Attention: Professor Bart Selman

Department of Computer Science

Raphael Rubin

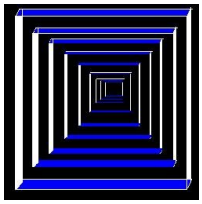
MENG ECE Cornell University

rr284@cornell.edu

Web Analytics: Search Engine based on linguistics and AI methods, Semester II, 2006-2007

Table of Contents

AN OPTIMAL WEBSITE DESIGN, SEARCH ENGINE AND AD PRODUCTION AND CUSTOMIZATION BASED ON STATISTICAL INFORMATION ON THE WWW



I. Executive Summary:

We extract minimal information about a category's content based on website design, semantics.

We decompose sentences, parsing them through thesaurus, dictionaries and encyclopedias, matching sentences to typical sentences; building a set of logic clauses matching a particular content.

We start from man made directories, parsing their websites and categories at two sub-levels.

We have tested different architectures for thread based web bots, some extremely powerful and allowing to control the number of threads running in the background.

We have achieved rule-based parsers for sentences, for specific thesaurus, encyclopedia and dictionary contents.

We have not been able to complete the replacement of these sets of logic clauses by neural networks made of incremental knowledge mechanisms, nor the extraction of the design of a website to categorize it.

We aim at providing business oriented information where precise reports of the type of content, or ad content, and where it appears would allow competitors to re-evaluate their objectives.

Moroever a parallel 'ping' of yahoo and googke queries would return best entries and publicity links and a similar approach could be used as such.

First, the main goal is extract the minimal information set about a particular web category, based on the entire mad made websites, thersaurus, dictionaire, encyclopedias available on the web.

Ultimately, the algorithm would return search results based not on the content but on its signification, breaking language barrers, domestic language and foreign language barriers.

We have not extracted the minimal informational content of a website based on its HTML designs, its image and video onctents, its disposition, specific keywords, only its semantics.

II. Methodology:

Someone's search would be decomposed as:

- CONFIDENCE VOTE

  - INDEXED SEARCH
  - LIVE EVENTS SEARCH
  - ADS SEARCH

A. INDEXED SEARCH

- CATEGORY KEYWORDS (Check Box) {Art=>Painting=>Classical) & (Museums) & (French)
- INFORMATION sentence: "French painter 20th century Monet"
- **COMPARE** with "English Painter 19th century Morrison"
- **COMPARE** with "other…"
- ON BASIS of
  - CATEGORY KEYWORDS (Check Box) {Arts->galleries->Prices, quotas)
  - INFORMATION: "which works are is most expensive?"

- CATEGORY KEYWORDS (Check Box) {Art=>Painting=>Classical) & (Museums) & (French)

- INFORMATION sentence: “French painter 20th century Monet”

- **MERGE** with “English Painter 19th century Morrison”

- **MERGE** with “other…”

- ON BASIS of

    - CATEGORY KEYWORDS (Check Box) {Arts->galleries->Prices, quotas)

    - INFORMATION sentence: “which works are is most expensive?”

The engine will return a set of sentences matching the specifications, generating an automatic report. It may be a lengthy process but will lead to very rigorous results.

If all sites have been previously indexed, it is not the case (lengthy).

Yet, indexation is not the ideal method for all requests.

Indeed searches modes include **INDEXED SEARCH, LIVE EVENTS SEARCH or ADS SEARCH.**

The report will contain a list of sentences containing the information requested, its source.

A 100% confidence vote will produce a relatively small document, while the decrease in the former percentage will increase the latter size.

Compare will output a line by line argumentative in favor or disfavor of the hypothesis raised.

Merge will produce a complementary or synthetic version of the multiple information provided.

B. ADS SEARCH

An advertiser wishes ultimately to adapt to the consumer's behavior. Yet, unless it goes through the traditional search engines capabilities - engines which have access to logs and monitor user behavior - they are being deprived of critical information.

Our aim is therefore to reverse engineer the *information held by these search engines* so that it is restored freely to the www end users, ad providers, webmasters and designers etc…

Let us suppose that ads are being served optimally on the World Wide Web, and that assumption may be far fetched, unfortunately;
meaning that every ad provider relies on services such as Google Ad sense

(Maximize one's site's revenue potential with contextually targeted ads)
and Google Ad Words (for ads providers)

Our purpose is to reverse engineer these services to be able to provide to third party clients relevant information, whether they do or do not have a pay per service account with search

engines such as Google, Yahoo, or MSN.

To maximize the amount of information we are seeking to retrieve, we will also use Yahoo's Publisher Network, Yahoo Advertising Agency, MSN's services
and MSN Advertising Experts Ad Center.

These paying services are provided to website content providers and to ad providers.

How do we reverse engineer this critical information, user logs, user queries?

We create scenarios:

Ads respond to user queries, on search engine sponsored links. They also appear

on visited commercial or informational websites.

Clearly, if there is a trend in keywords search, *it will likely be detected through the live events* ENGINE described above.

- CATEGORY KEYWORDS (Check Box) {Manufacturing=>textile=>shoes=>) & (USA) & (NYC) & (price comparators)

- INFORMATION sentence: “Buy the cheapest sporting goods with Nike”

- **MERGE** with **LIVE EVENTS**

This will return a report indicating how live events (new prices of sporting goods listed on website; doesn’t have to be ads) have affected the contents of ads published on the Internet, in the categories mentioned.

Thus the correlation of ads with the LIVE EVENTS indicates which part derives from LIVE EVENTS and which derives from LONG TERM TRENDS.

Thus LONG TERM TRENDS may be estimated, along LIVE EVENTS, and this, for every category.

C. LIVE EVENTS SEARCH

To browse the web in search of live events, inside categories such as news, sports, business etc… and enable advertisements to adapt to the user's behavior(requests inside a third party website) and to the outside web's information.

D. SERVICES TO THIRD PARTY AD PROVIDERS

The services we want to provide to the third party ad providers are:

1- Website Category and Sub Category Vs Ad Category and Sub Category HOTLIST

2- Ad Category and Sub Category Vs Sponsored Links HOTLIST

3- Keyword Search Vs Sponsored Links HOTLIST

4- Sponsored Links Category and Sub Category Vs Keyword Search HOTLIST

5- Ad Customization Vs Keywords Search HOTLIST

6- Live Events gathered from multiple sources Vs Keyword Search HOTLIST

7- Predicting future search engine queries Vs Live Events HOTLIST

8- Determining an ad success based on its duration, the number of sites of the campaign, the number of versions of the ad.

These services will enable ad providers to have a glimpse of the ads displayed by their competitors, to perceive their real market share or impact. These services will allow ad providers to make better ads, customized to specific websites, specific events, and specific queries.

E. SERVICES TO THIRD PARTY WEBSITES

The services we want to provide to the third party websites are the same as above plus:

1- Customization of Non Sponsored Links VS sponsored links content impact

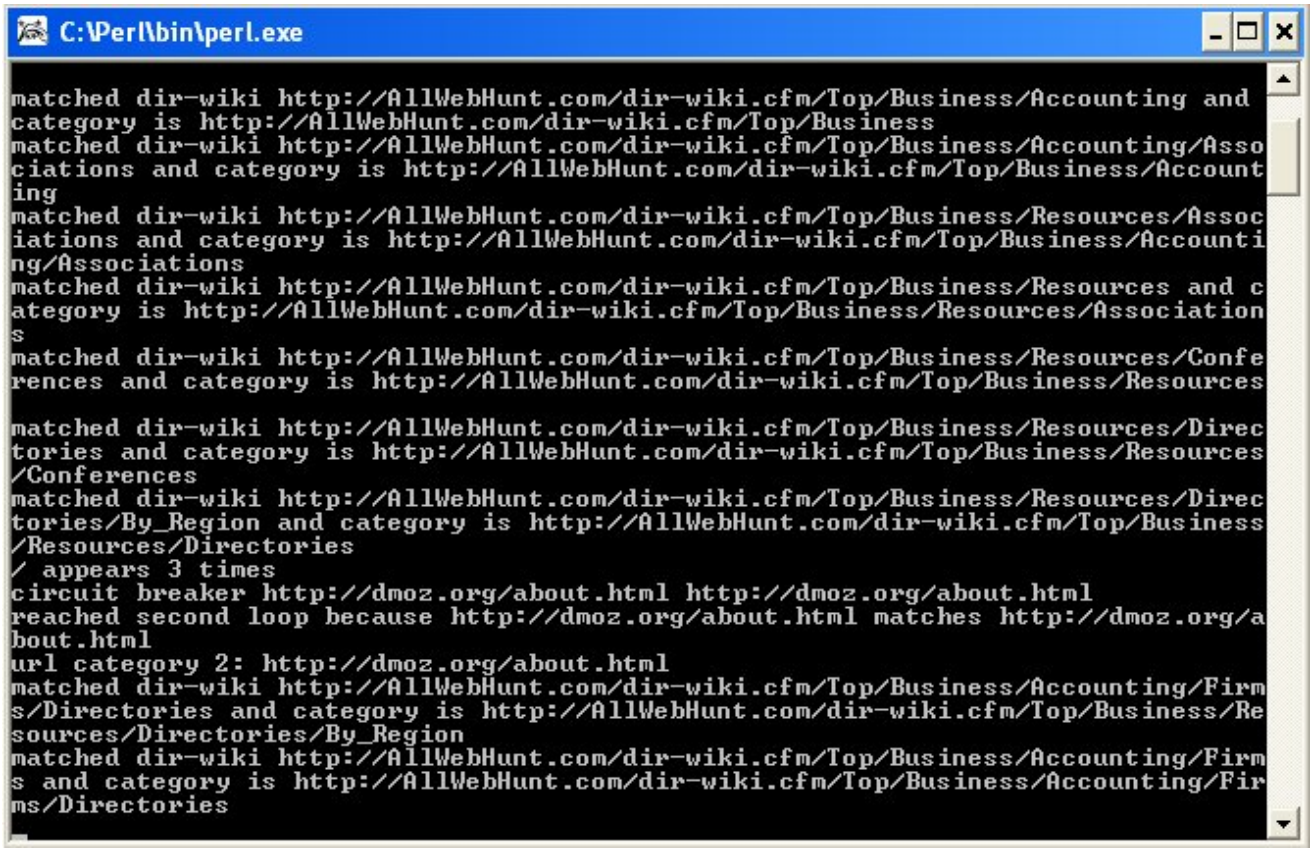
2- Customization of websites Vs predicted future search engine queries

---

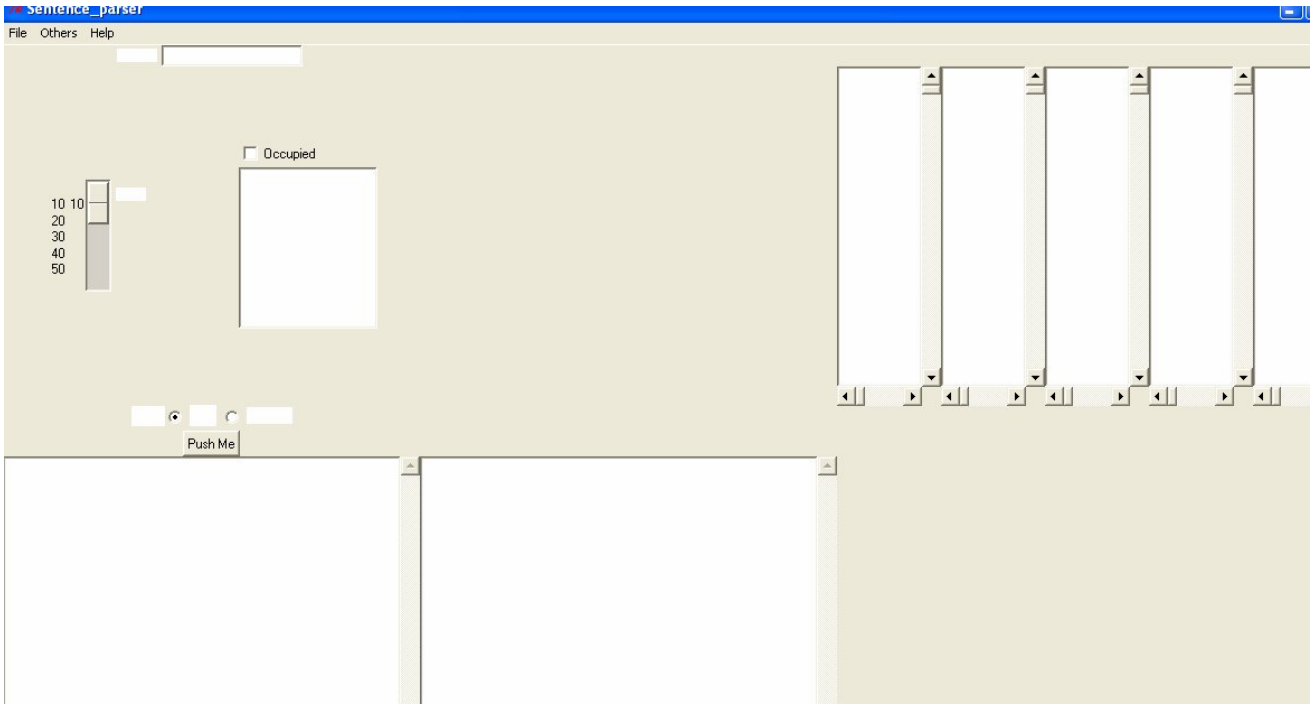
We have now databases containing the WWW websites classified by category and subcategory, in a top-bottom approach. We are interested in data mining.

The content of the websites is going to be extracted.

F. Screen Shots of the web bot, based on goole or allwebhunts web directories.



1. **Screen Shots of the sentence parser: (static)**



C:\eclipse\AllWebHunt>perl sentence_parser.pl

subject or object I found

begin rules with ITÆS

tense is present and abstract concept or phenomenon

article: a

begin rules with a

begin rules with common

found fvrofierhouregerreH2

begin rules with juggling

begin rules with act

preposition: on

begin rules with on

article: the

begin rules with the

begin rules with doorstep:

begin rules with rummaging

preposition: for

begin rules with for

article: the

begin rules with the

begin rules with house

begin rules with keys

preposition: with

begin rules with with

begin rules with one

begin rules with hand

begin rules with while

begin rules with balancing

article: a

begin rules with a

begin rules with bag

preposition: of

begin rules with of

begin rules with groceries

preposition: with

begin rules with with

article: the

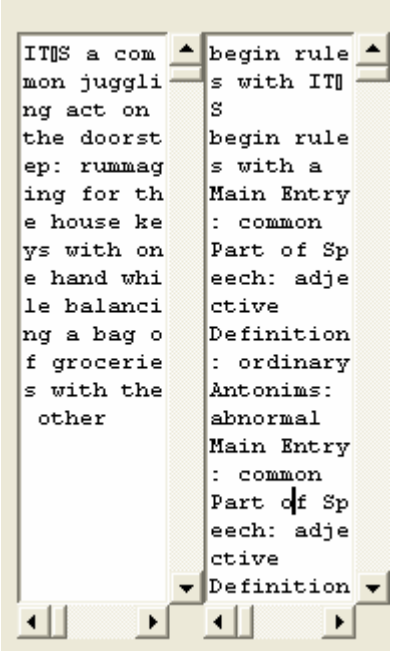
begin rules with the

begin rules with other

verb(present,be)

,adjective,adjective,pronoun,adjective,adjective,noun,noun,adverb/adjective,noun

,adverbThe basic cs requirements have0 been met



C) View of Dictinary, Thesaurus and Encyclopedias being parsed:

Dictionary.com Unabridged (v 1.1) – *Cite This Source*

nav·i·gate [**nav**-i-geyt] Pronunciation Key *verb,* **–gat·ed, –gat·ing.**

–verb (used with object)

1. to move on, over, or through (water, air, or land) in a ship or aircraft: *to navigate a river.*
2. to direct or manage (a ship, aircraft, or guided missile) on its course.
3. to ascertain or plot and control the course or position of (a ship, aircraft, etc.).
4. to pass over (the sea or other body of water), as a ship does.
5. to walk or find one's way on, in, or across: *It was difficult to navigate the stairs in the dark.*

–verb (used without object)

6. to direct or manage a ship, aircraft, or guided missile on its course.
7. to pass over the water, as a ship does.
8. to walk or find one's way.
9. to travel by ship or boat; sail.

