, quelli in comune con il primo strato (costituito dal WordNet inglese), mentre l'ILI ha il ruolo di fornire un collegamento generale a tutti i synset del database. Quando è possibile i record dell'ILI sono collegati a record della TCO che è una gerarchia contenente un quantitativo piuttosto ristretto di significati indipendenti dallo specifico linguaggio e di notevole rilevanza. Tali significati sono in totale 63 e sono definizioni basilari (come 'oggetto' e 'sostanza', 'dinamico' e 'statico') cui sono collegati 1024 records dell'ILI. Lo scopo della TCO è generare una categorizzazione a grandi livelli dell'insieme di tutti i synset. Infine la DO è pure una gerarchia che suddivide i significati per campo semantico e non per classificazione relativamente alle categorie lessicali individuando diversi domini semantici.

Con EuroWordNet è evidente l'obiettivo di precisare meglio le relazioni già presenti in WordNet e di specificare nuove particolarità. Oltre alle aggiunte di cui sopra sono state infatti studiate delle relazioni fra le diverse parti del discorso[4] che in WordNet sono organizzate in gerarchie diverse fra loro non collegate. Inoltre sono stati aggiunti attributi allo scopo di specificare meglio le relazioni esistenti.[5] Un lavoro specifico è infine stato fatto sulle relazioni che collegano le reti monolingue all'interfaccia ILI. La maggior au-

[4]Tali relazioni sono: XPOS-Near-Synonymy; XPOS-Near-Antonymy; Has-XPOS-Hypernym e Has-XPOS-Hyponym; Role e Involved; Causes e Is-Caused-By; Has-Subevent e Is-Subevent-Of; Be-In-State e Is-State_Of.

[5]Gli attributi sono i seguenti: congiunzione o disgiunzione di relazioni multiple dello stesso tipo collegate ad un synset; intenzionalità o meno della relazione causale; reversibilità della relazione; negazione (esprime esplicitamente che una relazione tra due nodi non sussiste).

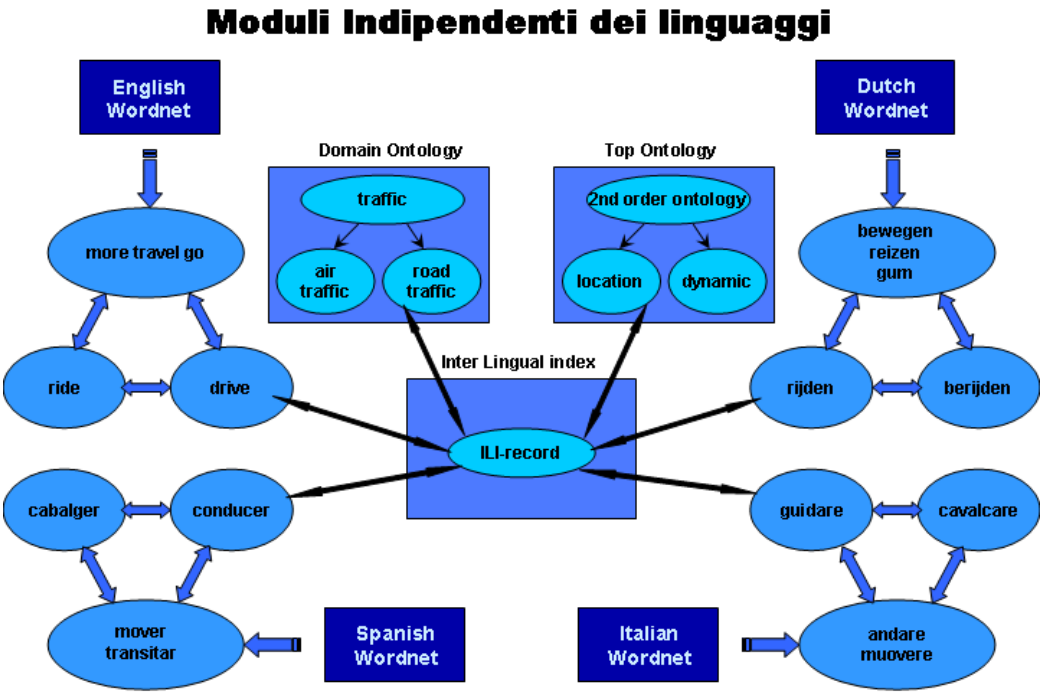

Figura 2.3: Lo schema di EuroWordNet

tonomia di EuroWordNet da WordNet consente un ulteriore differenza molto importante rispetto a WordNet stesso. Il primo vuole infatti riflettere solo la rappresentazione lessicale e le relazioni fra le parole di una lingua, per questo non sono presenti, a differenza di WordNet, nodi artificiali inseriti allo scopo di migliorare la classificazione e l'organizzazione di una gerarchia.

Una notevole spinta è stata data dalla natura assai modulare di EuroWordNet allo sviluppo di reti specifiche relative a determinati ambiti tematici. In quest'ottica è stato sviluppato in Italia un database sull'economia ispirato a WordNet e anche questo progetto in ambito matematico vuole seguire le stesse linee guida.

Capitolo 3

Alla base di un database matematico

Ciò che noi vogliamo realizzare è un database di termini matematici che si può inquadrare sotto la costituzione di EuroWordNet in quanto esso costituirà un modulo a sé stante eventualmente interfacciabile (per mezzo degli strumenti visti nella sezione 2.2) con altri database, primo fra tutti ItalWordNet. Come visto in tale sezione la costituzione del dizionario matematico è indipendente e deve unicamente soddisfare alle proprietà generali che caratterizzano WordNet.

Per quanto detto sopra la realizzazione della base di dati che caratterizzi il linguaggio della matematica si basa sull'applicazione dei principi di WordNet, visti nel capitolo 1, alle proprietà che intercorrono fra i vari concetti matematici. Questa fase è fondamentale e si concretizza nella realizzazione dei source file per il nostro dizionario che contengono tutti i termini considerati e le interconnessioni fra essi.

Il passo successivo, che vedremo nel capitolo 4, consiste nella scrittura dei programmi che costruiscono fisicamente le strutture dati e traducono le informazioni contenute nei source file in una forma che le renda memorizzabili

nelle suddette strutture. Tali programmi sono denominati 'grind programs'.

L'ultima fase del processo consiste nella composizione dei programmi che si occupano delle diverse possibili ricerche all'interno del dizionario e della stampa a video dei conseguenti risultati.

Una volta realizzato il secondo passo è possibile arricchire il database semplicemente scrivendo nuovi source file e facendoli elaborare dai programmi 'grind'.

3.1 I source file: sintassi

Alla base del dizionario stanno, come detto, i source file. La loro costituzione permette di comunicare tutti i dati necessari alla base di dati.

Come abbiamo visto nel paragrafo 1.6.1 essi sono costituiti da stringhe che contengono essenzialmente tre componenti:

- un identificativo;
- una serie di puntatori;
- eventualmente una glossa.

Per quanto riguarda i puntatori, componenti fondamentali perché tessono le fila della gerarchia, essi riflettono in tutto e per tutto le direttive standard presenti nel WordNet Reference Manual e riportate nel paragrafo 1.6.1.

3.2 I source file: come costruirli

Anzichè sulla struttura di questi file che già abbiamo esaminato ci concentriamo quindi sui concetti che portano alla loro formazione. La costruzione dei

source file è molto importante perché deve essere corretta, cioè rispettare le implicazioni relative ai puntatori in modo da costruire una gerarchia corretta e significativa all'interno della quale ci si possa muovere alla ricerca di un significato. La creazione di questi file è quindi alla base del successo del nostro dizionario e, una volta scritti i programmi di manipolazione e ricerca, diviene l'unica operazione che permette di allargare o migliorare il nostro sistema di memorizzazione e ricerca di dati.

Per capire meglio come avviene questa operazione consideriamo un esempio concreto e cioè la realizzazione del source file *noun.math.funzione* che contiene la gerarchia associata al termine matematico 'funzione' con delle ipotesi aggiuntive e cioè che la funzione sia definita in un intervalo aperto e limitato. Riportiamo nel listato 3.1 il listato del file in cui separiamo i diversi synset con una riga vuota per maggiore chiarezza; inoltre abbiamo rimosso tutte le glosse per accorciare la lunghezza dei synset che abbiamo dovuto, per evidenti motivi tipografici, disporre su più righe.

Listato 3.1: Source file noun.math.funzione

```
1 {funzione_in_un_intervallo_aperto_e_limitato
     funzione_continua,~ funzione_limitata,~
     funzione_integrabile,~ funzione_monotona,~
     funzione_periodica,~}
2
3 {funzione_continua funzione_in_un_intervallo_
     aperto_e_limitato,@ funzione_derivabile,~
     funzione_continua_e_limitata,~ funzione_
     continua_e_integrabile,~}
4
5 {funzione_limitata funzione_in_un_intervallo_
     aperto_e_limitato,@ funzione_continua_e_
     limitata,~}
6
7 {funzione_integrabile funzione_in_un_intervallo_
```

```
    aperto_e_limitato,@ funzione_continua_e_
    integrabile,~}
8
9 {funzione_monotona funzione_in_un_intervallo_
    aperto_e_limitato,@ funzione_strettamente_
    monotona,~ funzione_non_strettamente_monotona
    ,~}
10
11 {funzione_periodica funzione_in_un_intervallo_
    aperto_e_limitato,@ funzione_trigonometrica,~}
12
13 {funzione_derivabile funzione_continua,@ funzione
    _di_classe_Cn,~ funzione_primitiva,~}
14
15 {funzione_continua_e_limitata funzione_continua,@
     funzione_limitata,@ funzione_costante,~}
16
17 {funzione_continua_e_integrabile funzione_
    continua,@ funzione_integrabile,@ funzione_
    lineare,~}
18
19 {funzione_strettamente_monotona funzione_monotona
    ,@ funzione_strettamente_crescente,~ funzione_
    strettamente_decrescente,~}
20
21 {funzione_non_strettamente_monotona funzione_
    monotona,@ funzione_crescente,~ funzione_
    decrescente,~}
22
23 {funzione_trigonometrica funzione_periodica,@}
24
25 {funzione_di_classe_Cn funzione_derivabile,@
    funzione_parabola,~}
26
27 {funzione_primitiva funzione_derivabile,@}
28
29 {funzione_costante funzione_continua_e_limitata,@
    }
30
31 {funzione_lineare funzione_continua_e_integrabile
    ,@}
32
```

```
33 {funzione_strettamente_crescente funzione_
     strettamente_monotona,@ funzione_invertibile
     ,~}
34
35 {funzione_strettamente_decrescente funzione_
     strettamente_monotona,@ funzione_invertibile
     ,~}
36
37 {funzione_crescente funzione_non_strettamente_
     monotona,@ funzione_a_gradini_crescente,~}
38
39 {funzione_decrescente funzione_non_strettamente_
     monotona,@ funzione_a_gradini_decrescente,~}
40
41 {funzione_parabola funzione_di_classe_Cn,@}
42
43 {funzione_invertibile funzione_strettamente_
     crescente,@ funzione_strettamente_decrescente,
     @ funzione_esponenziale,~ funzione_logaritmo
     ,~}
44
45 {funzione_a_gradini_crescente funzione_crescente,
     @}
46
47 {funzione_a_gradini_decrescente funzione_
     decrescente,@}
48
49 {funzione_esponenziale funzione_invertibile,@}
50
51 {funzione_logaritmo funzione_invertibile,@}
```

Ci siamo posti in questo caso specifico per approfondirlo cercando di capire come costruire i puntatori, è però ovvio che una funzione definita in un intervallo aperto e limitato va in qualche modo collocata in una gerarchia più ampia e questo è fatto mediante il file *noun.math.relazione* di seguito riportato nel listato 3.2.

La gerarchia relativa a questo file è rappresentata, per maggior chiarezza, in figura 3.1.

Listato 3.2: Source file noun.math.relazione

```
1  {relazione funzione,~}
2
3  {funzione relazione,@ funzione_a_un_valore,~
       funzione_a_più_valori,~}
4
5  {funzione_a_un_valore funzione,@ funzione_in_un_
       intervallo_chiuso_e_limitato,~ funzione_in_un_
       intervallo_aperto_e_limitato,~ funzione_in_un_
       intervallo_aperto_e_illimitato,~}
6
7  {funzione_a_più_valori funzione,@}
8
9  {funzione_in_un_intervallo_chiuso_e_limitato
       funzione_a_un_valore,~}
10
11 {funzione_in_un_intervallo_aperto_e_limitato
       funzione_a_un_valore,~}
12
13 {funzione_in_un_intervallo_aperto_e_illimitato
       funzione_a_un_valore,~}
```

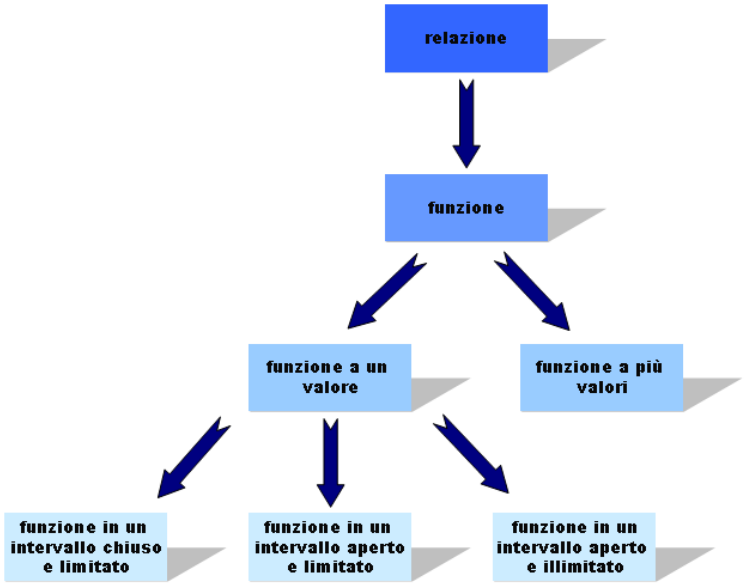

Figura 3.1: Gerarchia del significato 'relazione'

Come prima cosa notiamo che la struttura dei source file è assolutamente aderente ai principi di WordNet in quanto riflette esattamente quanto presentato nella sezione 1.6.1. Concentriamoci ora sulla gerarchia che si deduce dal file *noun.math.funzione*. La possiamo vedere chiaramente nelle figure[1] 3.2 e 3.3. La correttezza di questa gerarchia è fondamentale e si basa totalmente sul rispetto delle relazioni semantiche presenti fra i vari significati. In questo caso è presente soltanto la relazione semantica di iponimia/iperonimia e la sua applicazione è diretta conseguenza delle relazioni matematiche presenti fra gli oggetti rappresentati nel source file.

Consideriamo, osservando ad esempio la figura 3.2, la diramazione radice-foglia seguente:

[1]La figura è stata spezzata in due parti per renderla più leggibile

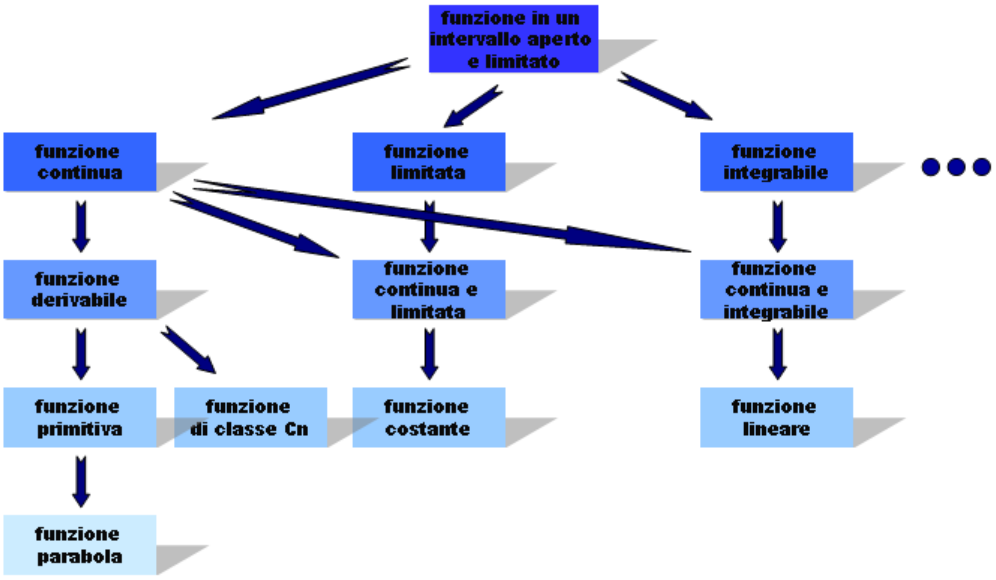

Figura 3.2: Gerarchia del significato 'funzione' (prima parte)

funzione in un intervallo aperto e limitato

$\Rightarrow$ funzione continua

$\Rightarrow$ funzione derivabile

$\Rightarrow$ funzione di classe C^n

$\Rightarrow$ funzione parabola

Si noti che è stato scelto di terminare, quando possibile, ogni diramazione con una foglia che costituisca un esempio della funzione presente allo stadio precedente. In questo caso la funzione parabola è un esempio di funzione di classe Cn.

Proviamo a percorrere la diramazione per verificarne la correttezza. La

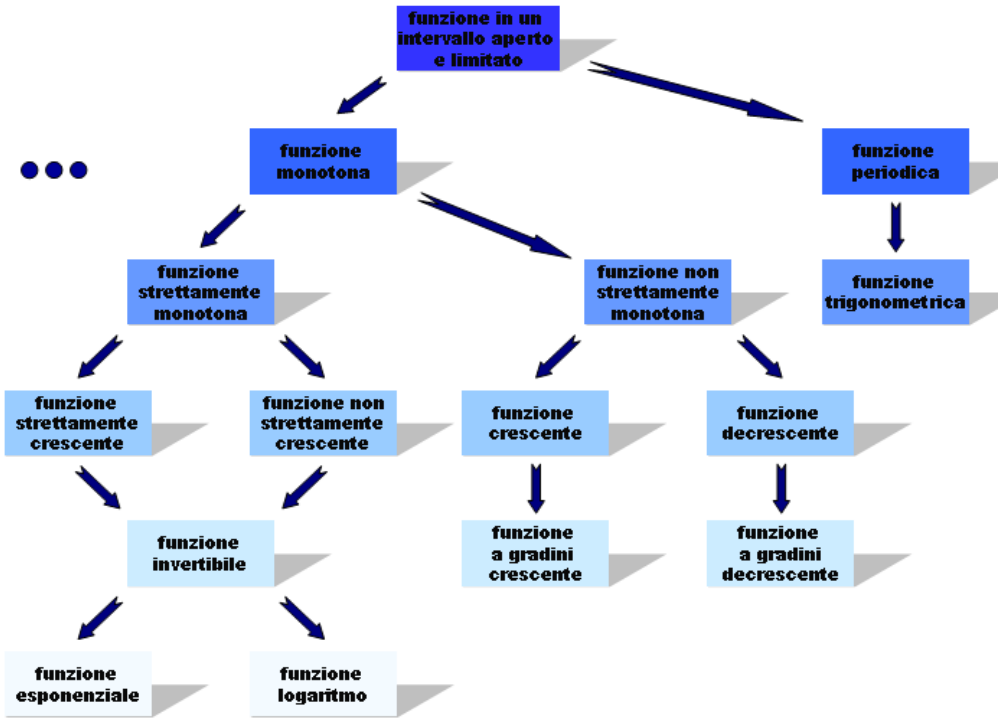

Figura 3.3: Gerarchia del significato 'funzione' (seconda parte)

radice è costituita dal significato di funzione con le ipotesi che essa sia definita in un intervallo aperto e limitato; questa ipotesi è essenziale in quanto essa permette di applicare dei teoremi che altrimenti non potremmo utilizzare. Per scendere di un gradino nel nostro albero dobbiamo aggiungere qualche proprietà in modo da ottenere qualcosa di più specifico, un iponimo. In questa diramazione aggiungiamo la continuità. La nostra funzione definita in un intervallo aperto e limitato può infatti essere continua o meno, ci mettiamo in un caso specifico considerandola continua. Fin qui è tutto piuttosto semplice, un passo importante viene compiuto nel successivo abbassamento di livello. Vogliamo infatti scendere ancor di più nello specifico e lo possiamo fare grazie ad una importante relazione matematica che è espressa per mezzo di un teorema:

Teorema 1 *Se una funzione è derivabile in un punto x_0, essa è anche continua in x_0.*

Questo teorema ci permette di affermare che una funzione derivabile in un intervallo aperto e limitato è in esso continua, non vale il viceversa. Ma quello ora definito è esattamente un rapporto di specializzazione/generalizzazione ovvero funzione derivabile è iponimo di funzione continua. Questo ci consente di considerare corretta l'associazione rappresentata nella gerarchia.

Scendendo ulteriormente nell'albero troviamo il significato funzione di classe C^n. Rispolveriamo la definizione di funzione di classe C^n:

Definizione 4 *Una funzione si dice di classe C^n in un intervallo aperto e limitato $]a,b[$ se è ivi derivabile n volte e se le sue derivate fino a quella di ordine n sono continue in $]a,b[$.*

Grazie a questa definizione possiamo dire che una funzione di classe C^n con n>=1 è derivabile mentre non vale il viceversa; di conseguenza il rapporto di iponimia è verificato.

Infine possiamo affermare che la funzione 'parabola' è una funzione continua e infinitamente derivabile con derivate tutte continue e perciò è un esempio (o una specializzazione) di funzione di classe C^n.

La realizzazione di un database matematico è quindi frutto della conoscenza matematica e dell'utilizzo dei teoremi al fine di determinare le relazioni di iponimia/iperonimia fra i vari concetti. Il risultato è una gerarchia consistente che permette di navigare in maniera corretta fra i diversi oggetti della matematica.

Capitolo 4

Un database matematico

La creazione di un database si basa su tre componenti base:

- una struttura dati in cui immettere tutte le informazioni;
- un insieme di programmi che si occupa di inserire queste informazioni;
- un insieme di programmi che si occupa di ricercare e rendere leggibili le informazioni di cui necessita l'utente.

In questo capitolo ci occupiamo di come il database matematico viene creato, andremo quindi a osservare come vengono realizzate le prime due componenti.

4.1 La struttura dati

Tutte le informazioni contenute nei source file devono essere trasferite e organizzate in una struttura dati che permetta di navigare fra le informazioni

in maniera efficiente. Essa è organizzata in diversi livelli incapsulati uno nell'altro ed è realizzata per mezzo di diversi 'header file'[1].

4.1.1 Il puntatore

Una prima entità di cui abbiamo bisogno è il puntatore. Come abbiamo visto WordNet si basa sulla capacità di mettere in relazione diversi significati in maniera tale da costituire una gerarchia (nel capitolo 3 abbiamo visto ad esempio la relazione di iponimia fra *funzione derivabile* e *funzione continua*). Sappiamo che nel source file relativo a un significato una relazione è rappresentata per mezzo di un identificativo di un altro significato seguito da una virgola e da un simbolo che identifica la tipologia della relazione. Le informazioni che dobbiamo memorizzare, per ogni significato, sono quindi due:

1. l'identificativo del valore semantico puntato;

2. il tipo di relazione che lega i due significati.

Tale struttura è definita come una classe puntatore in un header file puntatore.h, il codice C++ è riportato nel listato 4.1. La rappresentazione grafica della struttura dati in cui memorizzare le informazioni di cui sopra è riportata in figura 4.1. In essa:

1. **forma** rappresenta l'identificativo del significato ed è una stringa;

[1]Un header file è un file che, nei linguaggi di programmazione, è atto a contenere le strutture dati e i programmi che vi permetto no l'accesso. Il suo nome (file di intestazione) è dovuto al fatto che un tale file viene richiamato nell'intestazione di programmi che utilizzano le strutture dati in esso definite.

Listato 4.1: Classe puntatore

```
1  class puntatore
2  {
3  private:
4
5          string forma_di_parola;
6
7          string simbolo_puntatore;
8
9  public:
10
11         // inserisce un puntatore
12         void inserzione(string forma, string simbolo);
13
14         // estrae il simbolo di un puntatore
15         string estrai_simbolo();
16
17         // estrae la forma di parola puntata
18         string estrai_puntatore();
19 };
```

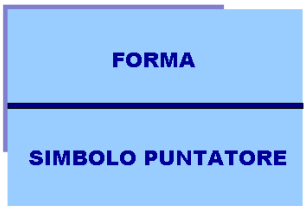

Figura 4.1: Struttura dati di un puntatore

2. **simbolo puntatore** è il simbolo che caratterizza il tipo di relazione secondo gli standard WordNet ed è pure una stringa perché vari simboli sono costituiti da più caratteri (se la relazione di iperonimia è caratterizzata dal simbolo @, la relazione di olonimia membro/insieme è identificata da #m, vedi tabelle 1.5(avverbi), 1.6(aggettivi), 1.7(verbi), 1.8(nomi)).

4.1.2 Il synset

Saliamo di un livello: ogni riga di un source file rappresenta un synset, quindi ogni riga deve essere memorizzata in un'opportuna struttura dati che rappresenterà l'entità synset. Ancora una volta ci chiediamo quali informazioni dobbiamo memorizzare; per come è fatto un synset sappiamo che esso contiene:

1. l'identificativo del valore semantico che il synset vuole caratterizzare;

2. i puntatori ai significati con cui tale valore semantico è in relazione secondo i parametri di WordNet;

3. eventualmente una glossa che contiene una definizione.