[Origin: 1580–90; < L *nāvigātus,* ptp. of *nāvigāre* to sail, deriv. of *nāvis* ship; for formation, see FUMIGATE]

9 results for: *navigate*

1–9 of 9 results

View results from: Dictionary | Thesaurus | Encyclopedia | All Reference | the Web

Roget's New Millennium™ Thesaurus – *Cite This Source*

Main Entry: navigate
Part of Speech: *verb*
Definition: guide
Synonyms: cross, cruise, direct, drive, handle, helm, journey, maneuver, operate, pilot, plan, plot, ride out, sail, skipper*, steer, voyage

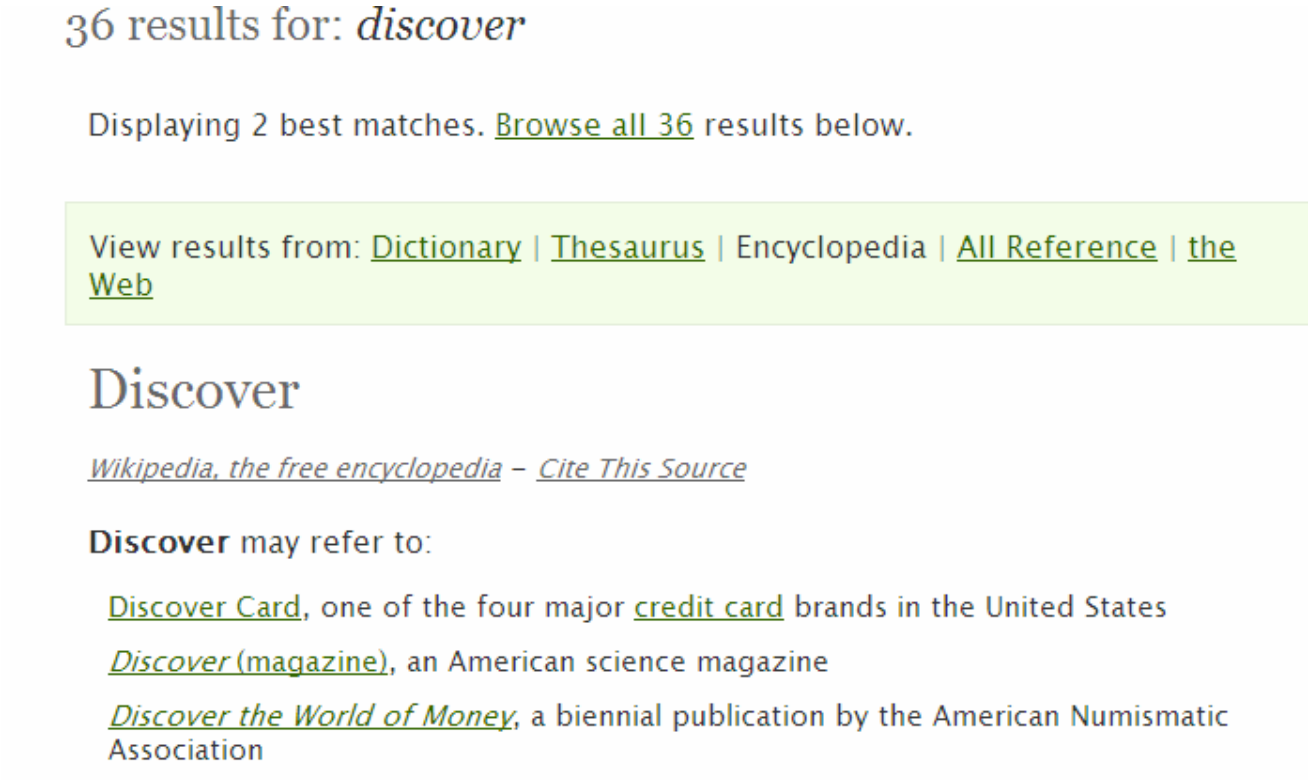
Source: *Roget's New Millennium™ Thesaurus, First Edition (v 1.3.1)*

** = informal or slang*

Roget's New Millennium™ Thesaurus – *Cite This Source*

Main Entry: cross
Part of Speech: *verb* [1]
Definition: traverse
Synonyms: bridge, cruise, cut across, extend over, ford, go across, meet, move across, navigate, overpass, pass over, ply, sail, span, transverse, voyage, zigzag
Antonyms: remain, stay

Source: *Roget's New Millennium™ Thesaurus, First Edition (v 1.3.1)*



III. Natural Language Processing: Information Extraction

We finally get to the part where we must read through every sentence of parsed text.

An example of information extraction is the extraction of instances of corporate mergers, more formally *MergerBetween*(*company*$_1$,*company*$_2$,*date*), from an online news sentence such as: "Yesterday, New-York based Foo Inc. announced their acquisition of Bar Corp."

Our task is to deduct the classification or category to which the website belongs according to itscontent.

IV. Sentence Structure:

Pronoun: *subject*ive personal pronoun {"I," "you," "she," "he," "it," "we," "you," "they."}

WEwas glad

***Objecti*ve personal pronoun: {**"me," "you," "her," "him," "it," "us," "you," and "them."}

she threw it.

Possessive Personal Pronouns {"mine," "yours," "hers," "his," "its," "ours," and "theirs."}

Demonstrative Pronouns { this, that, these, those}

Interrogative Pronouns {who,whom,which,what, whoever," "whomever," "whichever," and "whatever"}

Relative Pronoun : used to link one phrase or clause to another phrase or clause

{ "who," "whom," "that," and "which." "whoever," "whomever,"whichever"}

You may invite **whomever** you like to the party.

WEwill read **whichever** manuscript arrives first.

Indefinite Pronoun{ "all," "another," "any," "anybody," "anyone," "anything," "each," "everybody," "everyone," "everything," "few," "many," "nobody," "none," "one," "several," "some," "somebody," and "someone."}

Reflexive Pronouns { "myself," "yourself," "herself," "himself," "itself," "ourselves," "yourselves," and "themselves."}

Intensive Pronoun ***emphasizes its antecedent***

WEmyself

Adjective: modifies a {noun, pronoun}

{past participle} muffled sounds

Possessive{my," ``your," ``his," ``her," ``its," ``our," ``their")

WEcan't complete my **assignment**

Demonstrative {``this," ``these," ``that," ``those," and ``}

When the librarian tripped over **that** cord

Interrogative {which,what,whose}

Indefinite {many,all,any, much}

Possessive { my, your, his, her, its, their, our}

Noun: { **{**name a person, animal, place, thing, abstract idea}

{subject, a direct object, an indirect object, a subject complement, an object complement, an appositive, an adjective or an adverb}

Direct Object

The advertising executive drove **a flashy red Porsche**.

Indirect Object

Her secret admirer gave Mary **a bouquet of flowers**.

Subject Complement

The driver seems **tired**.

In this case, as explained above, the adjective "tired" modifies the noun "driver," which is the subject of the sentence.

Object Complement

WEconsider the driver **tired**.

Appositive: (*like the subject complement but without the linking verb)*

My brother **the research associate** works at a large

Adverb: modify {a verb, an adjective, another adverb, a phrase, or a clause **}**

Conjunctive adverbs:

The government has cut university budgets; **consequently**, class sizes have been increased.

Verb Class:

Verb { {action, reaction, emotion, location, physical, spiritual, identification, mental action }

{past {was,were,have, have} , current, future{will,shall}}

{regular, irregular}

{2 verbs in a row}

{ has an extension-conjugation}

Regular Verbs :

The number of arguments that a verb takes is called its *valency* or *valence*. Verbs can be classified according to their valency:

- **Intransitive** (valency = 1): the verb only has a subject. For example: "he runs", "it falls".
- **Transitive** (valency = 2): the verb has a subject and a direct object. For example: "she eats fish", "we hunt deer".
- **Ditransitive** (valency = 3): the verb has a subject, a direct object and an indirect or secondary object. For example: "WEgave her a book," "She sent me flowers."

Irregular Verbs :

Infinitive Simple Past Past Participle

Arise arose arisen ; Awake awakened / awoke awakened / awoken; backslide backslid backslidden / backslid; be was,were been; bear bore born/borne; beat beat beaten/beat; become became become; begine began begun; bend bent bent; bet bet/betted bet/betted; bid(farewell) bid/bade bidden; bid(offer amount) bid bid; bind bound bound; bite bit bitten; bleed bled bled; blow blew blown…

Preposition: links { nouns, pronouns and phrases to other words in a sentence}

{"about," "above," "across," "after," "against," "along," "among," "around," "at," "before," "behind," "below," "beneath," "beside," "between," "beyond," "but," "by," "despite," "down," "during," "except," "for," "from," "**in**," "inside," "into," "like," "near," "of," "off," "on," "onto," "out," "outside," "over," "past," "since," "through," "throughout," "till," "to," "toward," "under," "underneath," "until," "up," "upon," "with," "within," and "without."}

She held the xbook **over** the table.