Tale struttura è definita come una classe synset in un file header synset.h, il codice C++ è riportato nel listato 4.2. In figura 4.2 è rappresentata graficamente la struttura dati di cui sopra. In essa:

1. **forma di parola** rappresenta l'identificativo del significato descritto dal synset ed è una stringa;

Listato 4.2: Classe synset

```
class synset
{
private:

        string forma_di_parola;

        list<puntatore> p;

        string glossa;

public:

        // inserisce una forma di parola nel synset
        void insert_forma(string forma);

        // inserisce un puntatore nel synset
        void insert_puntatore(string forma, string simbolo
           );

        // inserisce una glossa nel synset
        void insert_glossa(string definizione);

        // restituisce la forma di parola presente nel
           synset
        string estrai_forma();

        // restituisce il puntatore presente nel synset
        list<puntatore> estrai_puntatore();

        // restituisce la glossa presente nel synset
        string estrai_glossa();
};
```

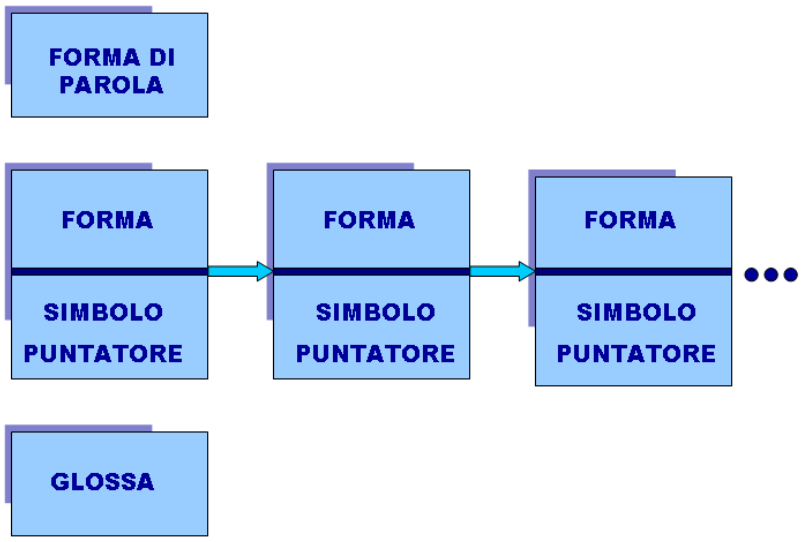

Figura 4.2: Struttura dati di un synset

2. **forma e simbolo** rappresentano un puntatore, come vediamo nella struttura è presente una lista di strutture puntatore, tale lista contiene tutte le relazioni relative al synset;

3. **glossa** rappresenta la definizione del significato ed è anch'essa una stringa.

È importante osservare che i puntatori, che sono parte fondamentale del synset, sono memorizzati in una lista. Questa scelta permette vantaggi e svantaggi. Il vantaggio principale è la facilità con cui questa struttura può crescere e di conseguenza un eventuale aggiornamento della base di dati (con inserimento di nuovi significati e, conseguentemente, nuove relazioni) sarà assorbito senza problemi. Lo svantaggio è una maggior lentezza in caso di ricerca di uno specifico puntatore che non è comunque così grande (il C++

mette a disposizione liste 'doppiamente linkate'[2] che permettono di muoversi molto agevolmente fra i diversi elementi) e inoltre l'operazione di cui parliamo non è fra le più frequenti.

4.1.3 Una tabella hash di synset

WordNet altro non è che una collezione di synset e il nostro dizionario matematico, sviluppato con gli stessi principi, è esso pure una collezione di synset. Una scelta decisiva per la realizzazione è come organizzare questa raccolta di synset. Si è scelto qui come in WordNet la struttura tabella hash.

Con le tabelle hash possiamo fare in modo che le operazioni sui dizionari abbiano un costo medio $O(1)$ (quindi costante), indipendentemente dalle dimensioni sia del dizionario, sia dell'insieme universale da cui si attingono gli elementi del dizionario. In figura 4.3 è riportata la struttura di una tabella hash con B posizioni (esse vengono denominate 'bucket'). Supponiamo di dover inserire nella nostra struttura un elemento x di cui non conosciamo la natura. Tale elemento viene passato come argomento ad una funzione, denominata **funzione hash**, che produce un valore intero compreso fra 0 e B-1 dove B indica il numero di *bucket* (letteralmente contenitori) della tabella hash. Indichiamo tale valore intero con *h(x)*, esso definisce il bucket in cui l'elemento x sarà inserito. In sostanza i bucket costituiscono gruppi di elementi e la funzione hash viene usata per decidere a quale di questi gruppi un elemento appartiene. Quale sia la funzione hash più opportuna dipende dagli elementi da inserire, la scelta fatta nel nostro caso sarà discussa nella

[2]Con questo si intende che due elementi consecutivi della lista hanno ognuno un puntatore in direzione dell'altro. In conseguenza di ciò ci si può muovere in entrambe le direzioni di percorrenza della lista.

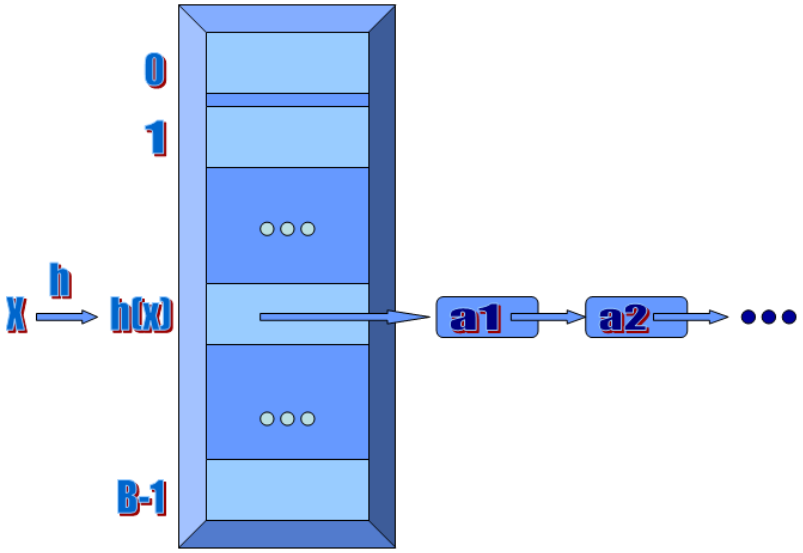

Figura 4.3: Tabella hash

paragrafo 4.2.2. La cosa importante è che la funzione hash mescoli in modo casuale gli elementi, in modo che ogni bucket tenda a contenerne circa lo stesso numero, indipendentemente dalla struttura più o meno casuale di tali elementi. Se questo non avviene la struttura in esame diviene molto meno efficiente. Ogni bucket è costituito da una lista in cui vengono memorizzati tutti gli elementi che la struttura hash assegna a quel bucket. Una scelta importante è quella del valore B caratterizzante il numero di bucket. Esso dipende dal numero di elementi e va scelto abbastanza grande per mantenere contenute le dimensioni delle liste. In caso contrario salirebbe, e di molto, il costo di ogni ricerca.

Da un punto di vista fisico viene realizzato un vettore di dimensione prefissata DIM pari al numero di bucket. Ogni elemento di questo vettore è

Listato 4.3: Classe db

```
1  class db
2  {
3  private:
4
5          ls tabella[DIM];
6
7  public:
8
9          // inserisce un synset nella tabella hash (bucket
               b)
10         int insert_synset(synset s, int b);
11
12         // estrae un synset dalla tabella hash (bucket b)
13         ls get_lista(int b);
14
15 };
```

una lista di synset. Per inserire un elemento si deve specificare il bucket in cui esso verrà inserito, tale valore viene naturalmente calcolato dalla funzione hash. Tale struttura è definita come una classe db in un file header db.h, il codice C++ è riportato nel listato 4.3. Il vettore definito contiene elementi di tipo **ls**; **ls** è un'altra classe che definisce semplicemente una lista di synset con le normali funzioni di inserimento ed estrazione, il codice ad essa relativo non è riportato perché non contiene elementi di interesse specifici.

4.2 I programmi

Questi programmi sono denominati 'grind' e il loro scopo è di inserire i dati contenuti nei source file in un formato database che faciliti la ricerca e il recupero delle informazioni, ovvero nella struttura dati che è stata esposta nella sezione 4.1. In pratica all'avvio del dizionario matematico tutti i source file presenti vengono elaborati con questi programmi al fine di costruire il database sul quale verranno effettuate le ricerche.

Questi programmi agiscono in due fasi:

- fase 1: 'parsing' dei source file e costruzione synset;
- fase 2: inserimento synset.

La prima fase consiste nel parsing dei source file. Fare il parsing significa fare la scansione di una stringa estraendo da essa le informazioni contenute ed eliminando ciò che non serve. Sappiamo che i source file sono testuali e quindi le stringhe che costituiscono ogni riga vengono scandite per estrarre le informazioni in esse rappresentate. Durante questa prima fase è buona cosa controllare che i source file siano realizzati correttamente, cioè secondo le direttive di WordNet esplicitate nella sezione 1.6. Nel caso vengano riscontrati errori, che possono essere concettuali, ma anche di semplice battitura, si possono realizzare dei programmi che provvedano, quando sono errori di scrittura, alla correzione degli stessi. Questa parte, evidenziata nel diagramma di flusso rappresentato in figura 4.4, non è stata realizzata nel presente lavoro di tesi. Come si può vedere nella stessa figura, una volta superato il test si può procedere alla creazione dei synset.

Da un punto di vista pratico si susseguono nella fase precedente la creazione dei synset le seguenti operazioni:

1. *apertura* del source file;
2. *estrazione* dal file di ogni *riga* e inserimento della stessa in un vettore di stringhe;
3. scansione del vettore di stringhe e *elaborazione* di ogni posizione per mezzo di una funzione elaborazione_righe.

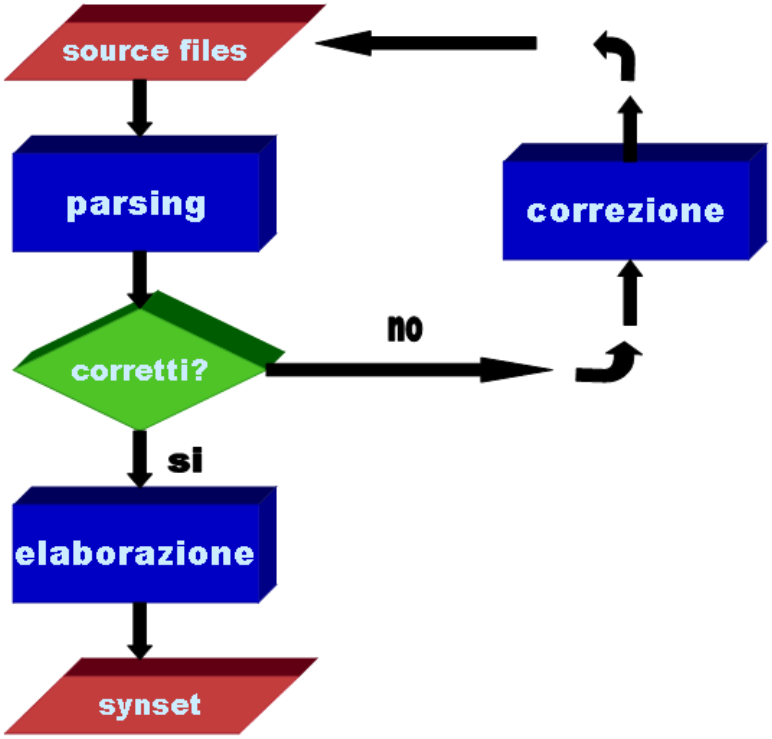

Figura 4.4: Elaborazione dei source file

Dopo questi passaggi (*apertura*, *estrazione*, *elaborazione*) abbiamo trasferito tutti i dati in una struttura (definita in 4.1.2) atta a ospitare il synset e si dovrà procedere all'inserimento dello stesso nella tabella hash.

4.2.1 Elaborazione di una riga

L'elaborazione di una riga è svolta dalla funzione **elaborazione_righe** che riceve come parametro una stringa con la riga corrente e restituisce l'identificativo della riga, il synset corrispondente e eventualmente una serie di sinonimi. Nel listato 4.4 è riportata la dichiarazione della funzione in esame, vediamo nel dettaglio cosa rappresentano i diversi parametri.

Listato 4.4: Dichiarazione della funzione elaborazione_righe

```
1 string elaborazione_righe(string riga, synset *s, vector<
    string> *sinonimi);
```

L'unico parametro di input è la stringa riga che contiene la riga corrente, quindi una delle righe estratte dal source file e quindi nella forma studiata nella sezione 1.6. Sappiamo, in virtù dell'organizzazione dei source file suddetta, che, dopo la prima parentesi graffa, è presente una forma di parola che costituisce l'identificativo del synset, eventualmente seguita da sinonimi. Tale forma di parola viene immediatamente estratta dalla funzione e viene restituita come parametro di ritorno della stessa, ad essa è infatti riferito il tipo di ritorno *string*. Nella dichiarazione è inoltre presente un puntatore a una struttura di tipo synset (descritta nella sezione 4.1.2), questo perché la funzione **elaborazione_righe** si occupa di estrarre tutte le informazioni presenti nel synset (identificativo, come già visto, puntatori ed, eventualmente, la glossa) e di inserirle in una struttura ad hoc restituendo il puntatore ad essa. Infine nel prototipo è presente un vettore di stringhe

denominato ‘sinonimi’. È opportuno fermarci un attimo a riflettere sul ruolo di questo elemento.

La domanda è la seguente: come gestisco i sinonimi? Una prima risposta può essere: creare un synset distinto per ogni sinonimo. Il problema legato a questa soluzione è la ridondanza con cui i dati verrebbero memorizzati. Infatti verrebbero creati più synset fra loro differenti solo per l’identificativo, con uguali puntatori e stessa glossa (se presente). Questa soluzione non è ovviamente ottimale perché comporta il salvataggio di informazioni ripetute. Per questo ordine di motivi si è scelta una strada diversa: la creazione di un synset fittizio. Quando ci si trova ad elaborare una stringa in cui sono presenti più termini sinonimi si procede alla creazione del synset vero e proprio (chiamiamolo *originale*) per il primo termine, utilizzato come identificativo. Per le restanti forme di parola con lo stesso significato si crea un synset fittizio in cui l’identificativo contiene la forma di parola ed è presente un puntatore al synset *originale* in cui sono memorizzate le informazioni pertinenti ad ogni forma di parola. Così facendo abbiamo un unico posto fisico in cui sono contenuti i dati e uno o più puntatori che associano ai sinonimi tali dati (vedi figura 4.5). In essa viene esaminato il caso dei termini sinonimi ‘inconsueto’ e ‘insolito’. Per ‘insolito’ viene creato il synset vero e proprio mentre per ‘inconsueto’ si crea un synset fittizio con identificativo corrispondente alla stessa forma di parola e un puntatore a ‘insolito’ che contiene come simbolo la stringa *sinonimo*. La funzione **elaborazione_righe** non si occupa di realizzare tutto questo, ma soltanto di restituire un vettore di stringhe contenente i diversi sinonimi del primo identificativo della riga, qualora fossero presenti. In caso contratio viene restituito un vettore vuoto.

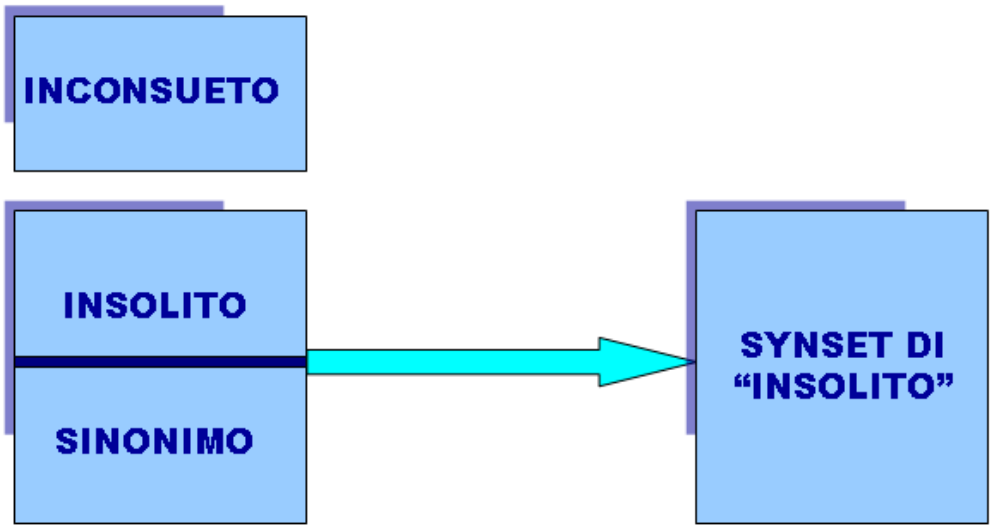

Figura 4.5: Gestione dei sinonimi

I sinonimi devono comunque essere memorizzati nella struttura perché essi hanno un ruolo importante anche nella comunicazione delle informazioni all'utente. Infatti, per riflettere la grande importanza dei synset come struttura e il ruolo della relazione di sinonimia all'interno del database, WordNet risponde alle ricerche dell'utente fornendo, prima ancora della definizione del termine cercato (fra l'altro non sempre presente), la lista dei sinonimi. Questo è importante perché permette di focalizzare subito il concetto. I sinonimi a un significato vanno quindi memorizzati e, per uniformità con la struttura descritta finora, essi vengono inseriti fra i puntatori mettendo nel campo simbolo la stringa 'sin'. Questo è corretto anche da un punto di vista concettuale perché la sinonimia è una relazione come le altre e come tale viene trattata.

4.2.2 La funzione hash

Una volta inseriti i dati nella struttura synset si tratta di memorizzare tale struttura nella tabella hash, il primo passo per fare questo è calcolare la posizione in cui deve avvenire l'inserimento e questo è fatto per mezzo della **funzione hash** presentata nel listato 4.5.

Listato 4.5: Funzione hash

```
1  int fhash(string str, int dimensione)
2  {
3          int valore=0;
4
5          // scansione della stringa carattere per carattere
6          for(int i=0;i<str.size();i++)
7          {
8                  // accumulo in valore la "somma dei
                      caratteri"
9                  valore=valore+(int)str[i];
10         }
11
12         // ritorno il resto di valore/dimensione
13         return (valore%dimensione);
14 }
```

Come abbiamo detto nel paragrafo 4.1.3 la scelta della funzione hash è molto importante perché essa deve essere tale che gli oggetti da memorizzare siano equamente distribuiti fra i diversi bucket. Noi abbiamo basato la scelta della posizione in cui inserire un synset sull'identificativo dello stesso, e, essendo questo una stringa, abbiamo deciso di scegliere la funzione hash più diffusa che opera su stringhe. Tale funzione riceve due parametri:

1. la stringa **str** contenente l'identificativo;

2. l'intero **dimensione** contenente il numero di bucket della tabella hash.

Essa inizializza a zero un contatore (la variabile accumulatore **valore**) dopodichè scorre la stringa str carattere per carattere sommando nel contatore il valore

numerico di ogni carattere. A questo punto, per ottenere un valore compreso fra 0 e il numero di bucket meno 1 la funzione calcola il resto dell'operazione di divisione fra l'accumulatore e il numero di bucket.

Essendo gli identificativi dell'intera base di dati piuttosto vari da un punto di vista alfabetico riteniamo che questa sia una buona scelta, cioè che la funzione hash soddisfi lo scopo di dividere i diversi synset in maniera equa fra i bucket.

4.2.3 Inserimento di un synset

Nota la posizione si può procedere all'inserimento del synset nella tabella hash. Questo avviene con la chiamata riportata nel listato 4.6.

Listato 4.6: Inserimento di un synset nella tabella hash

```
1                wn.insert_synset(s, posizione);
```

In essa wn rappresenta la tabella hash, quindi è una struttura del tipo db (definito nella sezione 4.1.3). Viene chiamata la funzione associata a tale classe **insert_synset** che procede all'effettivo inserimento ricevendo come parametri:

1. il synset **s** da inserire nella base di dati;
2. l'intero **posizione** contenente il numero del bucket della tabella hash in cui il synset deve essere inserito.

L'inserimento può essere fatto in diversi modi:

- casuale;
- in ordine alfabetico (basandosi sull'identificativo);

- ....

Scegliere un inserimento ordintato (la scelta di un ordinamento alfabetico è la più ovvia) comporta una maggior pesantezza dell'operazione di inserimento, ma una velocità di ricerca superiore. Nel nostro caso è stata fatta la scelta di non procedere ad un inserimento casuale perché la volontà è di strutturare la tabella hash in modo da ottenere, in corrispondenza ai vari bucket, liste piuttosto corte e quindi ottenere effettivamente prestazioni dell'ordine di $O(1)$ per quanto riguarda la ricerca. Questo rende non necessario fare un inserimento ordinato.

Capitolo 5

Ricerca e presentazione delle informazioni

Non meno importante della conoscenza delle informazioni è la loro ricerca e presentazione. È infatti fondamentale creare dei programmi che si muovano in maniera intelligente fra le informazioni memorizzate nella struttura dati presentata nel precedente capitolo e presentino con altrettanta efficacia quanto da loro reperito all'utente.

Quando l'utente desidera cercare delle informazioni inserisce anzitutto una o più parole che descrivono l'oggetto della sua ricerca. Una volta incamerate tali informazioni prodotte dall'utente, il sistema avvia una serie di operazioni che dovranno produrre un output che soddisfi la richiesta. Tali operazioni, schematizzate nelle figure 5.1 e 5.2, sono elencate di seguito:

1. eliminazione dalla stringa di input di informazioni ridondanti o non necessarie (prima fase della manipolazione);

2. riduzione di quanto ottenuto al punto precedente ad una stringa coe-

rente con il formato delle informazioni memorizzate nel database (seconda fase della manipolazione);

3. ricerca della stringa ottenuta all'interno del database, chiamiamola ricerca primaria;

4. ricerca delle informazioni a tale stringa collegate in base ai desideri dell'utente (vedi paragrafo 5.2.3), chiamiamola ricerca secondaria.

Notiamo dalla figura 5.2 che tutte le informazioni raccolte vengono elaborate. Questa fase, che ha lo scopo di produrre un output chiaro, verrà esaminata nella sezione 5.3.

5.1 Manipolazione della richiesta fornita dall'utente

La primissima fase di esame e modifica delle stringhe di input può prevedere l'eliminazione di termini non significativi quali gli articoli. Questa fase è delicata e lo stesso WordNet non la implementa; se infatti andiamo a cercare il termine *the book* non otteniamo alcuna risposta e non, come ci si potrebbe attendere, informazioni relative a *book*. Spesso si preferisce segnalare all'utente di inserire solamente le parole significative omettendo ad esempio gli articoli. La maggior parte dei sistemi di ricerca su basi di dati riguardanti libri prevede che, volendo cercare il testo 'La teoria unificata dei segnali', si inserisca la stringa 'teoria unificata segnali'.

Al di là di quanto detto fin'ora dobbiamo ricordare sempre che l'utente, alla ricerca di informazioni riguardanti un argomento, è solito inserire uno

Figura 5.1: Ricerca delle informazioni (prima parte)

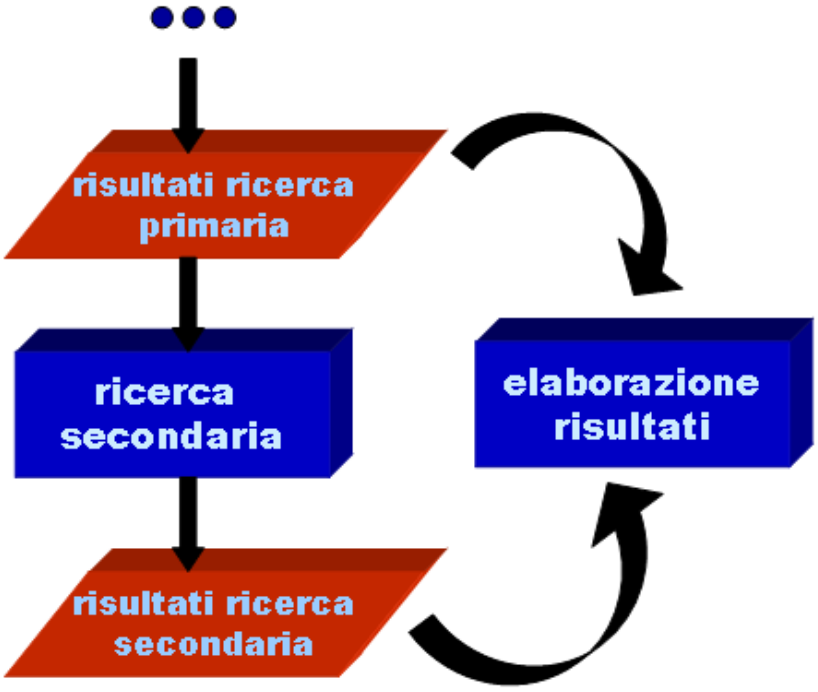

Figura 5.2: Ricerca delle informazioni (seconda parte)

o più termini che identificano, a suo parere in modo significativo, quanto lui sta cercando. I dati inseriti si discostano notevolmente dal formato con cui vengono memorizzate le informazioni nella base di dati per due ordini di motivi:

- dato un significato vi sono molte parole distinte che ad esso si riferiscono, ma si differenziano meramente per ragioni grammaticali;
- ogni database specifico adotta regole e formati che non sono quelli in uso nella lingua italiana.

In riferimento al primo punto si utilizzano dei cosiddetti programmi di 'lemmatizzazione'[1]. Questi non sono stati realizzati nel presente lavoro di tesi, ma sono oggetto di studio specifico e si sono raggiunti degli ottimi risultati in questo campo. Il loro scopo è quello, ad esempio, di ricavare da parole plurali la radice. Nel dizionario matematico oggetto di questo lavoro viene memorizzata la parola *funzione*, ma non il suo plurale *funzioni*, tuttavia si ritiene che l'utente alla ricerca della parola *funzioni* riceva come risposta tutte le informazioni associate al termine *funzione*. Allo stesso modo se in WordNet 2.0 andiamo a svolgere la ricerca della parola *brothers* otteniamo in risposta tutte le informazioni riguardanti *brother*. Come detto i programmi di lemmatizzazione sono vari e, in qualche modo, fra loro distinti. L'eventuale scelta di un tale programma deve essere accurata e funzionale alle caratteristiche che si pretendono da questa prima manipolazione delle stringhe di input.

[1]La lemmatizzazione è 'quel complesso di operazioni che conducono a riunire tutte le forme sotto il rispettivo lemma', intendendo per *lemma* 'ciascuna parola titolo o parola chiave di un dizionario' e per forma 'ogni possibile diversa realizzazione grafica di un lemma'. La lemmatizzazione, quindi, 'consiste nell'attribuire le varianti o flessive(*uomini*) o grafiche(*omo*) a una stessa parola(*uomo*), che funge da lemma'. Marco Passarotti

Per quanto riguarda il secondo punto dobbiamo ricordare che le stringhe inserite nel database riflettono fedelmente la struttura dei source file. Per questo una forma di parola composta da più termini viene memorizzata con gli stessi collegati da *underscore*, ricordiamo infatti che la parola *funzione continua* compare nel dizionario come *funzione_ continua*. È evidente come, anche qui, si renda necessaria una manipolazione al fine di ridurre quanto inserito dall'utente ad una forma coerente con quella adottata all'interno del sistema in esame. Se, ad esempio, un utente inserisce la stringa *funzioni continue* dovranno essere ricavati i singolari di entrambe le parole e, successivamente, questi andranno collegati con un underscore (vedi figura 5.3).