The dog is hiding **under** the porch because it knows it will be punished **for** chewing up a new pair **of** shoes.

Conjunctions: Coordinating{ and," "but," "or," "nor," "for," "so," or "yet"}

Subordinating { "after," "although," "as," "because," "before," "how," "if," "once," "since," "than," "that," "though," "till," "until," "when," "where," "whether," and "while."}

After she had learned to drive

If the paperwork arrives on time

Correlative { both ,and

Either, or

Neither, nor

Whether, or

Not only, but also}

ThesaurusReference provides:

Main Entry:

Part of Speech:

Definition:

Synonyms:

Antonyms:

DicitonaryReference provides:

**

Our Sample Result:

C:\eclipse\AllWebHunt>perl thesaurusreference_standalone.pl

Prefix: es

conjugation forms : caped

conjugation forms : caping

new sentence for verb without object: to escape from jail.

new sentence for verb without object: The words escaped from memory.

new sentence for verb without object: (of a rocket, molecule, etc.) to achieve escape velocity.

new sentence for verb with object: He escaped the police.

new sentence for verb with object: She escaped capture.

new sentence for verb with object: Her reply escapes me.

new sentence for noun: We used the tunnel as an escape.

new sentence for noun: She reads mystery stories as an escape.

new sentence for noun: the act of achieving escape velocity.

new sentence for adjective: an escape route.

related form: escapable adjective

related form: escapeless adjective

related form: escaper noun

related form: escapingly adverb

Synonims: flee, abscond, decamp.

Synonims: dodge, flee, avoid.

Synonims: flight.

related form: verb

new sentence for noun: Make your escape while the guard is away; There have been

several escapes from that prison; Escape was impossible; The explosion was caus

ed by an escape of gas.

IV: Suggestion for a Self-Modifying Program:

We need the clauses we build to increment iteratively to reflect the progress or closing-in definitions, in between the cast type expression:

```
return <<'    END_PROG';

                END_PROG
```

In order to achieve a self-modifying program, we have the ability as threads call several modules to modify them on the fly.

VI: Saving a Perl complex data structure from one execution to the next:

http://www.antipope.org/charlie/perl/perltut/perltut-12.txt

Perl programmers often want to save hairy data structures; for this reason, we turn to a couple of modules that people have written for exactly this job -- Data::Dumper and FreezeThaw.

VII: Information contained in the clauses:

A sentence is defined by its category: *Business/Agriculture_and_Forestry/Trade_Shows*

It is also made up of constituents: *subjective_personal_pronoun objective_personal_pronoun possessive_personal_pronoun demonstrative_pronouns interrogative_pronouns relative_pronouns indefinite_pronouns reflexive_pronouns intensive_pronouns past_participled_adjective possessive_adjective demonstrative_adjective interrogative_adjective indefinite_adjective possessive_adjective superlative_adjective comparative_adjective comparative_irregular_adjective superlative_irregular_adjective person_noun animal_noun place_noun thing_noun abstract_idea_noun subject_noun direct_object_noun indirect_object_noun suject_complement_noun object_complement_noun appositive_noun adjective_noun verb_adverb adjective_adverb adverb_adverb phrase_adverb action_verb emotion_verb location_verb physical_verb spiritual_verb identification_verb mental_verb past_verb present_verb future_verb regular_verb irregular_verb inarow_verb nouns_preposition pronouns_preposition phrases_preposition coordinating_conjunction subordinating_conjunction correlative_conjunction*

Let us analyze a sentence belonging to Iowa Power Farming Show **a website whose description is:**
An indoor trade show for farmers, featuring agriculture and farm equipment in Iowa, USA. Includes exhibitor list, floor plan, location and an on-line application form.

"Designed to showcase new equipment and products for farmers, the Iowa Power Farming Show is the largest indoor ag shows in the upper Midwest and one of only two Midwest ag shows bringing together major machinery manufacturers and exhibitors from around the world."

I. Sentence = Clause1 + comma + clause2
II. Clause1 = **Designed to showcase new equipment and products for farmers**
III. Clause2 = **the Iowa Power Farming Show is the largest indoor ag shows in the upper Midwest and one of only two Midwest ag shows bringing together major machinery manufacturers and exhibitors from around the world**
IV. Sample data for article the:

[the]Definite article, new type matching previously known type **article**

[Iowa] *noun,* a state in the central United States: ….

[Power]

[Farming]

[Show]

[is] irregular verb: be, present

[the] Definite article

[largest] superlative_adjective: large

[indoor] adjective

[ag] ?

[shows] noun

[in] preposition, adverb, adjective, noun, verb without object, idioms

[the] Definite article

[Upper] superlative_adjective: up

[Midwest]

[and] coordinating_conjunction

[One] number

[of] ?

[only] adverb

[two] number

[Midwest]

[ag] ?

[shows] noun

[bringing] irregular_verb: bring, past

[together] ?

[major] noun/adjective

[machinery] noun

[manufacturers] noun

[and] coordinating_conjunction

[exhibitors] noun

[from] ?

[around] ?

[the] definite article

[world] noun

V. Sentence Reduced to clauses:

Where **the** …**verb** … **object** is a full sentence

Where **for** and **to, in**, are the **properties** of the object.

Where **is defines : -**

Where **shows** may be a verb, noun.

Where **two** is plural.

This would generate the prolog clause:

```
sub sentence_category1 {
   return <<'    END_PROG';
```

Design (showcase (equipment (new), products(farmers)))

Iowa Power Farming Show : - largest indoor shows(upper Midwest)

Shows (bringing(major machinery manufacturers)

Category(Business, Agriculture, Forestry, Trade Shows)

Rules are drawn from exemplary sentences:

```
  END_PROG
}
```

VIII. Sentence Parser:

IT'S a common juggling act on the doorstep: rummaging for the house keys with one hand while balancing a bag of groceries with the other.

Be(tense,present) Be(sub,it) Be(obj,act) act(adj,juggling) act(noun,doorstep) rummage(tense,action) rummage(obj, keys) keys(noun,house) rummage(qualifier, hand) hand(number, one), rummage(concurrent, balance) balance(tense,action) balance(obj, bag) bag(noun,groceries), balance(qualifier, other)

Now a manufacturer is aiming to streamline household entry with a deadbolt lock that reduces the need to fish for keys. Instead, the lock opens with the swipe of a finger.

Now, Instead, best…typical commercial keywords

aim(tense,action) aim(sub,manufacturer), aim(verb, streamline) streamline(sub,entry), entry(noun,household) streamline(qualifier,lock) lock(noun,deadbolt) reduce(sub, lock) reduce(tense,present) reduce(obj, need) need(verb,fish), fish(obj,keys) open(sub, lock), open(tense,present), open(qualifier,swipe), swipe(noun,finger)

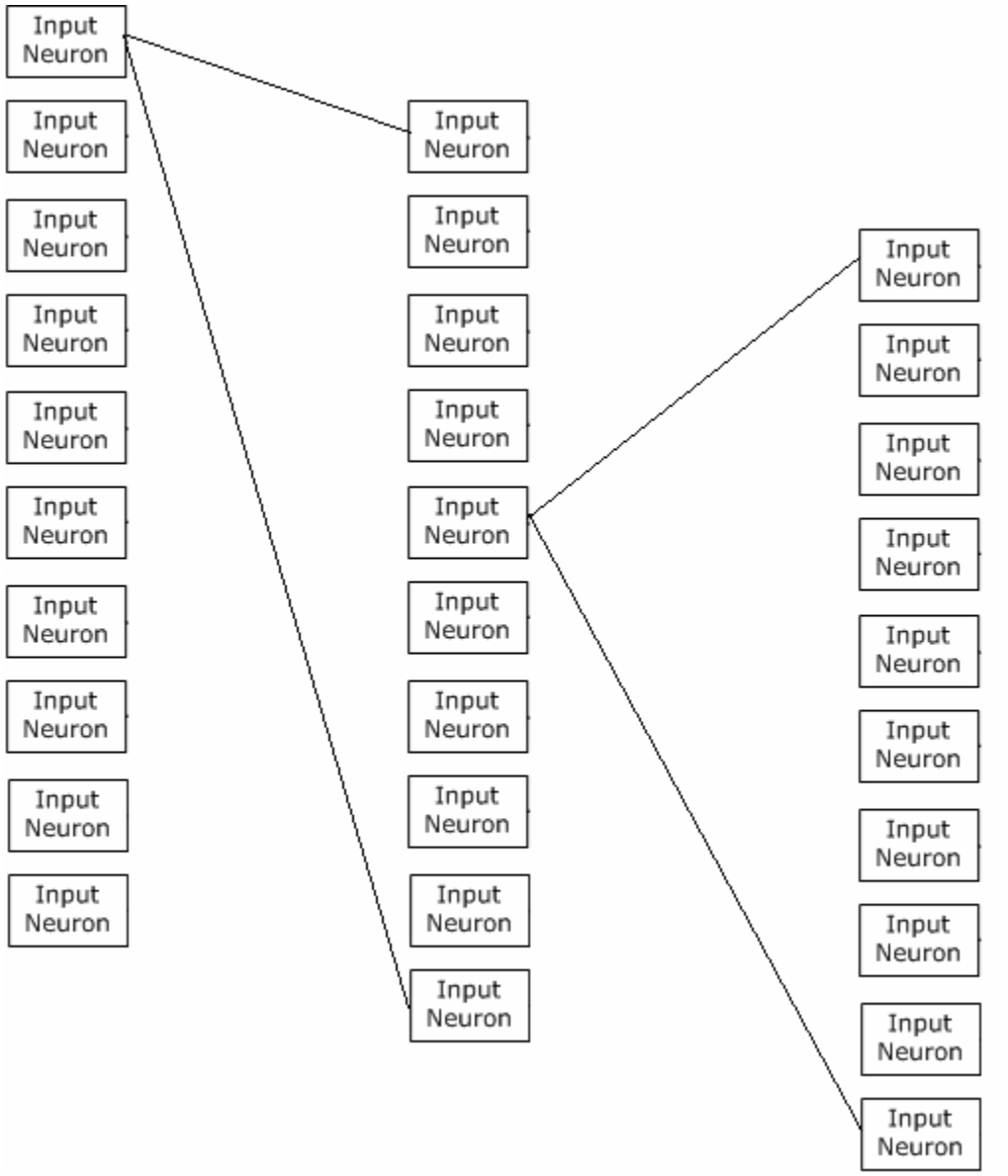
The lock is a scanner that stores the fingerprints of authorized users.

Be(tense,present) be(sub,lock) be(obj,scanner) store(tense,present) store(sub,scanner) store(obj, fingerprint) fingerprint(noun,user) user(adj,authorized)

If your print matches the stored ones, the bolt slides open smartly with a crisp, satisfying clack, welcoming you home the biometric way.

Match(tense,present) match(verb,condition) match(sub,print) print(pronoun,your), match(obj,ones) ones(adj, stored) open(sub,bolt) bolt(noun,slides) open(qualifier,clack) clack(adj, crisp) clack(adj, satisfying), welcome(tense,action) welcome(obj,clack) welcome(obj, you) welcome(qualifier, home)

XIV. Neural Networks Schematics:



We are limited by the computational power of a neural network, where the maximum number of inputs is restricted to 2^30.

We aim to incorporate the results of our prolog logic clauses into the training of our neural network. Since the input will be a series of properties such as:

Design (showcase (equipment (new), products(farmers)))

And the end result, a category (10 000 000 categories for instance),

We may choose in the future to adopt sequential learning algortithms, when concerned with extracting the minimal information set about a category.

IX-a) Many-Layered L[1]earning:

The difficulty we are facing when learning about the minimal informational content of a particular category is the quantity of information to assimilate. It may be best to proceed incrementaly, processing data as input stream and structuring knowledge in a useful way.

Few-layer learning algorithms (Rumelhart, 1986 et al.) limit the number of layers and hence cause compression and learnability of data to suffer.

There is for any Boolean expression to be learned a tradeoff as restricting its layers would increase connections and gates and the number of sub-samples to be learned.

This issue is similar to the DNF which although striking by its simplicity increases obstacles to learning and data compression.

Sequential learning is refered to by Banerji (1980) as a growing language. Clark & Thornton (1997) distinguish between type 1 and type 2 learning, where type 2 learning requires a mapping of given variables to previously uncovered tpe 1 variables to uncover otherwise unobservable irregularity.

Utgoff and Stracuzzi make 3 practical assumptions.

i. Type 1 linear threshold units and linear combination units are trainable.

ii. They are individually trainable at any time and anywhere in the network. Instead of propagating erros backwards, gradient descent is used locally

iii. The representation is based on propositional and numerical variables.

[1] Many-Layered Learning, Paul E. Utgoff, Department of Computer Science, University of Massachusetts et al.

suit(x) = (x `div` 13)

column stackable(c1,c2) <-> (suit color(c1) suit color(c2)) (1+rank(c1) = rank(c2))

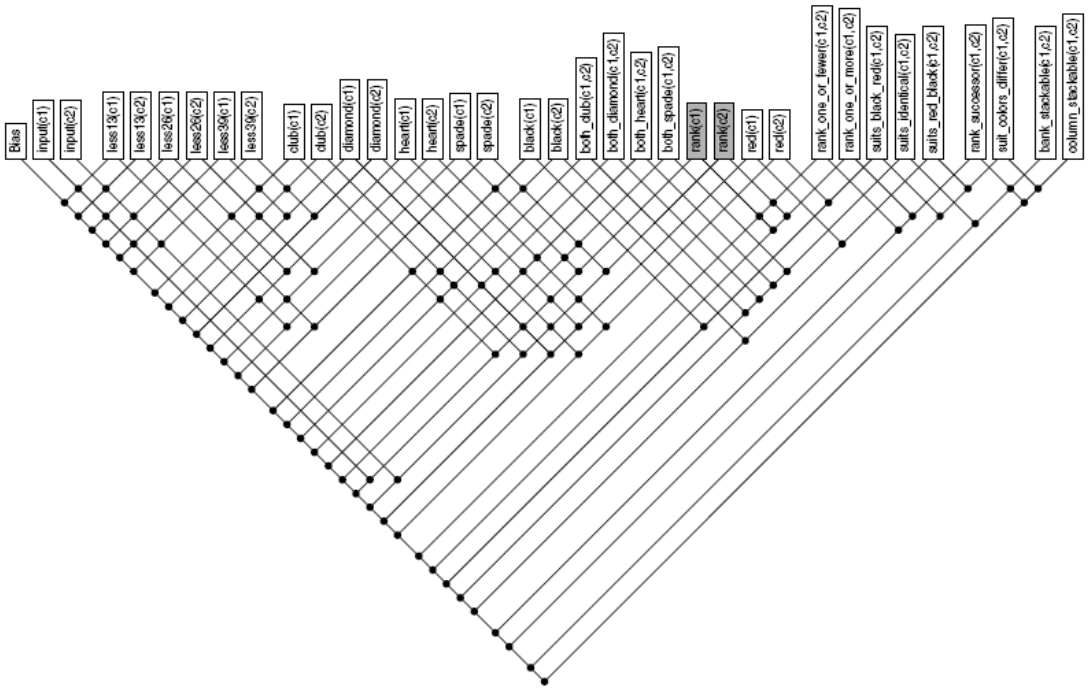


Figure 1. Hand-Designed Many-Layered Network

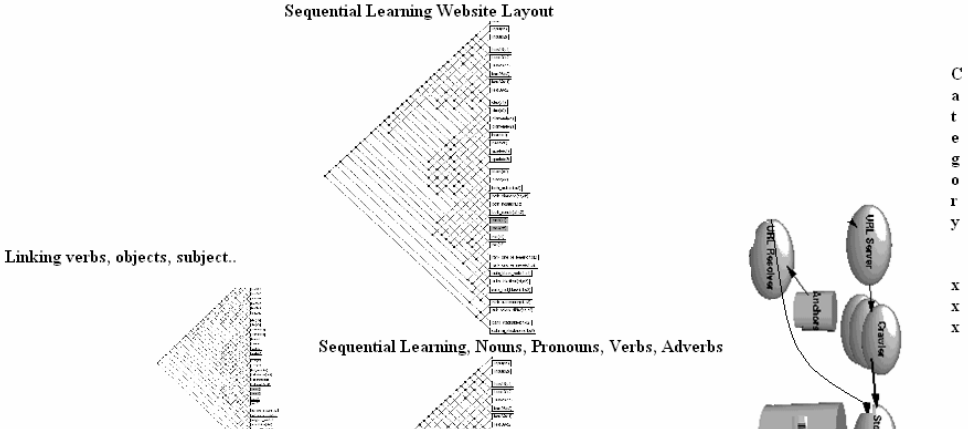
Illustration from Utgoff et al. (Many-layered learning)

Every box corresponds to a concept and is dependant on processed inputs {function1(c1), function2(c1,c2))

As well as produces an output, itself linked to another input.