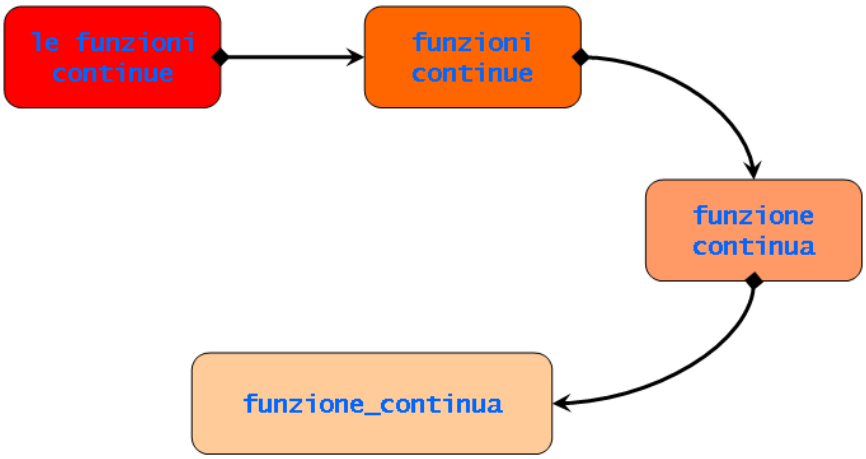

Figura 5.3: Manipolazione delle stringhe di input

5.2 Ricerca dell'informazione

Nel momento in cui si può ritenere conclusa la manipolazione dell'input fornito dall'utente si può procedere alla ricerca della stringa che tale manipolazione ha prodotto. Quello che vogliamo è individuare il synset che ha per identificativo la parola cercata.

È doveroso precisare che in una base di dati di grandi dimensioni qual è WordNet non è possibile svolgere l'operazione di grind e caricare in memoria tutti i source file in quanto ciò risulterebbe un'elaborazione troppo pesante. Tale modo di procedere è invece più consono a una base di dati 'giocattolo', come il dizionario matematico in esame, in quanto non influisce negativamente sul modo di rintracciare le informazioni.

5.2.1 Organizzazione della ricerca in WordNet

Per capire come funziona la ricerca in WordNet dobbiamo anzitutto vedere come è organizzato il database[2], cioè come sono memorizzati i dati dopo che i source file sono stati elaborati. Esplicitiamo che quanto segue è una trattazione volutamente sommaria in quanto si vuole trasmettere al lettore un'idea di come i dati sono organizzati, per questo si è scelto di non specificare in maniera formale l'esatta disposizione dei file in seguito introdotti.

Per ogni categoria sintattica vengono creati due file: *index.pos* e *data.pos* dove *pos* rappresenta, come abbiamo visto nel paragrafo 1.6.1, la categoria sintattica (può assumere i valori *noun*, *verb*, *adj* e *adv*). I file *data.pos* con-

[2]È interessante notare come i file del database siano in formato ASCII, ciò permette un facile accesso ad essi anche con applicazioni proprie da parte dell'utente.

tengono tutte le informazioni che erano presenti nei source file con i puntatori tradotti in indirizzi che individuano posizioni all'interno degli stessi file.

L'organizzazione della ricerca è basata sulla presenza degli *index file*. Ogni *index file* inizia con informazioni sul copyright, sul numero della versione, sulla licenza. A seguire vi sono i dati. Ogni linea di dati contiene le seguenti informazioni:

- la forma di parola cui si riferiscono i dati a seguire;
- l'indice di familiarità, che abbiamo visto essere un indicatore di significato[3] nella sezione 1.6.1, della forma di parola in esame;
- una lista dei tipi di puntatori relazionali usati in tutti i synset contenenti tale forma di parola[4];
- una lista di indici che costituiscono i byte di offset[5] nel file di dati (source file) corrispondente, ogni indice si riferisce all'apparizione della forma di parola in un synset diverso.

Quando si deve cercare una forma di parola si accede anzitutto agli *index file* alla ricerca della stessa e, una volta trovata, la riga ad essa corrispondente si hanno a disposizione gli indirizzi di tutti i synset che contengono quella forma di parola all'interno dei *data file*.

Ma quali informazioni contengono i *data file*? In sostanza possiamo affermare che essi contengono tutto quanto riguarda i synset descritti nei source

[3]Infatti è chiamato 'sense count'.

[4]Tale lista è utilizzata dal sistema di ricerca per informare l'utente su tutte le ricerche possibili a partire da quella forma di parola, vedi sezione A.

[5]Con questo si intende la distanza in byte che separa la presenza della forma di parola dall'inizio del file.

file e in particolare un file *data.pos* presenta diverse righe iniziali contenenti, come per gli *index file*, informazioni sul copyright, sul numero della versione, sulla licenza; a seguire è presente una lista di tutti i nomi dei file di input che sono stati elaborati dal *grinder*[6], in ordine di elaborazione; infine sono presenti i dati veri e propri organizzati, ancora una volta, a righe. Ognuna di tali righe contiene:

- l'indirizzo, inteso come byte di offset, del synset cui la riga si riferisce[7];
- un codice numerico che individua il source file in cui il synset è contenuto;
- il tipo di synset (*noun*, *verb*, *adjective*, *adjective satellite*, *adverb*);
- il numero di forme di parola presenti nel synset;
- l'indice di familiarità con il significato definito precedentemente per gli *index file*;
- il numero di puntatori presenti nel synset;
- i puntatori.

A loro volta i puntatori sono rappresentati da quattro informazioni:

- il simbolo del puntatore;
- l'indirizzo, sempre come byte di offset, del synset obiettivo all'interno del file *data.pos* che lo contiene;

[6]La parola *grinder* è equivalente a programmi grind, vedi sezione 4.2.

[7]Questa informazione è ridondante, in quanto è nota al momento in cui si è giunti a quella riga all'interno del file, ma viene inserita per agevolare alcuni programmi UNIX quali il comando *grep*.

- il file *data.pos* che contiene il synset obiettivo;

- un campo che indica, fra l'altro, se siamo in presenza di un puntatore lessicale o semantico (vedi paragrafo 1.4).

A questo punto è abbastanza ovvio come avvenga la ricerca in WordNet. Ottenuta la forma di parola da cercare, dopo la fase iniziale di manipolazione della richiesta dell'utente, si scorrono gli *index file* alla ricerca della stessa e, una volta trovata la riga corrispondente, si vanno ad aprire i *data file* di interesse e si recuperano i diversi synset che contengono quella forma di parola. Ad ogni synset sappiamo corrispondere un differente significato e si può dire quindi di aver recuperato tutti i possibili significati della forma di parola di interesse per l'utente. A questo punto l'utente potrà essere interessato, per ogni diverso significato, a ricerche più approfondite. Questa tematica verrà affrontata nel paragrafo 5.2.3 mentre nella sezione 5.3 vedremo come avviene la stampa a video dei risultati.

5.2.2 Organizzazione della ricerca nel dizionario matematico

Nel nostro dizionario matematico tutto è più semplice. Essendo infatti piccolo il numero dei source file prodotti è possibile, in quanto non troppo oneroso, procedere al caricamento degli stessi (cioè allo svolgimento dei programmi di grind su di essi), prima di iniziare la ricerca. Al termine di questa prima fase tutti i synset sono stati caricati, e quindi inseriti, nella tavola hash e la ricerca può iniziare.

La procedura che si occupa di reperire le informazioni nel database (chiamata **ricerca**), visualizzata nel listato 5.1, richiede tre parametri:

- **wn**, il puntatore al database, cioè alla tavola hash contenente tutti i dati (vedi sezione 4.1);
- **posizione**, il bucket all'interno della tavola hash in cui cercare la forma di parola;
- **parola**, la forma di parola di interesse.

La variabile **posizione** è un intero ottenuto calcolando la funzione hash sulla stringa **parola** e può oviamente essere calcolato anche all'interno della procedura. L'algoritmo applicato è piuttosto semplice ed esegue le seguenti operazioni:

1. estrazione del synset di interesse (quello con identificativo **parola**) dalla lista di synset presente nel bucket individuato da **posizione**;
2. estrazione da tale synset di identificativo, puntatori e, se presente, della glossa.

La ricerca semplice si può considerare terminata in quanto all'utente vengono fornite tutte le informazioni di base riguardanti la forma di parola di interesse. Ovviamente è presente nel dizionario, e quindi viene estratto, un synset per ogni diverso significato associato alla forma di parola in esame.

Listato 5.1: Ricerca di una forma di parola nella tavola hash

```
1 /*
2  * funzione ricerca
3  *
4  * ricerca le informazioni nel database riguardanti
5  * la forma di parola "parola" e le stampa
```

```
6   *
7   * si accede al database con (*wn).
8   */
9   void ricerca(db *wn, int posizione, string parola)
10  {
11          //variabili
12          ls lista_synset;
13          //lista dei synset in cui cercare
14          synset  supporto;
15          //synset di supporto
16          string simbolo;
17          //simbolo puntatore
18          int i = 1;
19          //contatore
20          list<puntatore> puntatori;
21          //lista dei puntatori
22
23          //estraggo il synset cercato e lo inserisco in
                supporto
24          lista_synset = (*wn).get_lista(posizione);
25          supporto = lista_synset.get_synset(parola);
26
27          /*****  stampa1: chiave *****/
28          cout << "\n\n\n Parola cercata:\n\n";
29          cout << "-----> chiave: " << supporto.estrai_forma
                () << "\n";
30
31          //estraggo la lista dei puntatori
32          puntatori = supporto.estrai_puntatore();
33          //scorro la lista dei puntatori e stampo a video
34          //definisco iteratori
35          list<puntatore>::iterator iter = puntatori.begin()
                ;
36          list<puntatore>::iterator iter_end = puntatori.end
                ();
37          //scansione lista
38          for(;iter != iter_end; ++iter)
39          {
40                  /*****  stampa2: sinonimi       *****/
41                  //cout<< (*iter).estrai_simbolo()
                        << " ---> "<<(*iter).estrai_puntatore
                       ()<<'\n';
42                  simbolo = (*iter).estrai_simbolo();
43                  if (simbolo == "sin")
44                  {
45                          cout<< "Sinonimo " << i << " --->
                               "<<(*iter).estrai_puntatore()<<
                               '\n';
46                          i = i+1;
47                  }
48          }
49
```

```
50          /*****  stampa3: glossa *****/
51          cout << "-----> definizione: " << supporto.
                estrai_glossa() << "\n";
52
53  }
```

5.2.3 Diverse tipologie di ricerca

La ricerca di base ha lo scopo di elencare tutti i significati associati a una determinata forma di parola. Se, ad esempio, cerchiamo la parola *chair* in WordNet otteniamo la lista di tutti i significati associati (quindi i synset che contengono tale parola). A partire da questo primo risultato si possono tuttavia iniziare molti approfondimenti.

Si possono svolgere diverse ricerche a seconda della categoria lessicale cui appartiene la forma di parola in esame. Supponiamo di avere a che fare con un nome (continuiamo a pensare all'esempio suddetto della parola *chair*). Come si può vedere nell'appendice A si possono svolgere essenzialmente sette diversi tipi di ricerche distinte:

- sinonimi;
- termini coordinati;
- iperonimi;
- iponimi;
- olonimi;
- meronimi;
- indice di familiarità.

Consideriamo qui, come esempio, la ricerca degli iponimi e degli iperonimi e vediamo come è svolta nel nostro dizionario matematico. Abbiamo scelto queste due relazioni perché permettono di strutturare i risultati come gerarchie e quindi una maggior facilità d'esame dell'output che può essere presentato come una vera e propria gerarchia. Prendiamo in esame la parola *funzione_continua*. Dopo aver svolto una normale ricerca supponiamo di volere cercare tutti gli iponimi della forma di parola in esame. Viene così richiamata la funzione **iponimi** riportata nel listato 5.2. Essa presenta la stessa lista di parametri vista per la funzione **ricerca**, esaminata nel paragrafo 5.2.2, con l'eccezione di un intero **i** che precedentemente non era presente e che individua il passo della ricorsione. La sua presenza è resa necessaria per motivi di stampa.