The above picture shows 7 layers of computation, individually trainable and dependant one upon the other.

We can proceed in the same manner to train our web semantics for each individual category as well as cross-over between categories.



The linking is at leat to the power 8^8 and

Likely to be O(10^10)

And therefore the following diagrams are only

A tiny illustration of the massive computation

required.

Indeed we need linked nouns to nouns,

Nouns to verbs, verbs to adverbs…

As well as verbs to adverbs and subjects…

Each category is made up of this assemblage of pyramids, sequentially trained to identify objects as well as to properly link them. Crossovers between categories eliminate redundancies.

Training leads to neural networks replacing prolog clauses.

We follow Utgoff et al.'s stream-to-layer algorithm.

~less26(c1), ~less39(c1), ~spade(c1), ~heart(c1), ~club(c1), diamond(c1), ~black(c1), red(c1), rank(c1,7), ~less13(c2), ~less26(c2), less39(c2), ~spade(c2), ~heart(c2), club(c2), ~diamond(c2), black(c2),

Utgoff's Input Stream 1

We propose that the variables used in the pyramids in our above schematics be set inside the indexer. C1 becomes 'eat', c2 becomes 'present tense'.

For every predicate or function name observed the algorithm updates the unit if it is unlearned yet.

Two prepositional arguments indicate a concept while a single numerical argument indicates a numerical function.

Any concept/function which is learned successfully has its output connected to the input of those functions not yet learned successfully.

For example:

Take the sentence: "I have decided to go work at the farm because that's a family tradition".

The algorithm runs through this sentence establishing that

family tradition <-> I have decided to go work at the farm

family tradition

work at the farm

The learned concepts are crossed with other categories and with what's has already been learned so as to extract mimimal information about a category.

A learned concept might be: IPO(stocks,Nadaq)

The 'frontier of receptivity' advances to an upper layer as T1 concepts are learned successfully.

The criterion for success is that the unit must have produced a correct evaluation for at least n consecutive examples. Success in our case would be determined by a minimal difference between the current set of weights and the one required for success,

In case of failure, new connections are added before training resumes.

X. Machine Learning: review

Clearly the issues of determining user characteristics and sets of rules and customizing the contents of websites are related to machine learning.

One opportunity we would like to build upon is learning across multiple internal databases, plus the web plus news feeds.

To learn checkers, one way is to maximize the returns of a particular function corresponding to some kind of a Bellman recursive equation where every state has a value V and where if b is not a final state, V (b) = V (b'), b' being the best final move starting from b and playing optimally to the end of the game.

How should our target function should look like, collection of rules, neural network, polynomial function of board features?

Here is a representation for a chess learned function:

Wo+ w1.bp(b)+ w2.rp(b)+w3.bk(b)+w4.rk(b)+w5.bt(b)+w6.rt(b)

Where the acronyms stand for black and red pieces, black and red kings, black and red pieces threatened.

V (b) = true target function

V^ (b) = learned function

V trained (b) = training value

A possible rule for training is that V trained (b) <= **V^ (Successor (b))**

How would we tune weights?

We select a random training example b, compute its error, V train-V, and then update the weights for each board feature, wWe<= wWe+ c*fi*error(b).

In a training set, we are given instances x, a target function c (attribute), hypotheses(conjunction of litterals), and training examples where the target function takes on positive and negative values.

By definition, the hypothesis is such that h(x) = c(x) for all x.

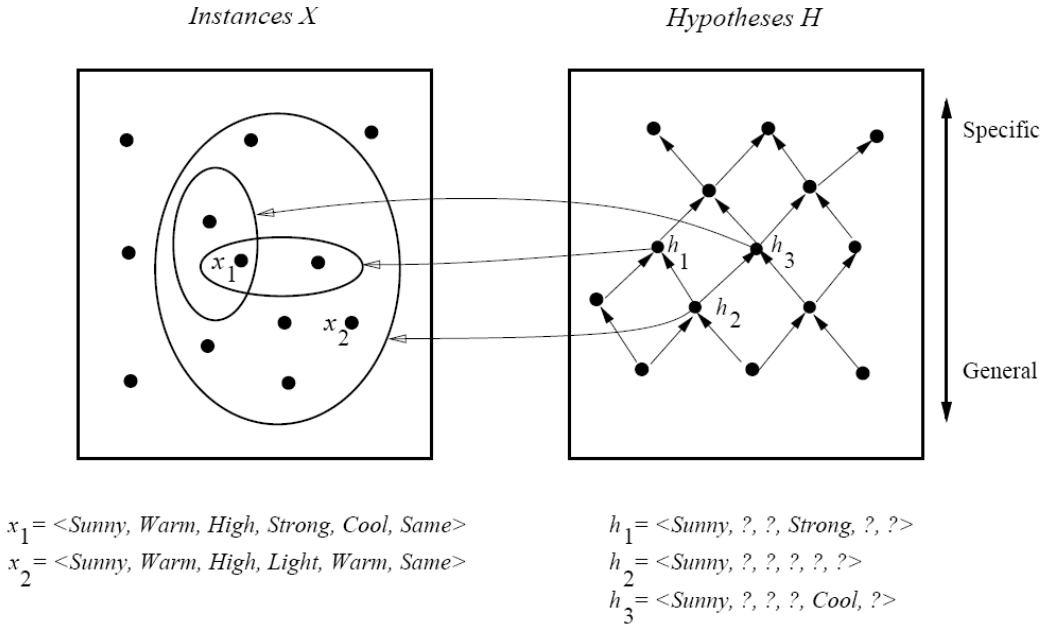


Figure 1(c)

Clearly hypotheses range from more general to more specific, matching the instance's target function result.

The algorithm Find-S outputs the as much general as possible hypothesis starting from the most specific and replacing in the hypothesis, the attribute not matching the next positive instance, by the attribute coming from the next more general hypothesis and matching this instance.

$h_0 = \langle \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing \rangle$

$x_1 = \langle Sunny\ Warm\ Normal\ Strong\ Warm\ Same \rangle, +$ $h_1 = \langle Sunny\ Warm\ Normal\ Strong\ Warm\ San$

$x_2 = \langle Sunny\ Warm\ High\ Strong\ Warm\ Same \rangle, +$ $h_2 = \langle Sunny\ Warm\ ?\ Strong\ Warm\ Same \rangle$

$x_3 = \langle Rainy\ Cold\ High\ Strong\ Warm\ Change \rangle, -$ $h_3 = \langle Sunny\ Warm\ ?\ Strong\ Warm\ Same \rangle$

$x_4 = \langle Sunny\ Warm\ High\ Strong\ Cool\ Change \rangle, +$ $h_4 = \langle Sunny\ Warm\ ?\ Strong\ ?\ ? \rangle$

Problems will arise because the choice of the first most specific hypothesis depends on the choice of the first instance and is therefore arbitrary; because there is no proof of learning; because of the possible inconsistency of the training data set.

The Version Space VS $_{H,D}$ is the subset of hypotheses from H consistent with training examples D.

The List-Then-Eliminate algorithm:

For each training example from the list of all hypotheses (Version Space), we remove the hypothesis not fitting.

Candidate Elimination Algorithm:

G: **The General Boundary** of version space is the set of its maximally general members.

S: **The Specific Boundary** of version space is all maximally specific members.

All members lie in between.

For each training example d, **if d is positive,** remove from G any inconsistent hypothesis with d.

For each hypothesis s in S, not consistent with d, remove s from S, and add to S all minimal generalizations h of s such that **h is consistent with d, some member of G being more general than h,**

Finally, remove from S any hypothesis more general than another hypothesis in S.

For each training example d, **if d is negative,** remove from S any inconsistent hypothesis with d.

For each hypothesis g in G not consistent with d, remove g from G

Add to G all min specializations h of G such that h is consistent with d and some member of S is more specific than h.

Remove from G any hypothesis that is less general than another hypothesis in G.

This algorithm is a least commitment algorithm; considering all rules to be viable.

Initially G is set <?, ?> and S is set <-, ->.

Consider the examples in the table below:

Example	Sky	AirTemp	Humdty	Wind	Water	Forecast	Class
Ex1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
Ex2	Sunny	Warm	High	Strong	Warm	Same	Yes
Ex3	Rainy	Cold	High	Strong	Warm	Change	No
Ex4	Sunny	Warm	High	Strong	Cool	Change	Yes

Say that our task is to learn rules that will classify a day as being a good day to play a water sport.

We initialize our S and G sets as follows:

G = {<?, ?, ?, ?, ?, ?>}
S = {<-, -, -, -, -, ->}

The first example is positive, so our S set is too specific. S and G are now as follows:

G = {<?, ?, ?, ?, ?, ?>}
S = {<Sunny, Warm, Normal, Strong, Warm, Same>}

The second example is also positive, so we consider our S set again. It is not consistent with the new example, so we consider all least generalizations that will include Ex2:

G = {<?, ?, ?, ?, ?, ?>}
S = {<Sunny, Warm, ?, Strong, Warm, Same>}

The third example is negative. Here we need to modify the G, because it is too general. We add to G all least specializations that exclude Ex3:

G = {<Sunny, ?, ?, ?, ?, ?>, <?, Warm, ?, ?, ?, ?>, <?, ?, ?, ?, ?, Same>}
S = {<Sunny, Warm, ?, Strong, Warm, Same>}

Note that we didn't include <?, ?, ?, ?, Cool, ?> because that would not be consistent with S.

Finally, we are given the fourth example, which is positive. First, we remove from G anything inconsistent with this example:

G = {<Sunny, ?, ?, ?, ?, ?>, <?, Warm, ?, ?, ?, ?>}

We then remove the single hypothesis from S because it is inconsistent with the new positive example. We minimally generalize that hypothesis and put the new hypotheses back into S:

S = {<Sunny, Warm, ?, Strong, ?, ?>}

At this point, there are no more examples, so the algorithm terminates. The rules in the Version Space are now:

<Sunny, ?, ?, ?, ?, ?>, <?, Warm, ?, ?, ?, ?>	[the G set]
<Sunny, Warm, ?, ?, ?, ?>	[everything in between]
<Sunny, Warm, ?, Strong, ?, ?>	[the S set]

Inductive Bias:

Let the concept learning algorithm L, instances X and target c, training examples D, where L(x,D) is the classification of the instance after training.

The inductive bias is the minimal **set of assertions B** such that for any c and D,

$$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)]$$

where $A \vdash B$ means A logically entails B

Decision Tree Learning: as seen in CS472, used if target is discrete valued

```
[833+,167-] .83+ .17-
Fetal_Presentation = 1: [822+,116-] .88+ .12-
| Previous_Csection = 0: [767+,81-] .90+ .10-
| | Primiparous = 0: [399+,13-] .97+ .03-
| | Primiparous = 1: [368+,68-] .84+ .16-
| | | Fetal_Distress = 0: [334+,47-] .88+ .12-
| | | | Birth_Weight < 3349: [201+,10.6-] .95+ .05
| | | | Birth_Weight >= 3349: [133+,36.4-] .78+ .2
| | | Fetal_Distress = 1: [34+,21-] .62+ .38-
| Previous_Csection = 1: [55+,35-] .61+ .39-
Fetal_Presentation = 2: [3+,29-] .11+ .89-
Fetal_Presentation = 3: [8+,22-] .27+ .73-
```

Sample Statistics Above

The entropy measures the impurity of S: $Entropy(S) = \sum p.\log_2 1/p$

The **Information Gain** = $E(S) - (numberofcases_i / total).Ei - (numberofcases_j / total).Ej$

Over fitting: Hypothesis overfits training data if an alternative hypothesis is such that:

Error train(h)<error train(h') and error D(h) > error D(h')

To avoid over fitting we could stop growing when data split is not statistically significant, or though post-pruning **such that** size(tree)+size(misclassification(tree)) is minimized.

Reduced Error Pruning, Post Pruning.

Converting a Tree to Rules.

XI. Interactive Data Mining (haiku system)

We use a symbolic genetic algorithm to evolve the rules; this produces terms to

describe the underlying data of the form:

This algorithm would be used to extract informational content about a website based upon its

design.

A set of rules would be built based on optimal websites of type: commercial, banking, rela estate….

Then these rules would be gathered in logic clauses just like it is being done for semantics contents and help sort out contents of websites.

IF backgorund*colour* = red & *text*= black & *size* < 3 THEN label = commercial

Once a set of these rules is gathered, it is easiest to eliminate the redundant and false ones using seen-above machine learning techniques based on enthropy.

We require a training dataset and a testing data set.

We may also start with a number of random rules, and evolve the population through subsequent generations based on how well each rule performs on the dataset: how accurate it is, and how many false positives and negatives it produces, and its coverage of the data. The genetic algorithm aims to optimise an objective function, and manipulation of this function allows us to explore different areas of the search space. For example, we can strongly penalise rules that give false positive results, and achieve a different type of description than rules that may be more general and have greater coverage, but make a few more mistakes. Each rule is analysed in terms of the objective function and given a score, its fitness. The fittest rules are then taken as the basis for the next population, and new rules created.

Rules are created by "breeding" good rules together, which combine their attributes. A new rule is created by cutting two parent rules and joining the different halves; it then inherits characteristics from both parents.

XII: Google Search Technology:

1- Overview

According to Google, human maintained lists (Yahoo's) cover popular topics effectively but are
subjective, expensive to build and maintain, slow to improve, and cannot cover all esoteric topic; while at the other end of the spectrum automated search engines that rely on keyword matching usually return too many low quality matches.

Search Engines face issues of storage space (indices and documents) and response to huge volumes of queries.

Google uses the link structure of the Web to calculate a quality ranking for each web page. This ranking
is called PageRank. Second, Google utilizes link to improve search results.

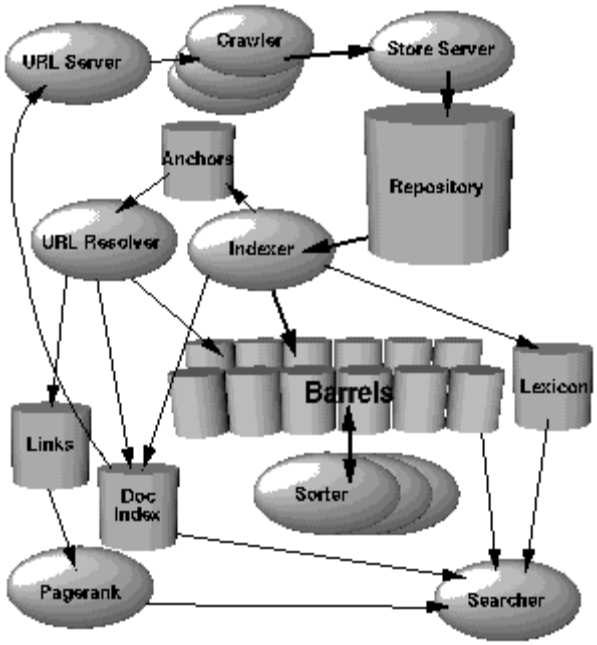
A page title and link prioritarization are good enough for most searches.

PageRank extends the counting of all other links emanating from a different wep page, by not counting links from all pages equally, and by normalizing by the number of links on a page.
The text of the links themselves is put at use. Since it may describe even better than the page itself, its contents and since it may point to documents that may not be crawled.

It is Google's strategy to return not only pages which best match the query but also which are consistent with the user's intention.

Internal meta information may not be reliable; however external meta-information(reputation, update frequency, popularity).

2- Architecture



©

URLserver => lists of URLs to be fetched to the
Crawlers.

Store Server indexes fetched pages, compresses them in the Repository.

Clearly our main technology of semantics parsing would occur in the Indexer. Google uncompresses the pages and records words hits on the page(word, position in document, an approximation of font
size, and capitalization).