Listato 5.2: Ricerca degli iponimi

```
1  /*
2   * funzione iponimi
3   *
4   * ricerca le informazioni nel database e le stampa
5   * creando una gerarchia di meronimi partendo dalla
6   * parola cercata
7   *
8   * si accede al database con (*wn).
9   */
10 void iponimi(db *wn, int posizione, string parola, int i)
11 {
12         //variabili
13         ls lista_synset;
14         //lista dei synset in cui cercare
15         synset supporto;
16         //synset di supporto
17         string simbolo;
18         //simbolo puntatore
19         string punta;
20         //stringa puntatore
21         string visual1 = "---> ", visual2 = "-----";
22         //stringhe per la visualizzazione
23         int numero;
24         //valore funzione hash
25         list<puntatore> puntatori;
```

```
26          //lista dei puntatori
27          vector<string> *livello_successivo = new vector<
                string>;
28          //identificativi livello successivo nella
                gerarchia
29
30          //estraggo il synset cercato e lo inserisco in
                supporto
31          lista_synset = (*wn).get_lista(posizione);
32          supporto = lista_synset.get_synset(parola);
33
34          /*****  stampa1: chiave *****/
35          //identazione grafica
36          for (int j=0; j!=i; j++)
37                  visual1 = visual2 + visual1;
38          //stampa
39          cout << visual1 << supporto.estrai_forma() << "\n"
                ;
40          if (i == 0)
41                  cout << "\n";
42          //passo della ricorsione
43          i = i + 1;
44
45          //estraggo la lista dei puntatori
46          puntatori = supporto.estrai_puntatore();
47          //scorro la lista dei puntatori e stampo a video
48          //definisco iteratori
49          list<puntatore>::iterator iter = puntatori.begin()
                ;
50          list<puntatore>::iterator iter_end = puntatori.end
                ();
51          //scansione lista
52          for(;iter != iter_end; ++iter)
53          {
54                  /*****  stampa2: scendiamo nella gerarchia
                            *****/
55                  //estrazione simbolo puntatore
56                  simbolo = (*iter).estrai_simbolo();
57                  if (simbolo == "~")
58                  {
59                          punta = (*iter).estrai_puntatore()
                                ;
60                          numero = fhash(punta, DIM);
61                          iponimi(wn, numero, punta, i);
62                  }
63          }
64  }
```

La procedura vista è ricorsiva. Infatti essa recupera la lista dei puntatori presenti nel synset di partenza per poi fare una scansione di tale lista e riap-

plicare se stessa (cioè la funzione **iponimi**) ad ogni puntatore. Così facendo la funzione scende la gerarchia (strutturata come abbiamo visto nel paragrafo 3.2) richiamando se stessa ad ogni livello per reperire le informazioni essenziali presenti a quel livello. Così facendo si riesce a ricostruire la parte di gerarchia che va dalla forma di parola in esame (che diviene la radice di questa parte di gerarchia) alle foglie che ad essa si possono ricondurre. In riferimento alla forma di parola *funzione_ continua* l'output ottenuto è illustrato in figura 5.4. Come si vede in figura viene percorso ogni ramo che

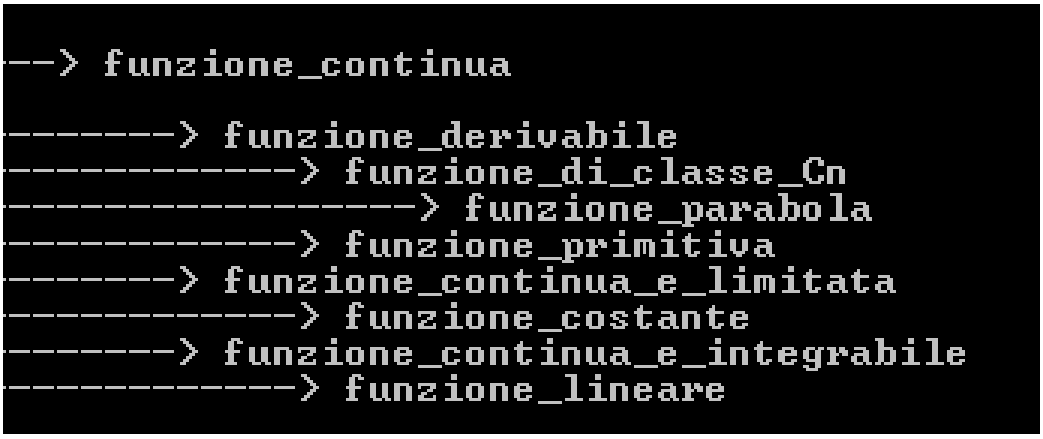

Figura 5.4: Output della ricerca degli iponimi per la forma di parola *funzione_ continua*

parte dalla forma di parola in esame e porta ad una foglia. Per esempio seguiamo la diramazione *funzione_ continua* –> *funzione_ derivabile* –> *funzione_ di_ classe_ Cn* –> *funzione_ parabola* e vediamo che essa ricostruisce in maniera corretta parte della gerarchia già esaminata nel paragrafo 3.2, cercando, a partire dal synset che descrive la funzione continua, gli iponimi ad essa associati e ricostruendo un albero con un maggior grado di specificità via via che si scendono i livelli.

Esaminiamo ora la ricerca degli iperonimi e la funzione **iperonimi** (riportata nel listato 5.3) che di tale ricerca si occupa. Osserviamo anzitutto che la procedura è strutturata in maniera molto simile a **iponimi** con un'identica lista di parametri e con un'organizzazione assolutamente speculare. L'unica cosa che cambia è la ricerca dei puntatori: in quest'ultima procedura si cercano quelli che rappresentano la relazione di iperonimia (quindi identificati dal simbolo @) anziché quelli associati alla relazione di iponimia (identificati dal simbolo ∼). Quello che ci aspettiamo è che tale procedura risalga la diramazione che porta dalla parola in esame alla radice della gerarchia che la contiene.[8]

Listato 5.3: Ricerca degli iperonimi

```
1  /*
2   * funzione iperonimi
3   *
4   * ricerca le informazioni nel database e le stampa
5   * creando una gerarchia di meronimi partendo dalla
6   * parola cercata
7   *
8   * si accede al database con (*wn).
9   */
10 void iperonimi(db *wn, int posizione, string parola, int i
       )
11 {
12         //variabili
13         ls lista_synset;
14         //lista dei synset in cui cercare
15         synset supporto;
16         //synset di supporto
17         string simbolo;
18         //simbolo puntatore
19         string punta;
20         //stringa puntatore
21         string visual1 = "---> ", visual2 = "-----";
22         //stringa per la visualizzazione
23         int numero;
24         //valore funzione hash
25         list<puntatore> puntatori;
```

[8] L'utilizzo dell'indice di familiarità permetterebbe, come già visto nel paragrafo 1.6.1, di eliminare quei nodi presenti nella diramazione meno significativi perché troppo specifici.

```
26          //lista dei puntatori
27          vector<string> *livello_successivo = new vector<
               string>;
28          //identificativi livello successivo nella
               gerarchia
29
30          //estraggo il synset cercato e lo inserisco in
               supporto
31          lista_synset = (*wn).get_lista(posizione);
32          supporto = lista_synset.get_synset(parola);
33
34          /*****  stampa1: chiave *****/
35          //identazione grafica
36          for (int j=0; j!=i; j++)
37                  visual1 = visual2 + visual1;
38          //stampa
39          cout << visual1 << supporto.estrai_forma() << "\n"
               ;
40          if (i == 0)
41                  cout << "\n";
42          //passo della ricorsione
43          i = i + 1;
44
45          //estraggo la lista dei puntatori
46          puntatori = supporto.estrai_puntatore();
47          //scorro la lista dei puntatori e stampo a video
48          //definisco iteratori
49          list<puntatore>::iterator iter = puntatori.begin()
               ;
50          list<puntatore>::iterator iter_end = puntatori.end
               ();
51          //scansione lista
52          for(;iter != iter_end; ++iter)
53          {
54                  /*****  stampa2: scendiamo nella gerarchia
                            *****/
55                  //estrazione simbolo puntatore
56                  simbolo = (*iter).estrai_simbolo();
57                  if (simbolo == "@")
58                  {
59                          punta = (*iter).estrai_puntatore()
                               ;
60                          numero = fhash(punta, DIM);
61                          iperonimi(wn, numero, punta, i);
62                  }
63          }
64  }
```

Andando ad esaminare l'output associato alla ricerca degli iperonimi in riferimento, ad esempio, alla forma di parola *funzione_ crescente* troviamo

quanto presentato in figura 5.5. Come si vede in figura viene effettiva-

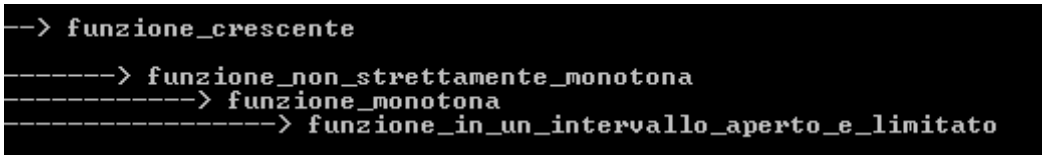

Figura 5.5: Output della ricerca degli iperonimi per la forma di parola *funzione_continua*

mente percorso dalla procedura il ramo che parte dalla forma di parola in esame e porta alla radice della gerarchia. Vediamo infatti che la diramazione *funzione_crescente –> funzione_non_strettamente_monotona –> funzione_monotona –> funzione_in_un_intervallo_aperto_e_limitato*[9] ricostruisce in maniera corretta il ramo di interesse presente nella gerarchia esaminata nel paragrafo 3.2, cercando, a partire dal synset che descrive la funzione crescente, gli iperonimi ad essa associati e ricostruendo un percorso con un maggior grado di generalità via via che si salgono i livelli.

In maniera analoga sono organizzate le procedure che si occupano delle altre ricerche possibili. Tali ricerche sono elencate nella tabella 5.6.

Riguardo alla tabella vogliamo precisare che:

- le sigle *m/i*, *m/o*, *c/o* specificano il tipo di relazione di olonimia o meronimia e stanno rispettivamente per *membro/insieme*, *materiale/oggetto*, *componente/oggetto*;

[9]Con la presenza dell'indice di familiarità si sarebbe magari potuto ricosrtuire, più semplicemente, il percorso *funzione_crescente –> funzione_monotona –> funzione_in_un_intervallo_aperto_e_limitato*

nomi	sinonimi
	iperonimi
	iponimi
	olonimi (m/i)
	olonimi (m/o)
	olonimi (c/o)
	meronimi (m/i)
	meronimi(m/o)
	meronimi(c/o)
	attributi
aggettivi	sinonimi
	nomi da cui è derivato
	antonimi
	participio del verbo
	scala di valori
verbi	sinonimi
	antonimi
	iperonimi
	iponimi
	troponimi
	gruppo verbale
avverbi	sinonimi
	antonimi
	aggettivoi da cui è derivato

Figura 5.6: Elenco delle possibili ricerche a partire da un synset

- *scala di valori* riferito agli aggettivi indica la scala presente fra gli opposti valori di due aggettivi antonimi (pensiamo a piccolo-grande) di cui l'aggettivo in esame fa parte;

- *troponimo* indica i modi verbali.

5.3 Stampa a video delle informazioni

Una volta raccolte le informazioni è importante comunicarle in modo chiaro all'utente. L'interfaccia grafica di WordNet, come vedremo nell'appendice A, è tanto essenziale quanto chiara. Possiamo dire che è piuttosto semplice presentare i risultati di una ricerca di base, in quanto si tratta di presentare un diverso significato per ogni synset associato alla forma di parola oggetto della ricerca. Per ogni significato si presentano l'identificativo, i sinonimi e l'eventuale glossa. Volendo, ma è un'informazione aggiuntiva di second'ordine, si può presentare la lista dei sinonimi. È buona cosa suddividere i significati a seconda della categoria lessicale cui appartengono (vedi sezione 1.5). Per esempio in risposta alla ricerca della parola *chair* WordNet 2.0 fornisce l'output illustrato in figura 5.7 (si è scelto di riportare solo la parte contenete l'elenco dei sinonimi, 'tagliando' le glosse, per motivi tipografici).

Diverso può essere il modo di presentare le ricerche specifiche che abbiamo presentato nel paragrafo 5.2.3. Quando la risposta alla ricerca è strutturabile come gerarchia è, a nostro parere, opportuno organizzare sempre l'output di conseguenza. Questo è stato fatto, ad esempio, nelle figure 5.4 e 5.5 che riportano risultati di ricerche effettuate all'interno del dizionario matematico. Negli altri casi i risultati si presentano solitamente come un elenco di

1. (35) **chair** -- (a seat for one person, with a support for the back; "he put his
2. (2) professorship, **chair** -- (the position of professor; "he was awarded an en
3. president, chairman, chairwoman, **chair**, chairperson -- (the officer who pres
   remarks to the chairperson")
4. electric chair, **chair**, death chair, hot seat -- (an instrument of execution by el
   sentenced to die in the chair")

The verb chair has 2 senses (no senses from tagged texts)

1. **chair**, chairman -- (act or preside as chair, as of an academic department in a
   years")
2. moderate, **chair**, lead -- (preside over; "John moderated the discussion")

Figura 5.7: Risposta di WordNet 2.0 alla ricerca della parola *chair*

significati. Pensiamo ad esempio di cercare i significati associati alla forma di parola *car* e di andare a svolgere un'ulteriore ricerca dei meronimi, quindi della 'parti di', in riferimento al significato più diffuso cioè quello di automobile. Il risultato è costituito da un insieme di parti di un'automobile, dal motore al volante, che si prestano ad essere presentati sotto forma di elenco come è evidenziato in figura 5.8 (ancora una volta è riportata una parte significativa dell'output a discapito delle glosse).

È molto importante dedicare del tempo alla produzione di un'interfaccia grafica che permetta all'utente di interagire in maniera efficace con il dizionario che intende utilizzare. In particolare la visualizzazione dei risultati non è assolutamente difficile da implementare e realizzare, ma è molto importante fare tutto ciò con attenzione perché solo presentando in modo chiaro ed efficace i risultati di una ricerca l'utente può sfruttare a pieno la bonta di un dizionario on-line e, più in generale, di un qualsiasi contenitore di informazioni.

car, auto, automobile, machine, motorcar -- (4-wheeled motor vehicle; usually propelled by an internal cc
car to get to work")
HAS PART: accelerator, accelerator pedal, gas pedal, gas, throttle, gun -- (a pedal that controls th
the gas")
HAS PART: air bag -- (a safety restraint in an automobile; the bag inflates on collision and prevent
being thrown forward)
HAS PART: auto accessory -- (an accessory for an automobile)
HAS PART: automobile engine -- (the engine that propels an automobile)
HAS PART: automobile horn, car horn, motor horn, horn, hooter -- (a device on an automobile fo
HAS PART: buffer, fender -- (a cushion-like device that reduces shock due to contact)
HAS PART: bumper -- (a mechanical device consisting of bars at either end of a vehicle to absorb
damage)
HAS PART: car door -- (the door of a car)
HAS PART: car mirror -- (a mirror that the driver of a car can use)
HAS PART: car seat -- (a seat in a car)
HAS PART: car window -- (a window in a car)
HAS PART: fender, wing -- (a barrier that surrounds the wheels of a vehicle to block splashing w
call a fender a wing")
HAS PART: first gear, first, low gear, low -- (the lowest forward gear ratio in the gear box of a m
moving)
HAS PART: floorboard -- (the floor of an automobile)
HAS PART: gasoline engine -- (an internal-combustion engine that burns gasoline; most automobil
engines)
HAS PART: glove compartment -- (compartment on the dashboard of a car)
HAS PART: grille, radiator grille -- (grating that admits cooling air to car's radiator)

Figura 5.8: Risposta di WordNet 2.0 alla ricerca dei meronimi della parola *car*

Conclusioni e sviluppi futuri

Come visto nei capitoli 3, 4 e 5 la realizzazione di un dizionario matematico si può suddividere in tre fasi:

- scrittura dei source file;
- creazione dei programmi grind;
- elaborazione dei programmi di ricerca e stampa delle informazioni.

Nel presente lavoro di tesi la prima fase è stata affrontata, ma non conclusa, in quanto creare un insieme esauriente di source file per la matematica italiana richiede un lavoro molto impegnativo da condurre preferibilmente a piu' mani. Sono stati realizzati alcuni source file di esempio per capire come è necessario procedere alla loro scrittura, e questo è un passaggio davvero importante; è stata inoltre realizzata una lista di termini, presenti nell'appendice B, da cui si possono creare le gerarchie e, di conseguenza, i source file che possono essere alla base di un dizionario completo.

La seconda e la terza fase sono state affrontate in maniera piuttosto esauriente. La loro implementazione è stata fatta mediante il linguaggio di programmazione C++. È stato pensato di realizzare un'interfaccia grafica

con utilizzo delle librerie wxWindows, interfaccia che può risultare davvero uno strumento molto utile in un'applicazione di questo tipo. La sua realizzazione è quindi uno degli sviluppi futuri di grande interesse. Crediamo che la difficoltà maggiore stia nello studio di un'interfaccia semplice, ma che permetta all'utente di accedere alle informazioni in modo chiaro e di svolgere senza difficoltà le sue ricerche. A questo proposito si dimostra molto comoda l'interfaccia di WordNet 2.0 che viene esaminata nell'appendice A.

L'altro sviluppo possibile e di grande interesse è quello, sopra citato, di realizzare un insieme esauriente di source file. Questa operazione porta con sé una conseguente modifica alla struttura della nostra base di dati. Infatti attualmente tutti i source file presenti vengono elaborati dai programmi grind, e questo è possibile perché la base di dati è di piccole dimensioni. Come è invece spiegato nella sezione 5.2.1 in riferimento a WordNet, per un database di grandi dimensioni è conveniente un'organizzazione diversa che permetta una ricerca preliminare che permetta di determinare i source file di interesse su cui eseguire i programmi grind. Questo passo è fondamentale e strettamente collegato alla realizzazione di un dizionario esauriente in ambito matematico.

Un ultimo compito che potrebbe essere svolto è un lavoro di aggiornamento sull'interfaccia ILI e di conseguenza sulla Top Concept Ontology e sulla Domain Ontology al fine di inserire il presente dizionario in ItalWordNet e quindi EuroWordNet (vedi sezione 2.2).

Appendice A

WordNet 2.0: una veloce guida all'uso

Vediamo brevemente come può essere utilizzato WordNet 2.0, l'ultima versione disponibile on-line della base di dati realizzata a Princeton.

WordNet possiede un'interfaccia grafica molto semplice, essenziale, ma non per questo poco efficace. In figura A.1 vediamo la schermata iniziale di WordNet una volta lanciato. In essa basta inserire la parola da cercare in 'Search Word' e premere 'Invio' per iniziare la ricerca.

Riprendiamo l'esempio fatto nel capitolo 1 e ipotizziamo di cercare la parola *dog*. Se è stata fatta una ricerca semplice, senza modificare le opzioni predefinite, otteniamo in risposta quanto visualizzato nella figura A.2. Vengono presentati in ordine i diversi significati divisi in blocchi ognuno associato ad una fra le diverse categorie (nomi, aggettivi, avverbi, verbi). Ovviamente se non esistono nel database significati associati ad una specifica categoria questi non vengono presentati. Nell'esempio vediamo come siano presenti 7 significati associati alla parola *dog*. Ogni significato è costituito dalla lista dei sinonimi seguita, se presente, dalla glossa fra parentesi tonde.

Una volta svolta la ricerca appaiono dei pulsanti associati alle quattro ca-

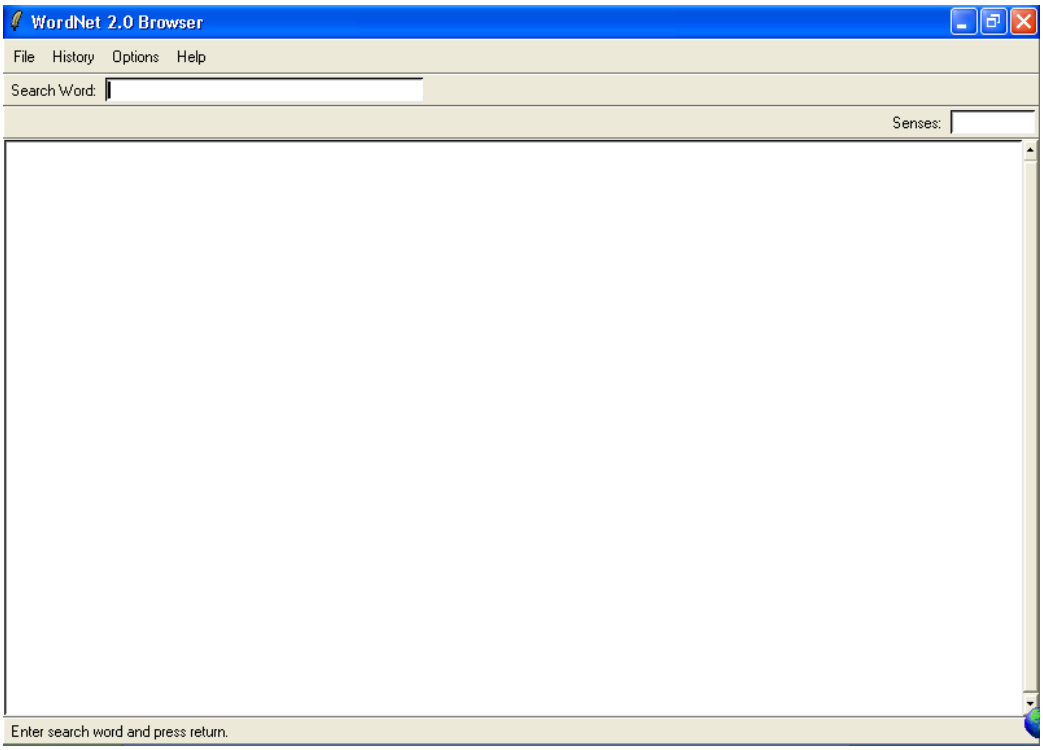

Figura A.1: Come si presenta WordNet all'apertura

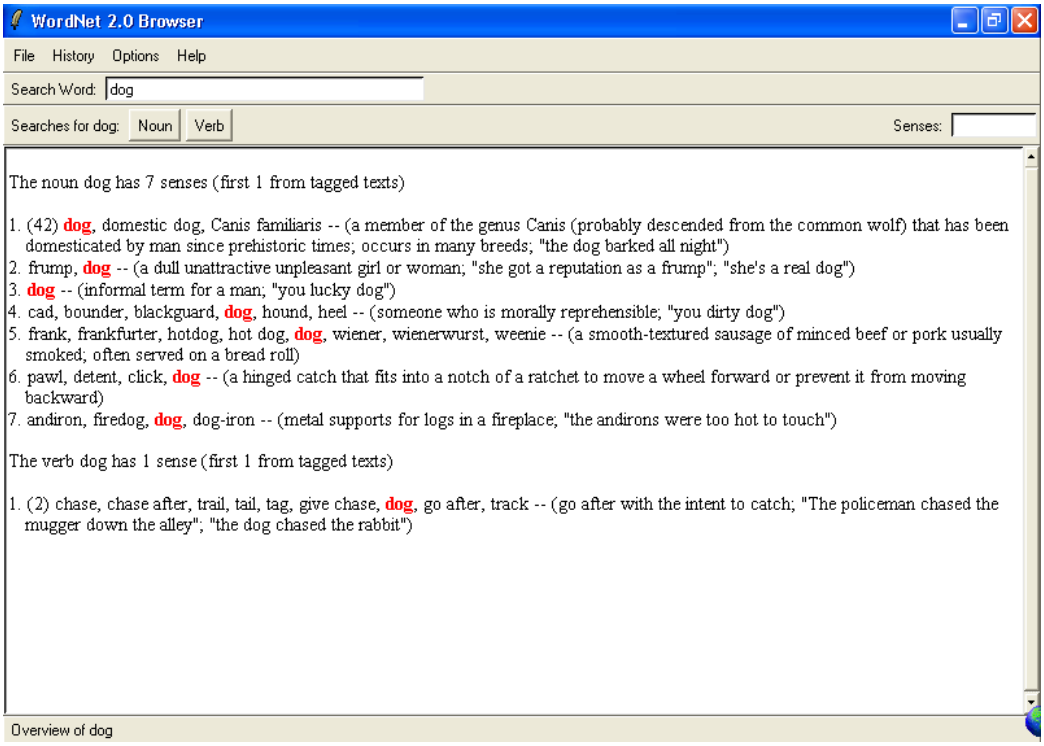

Figura A.2: Ricerca della parola *dog*

tegorie sintattiche che permettono di approfondire la stessa ottenendo ulteriori informazioni. Nelle figura A.3 vediamo, per esempio, le opzioni associate alla categoria nomi. Possiamo accedere ad ulteriori informazioni riguardanti:

Figura A.3: Ricerca ulteriore: opzioni associate ai nomi

- sinonimi;
- termini coordinati;
- iperonimi;
- iponimi;

- olonimi;
- meronimi;
- indice di familiarità.

Per chiarimenti riguardanti queste categorie rimandiamo alla sezione 1.4.

Nelle figure A.4, A.5, A.6 sono presentate le opzioni relative alle categorie verbi, aggettivi, avverbi.

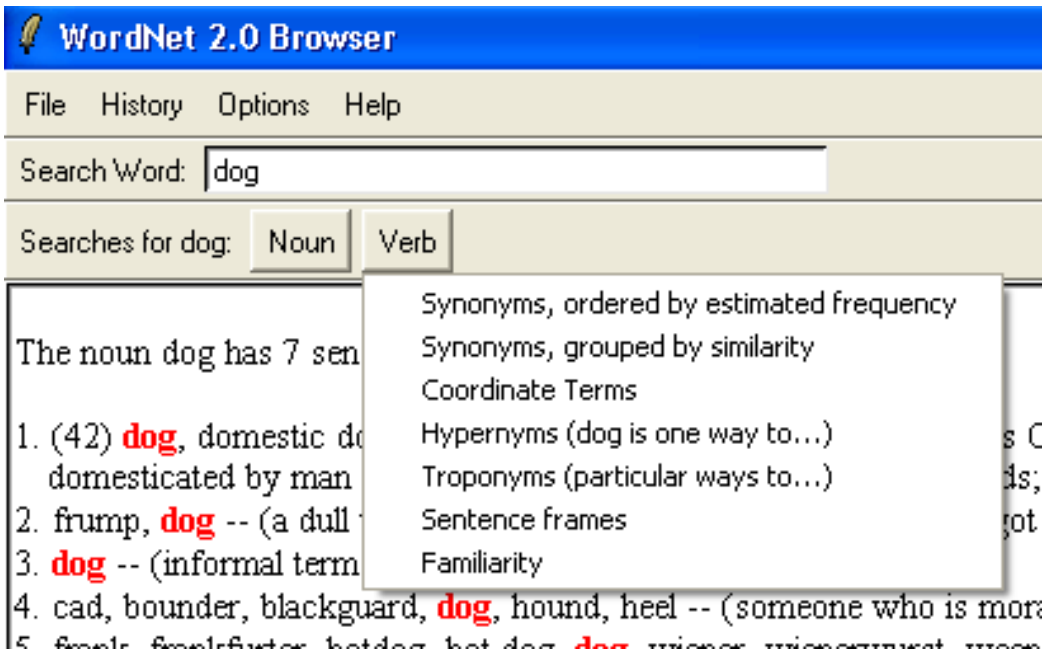

Figura A.4: Ricerca ulteriore: opzioni associate ai verbi

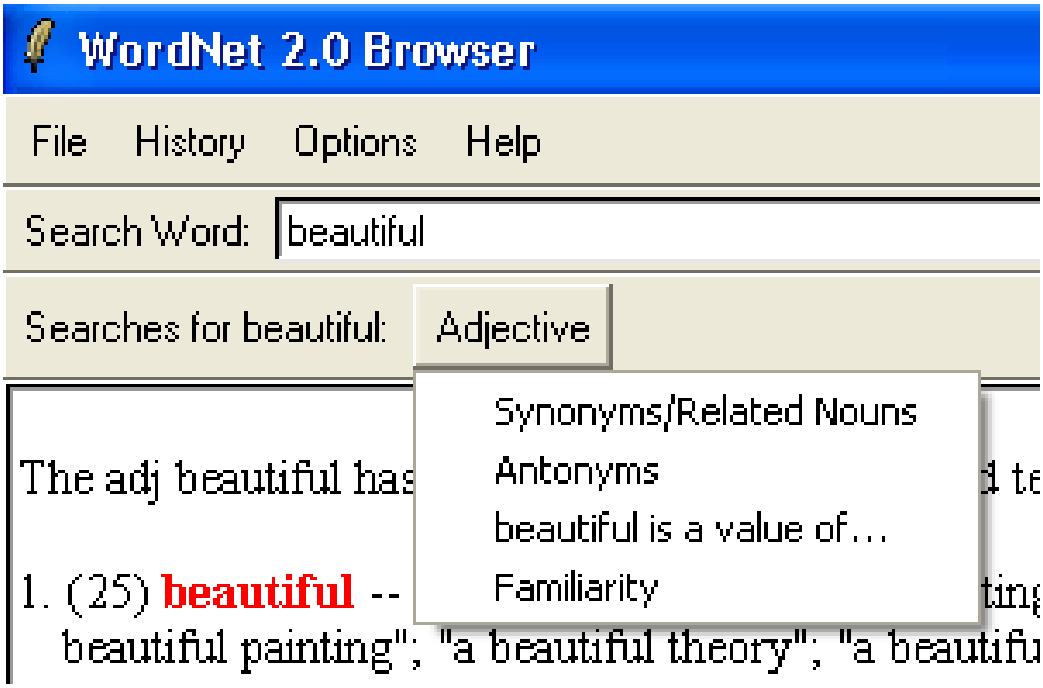

Figura A.5: Ricerca ulteriore: opzioni associate agli aggettivi

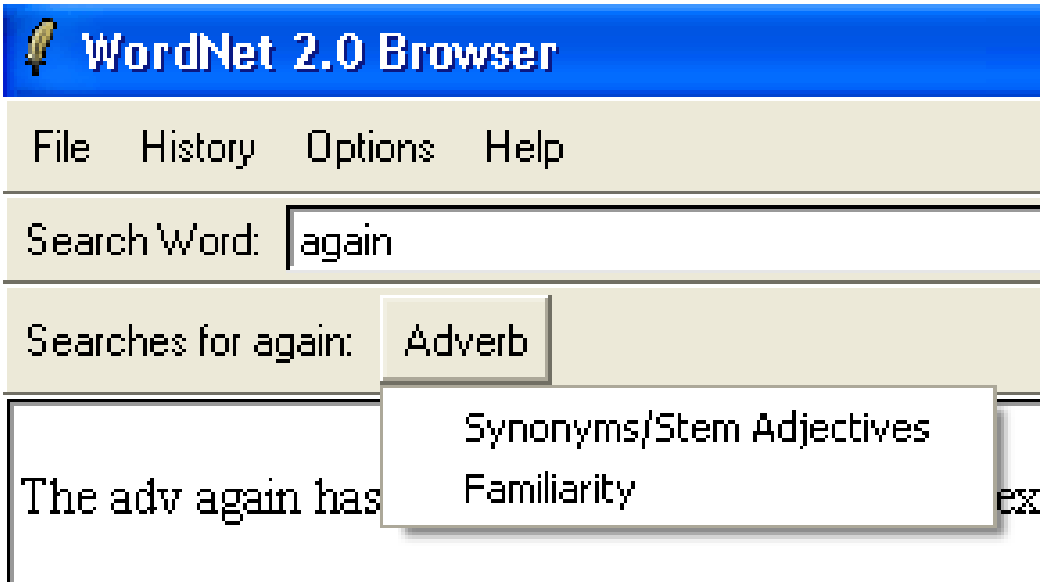

Figura A.6: Ricerca ulteriore: opzioni associate agli avverbi

Appendice B

Dizionario matematico: source file di partenza

Come abbiamo visto nel capitolo 3 alla base di un dizionario matematico è necessario porre dei source file che coprano in modo esauriente il campo tematico in esame. In questa appendice è riportata a seguire una lista di possibili termini da cui partire per creare un adeguato insieme di gerarchie e quindi di source file.

algebra
algoritmo
ampiezza
angolo
approssimazione
arco
argomento
arrotondamento
ascissa
asintoto
asse
asserzione

assoluto
autovalore
autovettore
binomio
bisettrice
calcolo
cammino
campionamento
campo
chiuso
cifra
classe
codominio
coefficiente
combinazione
complesso
condizione
coniugato
convergenza
coordinata
costante
crescenza
cubica
curva
dato
decrescenza
definizione
denominatore
densità
derivazione
determinante
deviazione
diagonale
diagramma
differenza
differenziale
dimensione
direzione
discontinuità
discretezza
disparità
disuguaglianza
divergenza
divisione
dominio
elemento
equazione
errore
esistenza
esponente

espressione

estrapolazione

estrazione

estremo

fattore

figura

forma

formula

funzionale

funzione

gradiente

grado

grafico

grandezza

hessiano

identità

immagine

imprecisione

incertezza

inclinazione

incognita

incremento

indice

indicizzazione

indipendenza

infinitesimo

infinito

insieme

integrale

integrazione

intero

interpolazione

intervallo

intorno

inversione

iterazione

jacobiano

lagrangiano

limitazione

limite

linea

massimo

matrice

media

mediana

membro

metodo

minimo

minore

misura

modulo
moltiplicazione
monotonia
negativo
negazione
nodo
norma
normalizzazione
numeratore
numerazione
numero
operatore
operazione
opposto
ordinata
ordine
origine
ortogonalità
oscillazione
ottimizzazione
parabola
parallelismo
parametro
passo
pendenza
peso
piano
polinomio
polo
positivo
posizione
potenza
precisione
predizione
primitiva
principio
probabilità
processo
prodotto
proporzione
proprietà
propagazione
punto
quadratura
quadrato
quoziente
radice
rango
rapporto
rappresentazione

valutazione

variabile

variazione

vettore

virgola

zero

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Indice analitico

Bibliografia

[1] *WordNet: An Electronic Lexical Database (1990)*, MIT Press.

[2] *WordNet Reference Manual*

[3] **Artale, Alessandro e Magnini, Bernardo e Strapparava, Carlo** *WordNet for Italian and Its use for Lexical Discrimination (1997)*, In Maurizio Lenzerini (Ed.) AI*IA 97: Advances in Artificial Intelligence Proceeedings of the 5th Congress of the Italian Association for Artificial Intelligence, Roma.

[4] **Bentivogli, Luisa** *Relazioni lessicali e semantiche nella costruzione di un lessico computazionale multilingue: problematiche tecniche e filosofiche (1998)*, Tesi di laurea, Università degli Studi di Trento.

[5] **Cariolaro, Gianfranco** *La Teoria Unificata dei Segnali (1996)*, UTET Libreria

[6] **Chiffi, Antonio** *Analisi Matematica 1 (1997)*, Edizioni C.A.G. - Padova.

[7] **Chiffi, Antonio** *Analisi Matematica 2 (1997)*, Edizioni C.A.G. - Padova.

[8] **Lippman, Stanley B.e Lajoie, Josèe** *C++ Corso di programmazione (terza edizione) (2000)*, Addison - Wesley.

[9] **Miller, G.A. e Beckwith, R. e Fellbaum, C. e Gross, D. e Miller, K.** *WordNet: an on-line lexical database (1990)*, International Journal of Lexicography (special issue), 3 (4), pp. 235-312.

[10] **Miller, G.A. e Beckwith, R. e Fellbaum, C. e Gross, D. e Miller, K.** *Introduction to WordNet: an on-line lexical database (1993)*, http://www.cogsci.princeton.edu/ wn/.

[11] **Miller, G.A. e Fellbaum, C.** *Semantic networks of English (1991)*, In Levin, B. e Pinker, S. (a cura di): Lexical and conceptual semantics. Cambridge (Mass.): Blackwell.

[12] **Miller, G.A.** *WordNet: a Dictionary Browser (1985)*, In Proceedings of the First International Conference on Information in Data. University of Waterloo.

[13] **Minnaja, Carlo** *Metodi matematici per l'ingegneria (Parte prima) (1997)*, Edizioni Libreria Progetto - Padova.

[14] **Minnaja, Carlo** *Metodi matematici per l'ingegneria (Parte seconda) (2000)*, Edizioni Libreria Progetto - Padova.

[15] **Monti, C.M. e Pierobon, G.** *Teoria della probabilità (2000)*, Edizioni Decibel - Padova.

[16] **Passarotti, Marco** *La lemmatizzazione. Cos'è, perché si deve fare, come io credo convenga farla (2002).*

[17] **Roventini, Adriana e Alonge, Antonietta e Bertagna, Francesca e Calzolari, Nicoletta e Marinelli, Rita e Magnini, Bernardo e Speranza, Manuela e Zampolli, Antonio** *ItalWordNet: a Large Semantic Database for the Automatic Treatment of the Italian Language (2001)*, Istituto di Linguistica Computazionale di Pisa, ITC-irst Trento.

[18] **Stefanini, Erika** *Risoluzione di ambiguità semantiche per la ricerca di similarità tra frasi (2003)*, Tesi di laurea, Università degli Studi di Modena e Reggio Emilia.

[19] **Vossen, Piek** *Extending, trimming and fusing WordNet for technical documents (2001)*, Proceedings of the NAACL 2001 Workshop on WordNet and Other Lexical Resources, Pittsburgh, June 2001.

[20] **Zingarelli, Nicola** *Vocabolario della lingua italiana (1988)*, Zanichelli.

[21] **Zipf, G.K. (1945)** *The Meaning-Frequency Relationship of Words*, Journal of General Psychology, 33, pp. 251-256.

[22] *EuroWordNetBuilding*
`http://www.illc.uva.nl/EuroWordNet/`

[23] *GlobalWordNet*
`http://www.globalwordnet.org/`

[24] *Istituto di Linguistica Computazionale*
`http://www.ilc.cnr.it/indexflash.html`

[25] *ItalWordNet*

http://tcc.itc.it/research/textec/topics/multiwordnet/

[26] *Programmazione.it*

http://www.programmazione.it/

[27] *WordNet*

http://www.cogsci.princeton.edu/~wn/

Grazie

Ho aspettato per tanto tempo questo giorno. Questo è il punto di arrivo di un cammino lungo che ho affrontato a tratti con entusiasmo, a tratti con tanta difficoltà. Specialmente l'ultimo periodo è stato il più duro e in alcuni momenti credevo di non farcela, e proprio in questi momenti sono state importanti le persone che hanno avuto fiducia in me e che, se avevo bisogno, c'erano, pronte a dirmi 'Dai che ce la fai!'. Per questo voglio dire grazie a tutti coloro che sono stati tanto presenti nel mio cuore in questi anni.

Voglio dire grazie alla mia famiglia perché ha sempre creduto che sarei arrivato alla laurea, a volte anche più di me. Un grazie va anche alle mie nonne che mi hanno sempre portato nei loro pensieri e nelle loro preghiere.

Grazie a Elisa perché la parola amicizia mi fa pensare a lei e alla sua presenza nella mia vita, tanto importante e tanto bella per me; perché quando la vedo mi nasce sempre un sorriso.

Grazie a Mattia, Taba, Marco e Lion, gli amici di sempre, quelli che non si scelgono, ma sono contento di avere avuto, quelli con cui ho trascorso tanta parte della mia vita.

Grazie a Federico perché ha condiviso con me gioie e difficoltà come

nessun'altro negli anni universitari.

Grazie ai ragazzi di via Colle 10/6 perché posso dire davvero di aver passato con loro quattro anni alla grande.

Un grazie va anche a chi mi ha aiutato a realizzare questa tesi, atto conclusivo della mia carriera universitaria.

Grazie al professor Carlo Minnaja ed alla professoressa Laura Paccagnella per la loro disponibilità e il loro continuo appoggio.

Grazie a Rudi perché non so quante cose gli ho chiesto, dato che sono troppe, e lui mi ha sempre aiutato.

Infine grazie a Italo Calvino che un giorno scrisse 'Non illuderti mai, ma non smettere di credere che tutto ciò che fai potrà servire'.