The links are stored in an **anchor file. The** URL Resolver traces back to the DOCId's of the documents pointed to by the link.

References:

(1) Contents Analysis: Mike Palmquist of the Department of English at Colorado State University

(2) Supporting serendipity: Using ambient intelligence to augment user exploration for data mining and web browsing

Russel Beale, January 29th 2007

Advanced Interaction group, School of Computer Science, University of Birmingham

(3) A survey of modern knowledge modeling techniques: state of the art

Vladan Devedzic, Department of Information Systems, University of Belgrade

(4) Using text mining to infer semantic attributes for retail data mining

Accenture Technology Labs, Rayid GhanWeand Andrew E. Fano

(5) Textbook: Machine Learning. T. Mitchell. McGraw Hill, 1997.

(6)The Anatomy of a Large-Scale Hypertextual: Web Search Engine Sergey Brin and Lawrence Page

(7) Many-Layered Learning: Paul E. Utgoff, Department of Computer Science, University of Massachusetts and al.

Source Code:

Thesaurus Class:
This Class exhaustively defines one word, in a sentence, by its attributes.

```
#!/usr/bin/perl
package Thesaurus;

sub new {
    my ($class) = @_;
    my $self = {
        _mainEntry => undef,
```

```
#from the exhaustive list of types:  pronoun, adjective, article,
preposition, conjunction
        _type  =>   undef,
#irregular verb present
        _irregular_verb_present  => undef,
#irregular verb
          _regular_form   =>  undef,
          _preterit     =>   undef,
          _plus_perfect       =>   undef,
#number
         _number   => undef,


        _partofSpeech  => undef,
Taken from the encyclopedia, for thr word arts: see also arts(movies)
arts(theatre)…
        _context_word   => undef,
These are arrays which wiil later be split; correctly identifying
synonims, context, antonyms, part of speech and relations to other
objects in the sentence will allow to extract as minimal the
information as possible.
        _synonims   => undef,
        _antonyms   => undef,


#if verb
        _conjugation2   => undef,
        _prefix        => undef,
        _synonims_verb  => undef,
        _conjugation_forms   =>  undef,
        _related_forms    =>   undef,
        Example sentences allow to create analogies to the currently
       parsed  sentence.
        _sentence_verb_without_object => undef,
        _sentence_verb_with_object => undef,
        _sentence_for_noun => undef,
        _sentence_for_adjective => undef,

#if adjective


#if adverb



#if noun

#logic of sentence building#
#linked and type of link
        _linked_noun => undef,

        _linked_verb => undef,

        _linked_adverb => undef,

        _linked_adjective => undef,
```

```
        _linked_capital_noun => undef,

This allows to build the objects mentionned above: verb(object,chair)
#context:
#if verb


#if adjective


#if adverb


#if noun
#in See also:      noun(category),definition
           _see_also_noun  => undef,
#in ...may mean:   noun(category),definition,.....
if the noun is made of 2 capital words, it is a name
           _may_mean_noun  => undef,

#in ... can refer to:
           _can_refer_noun  => undef,
#may also be used for :  noun(category),definition
           _may_also_be_used_for  => undef,


#Global Context info:
#All highlighted words to context definition: as well as their category
in <H2>
           _context_definition  => undef,


    };
    bless $self, $class;
    return $self;
}
sub mainEntry {
    my ( $self, $mainEntry ) = @_;
    $self->{_mainEntry} = $mainEntry if defined($mainEntry);
    return $self->{_mainEntry};
}

#accessor method for Person last name
sub type {
    my ( $self, $type ) = @_;
    $self->{_type} = $type if defined($type);
    return $self->{_type};
}

sub irregular_verb_present {
    my ( $self, $irregular_verb_present ) = @_;
    $self->{_irregular_verb_present} = $irregular_verb_present if
defined($irregular_verb_present);
    return $self->{_irregular_verb_present};
}

sub regular_form {
```

```
    my ( $self, $regular_form ) = @_;
    $self->{_regular_form} = $regular_form if defined($regular_form);
    return $self->{_regular_form};
}
sub preterit {
    my ( $self, $preterit ) = @_;
    $self->{_preterit} = $preterit if defined($preterit);
    return $self->{_preterit};
}

sub plus_perfect {
    my ( $self, $plus_perfect ) = @_;
    $self->{_plus_perfect} = $plus_perfect if defined($plus_perfect);
    return $self->{_plus_perfect};
}
sub number {
    my ( $self, $number ) = @_;
    $self->{_number} = $number if defined($number);
    return $self->{_number};
}


sub partofSpeech {
    my ( $self, $partofSpeech ) = @_;
    $self->{_partofSpeech} = $partofSpeech if defined($partofSpeech);
    return $self->{_partofSpeech};
}



#accessor method for Person social security number
sub context_word {
    my ( $self, $context_word ) = @_;
    $self->{_context_word} = $context_word if defined($context_word);
    return $self->{_context_word};
}

sub synonims {
    my ( $self, $synonims ) = @_;
    $self->{_synonims} = $synonims if defined($synonims);
    return $self->{_synonims};
}

sub antonyms {
    my ( $self, $antonyms ) = @_;
    $self->{_antonyms} = $antonyms if defined($antonyms);
    return $self->{_antonyms};
}

sub conjugation2 {
    my ( $self, $conjugation2 ) = @_;
    $self->{_conjugation2} = $conjugation2 if defined($conjugation2);
    return $self->{_conjugation2};
}

sub prefix {
    my ( $self, $prefix ) = @_;
```

```
    $self->{_prefix} = $prefix if defined($prefix);
    return $self->{_prefix};
}
sub synonims_verb {
    my ( $self, $synonims_verb ) = @_;
    $self->{_synonims_verb} = $synonims_verb if
defined($synonims_verb);
    return $self->{_synonims_verb};
}
sub conjugation_forms {
    my ( $self, $conjugation_forms ) = @_;
    $self->{_conjugation_forms} = $conjugation_forms if
defined($conjugation_forms);
    return $self->{_conjugation_forms};
}
sub related_forms {
    my ( $self, $related_forms ) = @_;
    $self->{_related_forms} = $related_forms if
defined($related_forms);
    return $self->{_related_forms};
}

sub sentence_verb_without_object {
    my ( $self, $sentence_verb_without_object ) = @_;
    $self->{_sentence_verb_without_object} =
$sentence_verb_without_object if
defined($sentence_verb_without_object);
    return $self->{_sentence_verb_without_object};
}
sub sentence_verb_with_object  {
    my ( $self, $sentence_verb_with_object ) = @_;
    $self->{_sentence_verb_with_object} = $sentence_verb_with_object if
defined($sentence_verb_with_object);
    return $self->{_sentence_verb_with_object};
}
sub sentence_for_noun  {
    my ( $self, $sentence_for_noun ) = @_;
    $self->{_sentence_for_noun} = $sentence_for_noun if
defined($sentence_for_noun);
    return $self->{_sentence_for_noun};
}
sub sentence_for_adjective  {
    my ( $self, $sentence_for_adjective ) = @_;
    $self->{_sentence_for_adjective} = $sentence_for_adjective if
defined($sentence_for_adjective);
    return $self->{_sentence_for_adjective};
}

sub linked_noun  {
    my ( $self, $linked_noun ) = @_;
    $self->{_linked_noun} = $linked_noun if defined($linked_noun);
    return $self->{_linked_noun};
}
sub linked_verb  {
    my ( $self, $linked_verb ) = @_;
    $self->{_linked_verb} = $linked_verb if defined($linked_verb);
    return $self->{_linked_verb};
```

```
}
sub linked_adverb  {
    my ( $self, $linked_adverb ) = @_;
    $self->{_linked_adverb} = $linked_adverb if
defined($linked_adverb);
    return $self->{_linked_adverb};
}
sub linked_adjective  {
    my ( $self, $linked_adjective ) = @_;
    $self->{_linked_adjective} = $linked_adjective if
defined($linked_adjective);
    return $self->{_linked_adjective};
}
sub linked_capital_noun  {
    my ( $self, $linked_capital_noun ) = @_;
    $self->{_linked_capital_noun} = $linked_capital_noun if
defined($linked_capital_noun);
    return $self->{_linked_capital_noun};
}
sub see_also_noun  {
    my ( $self, $see_also_noun ) = @_;
    $self->{_see_also_noun} = $see_also_noun if
defined($see_also_noun);
    return $self->{_see_also_noun};
}
sub may_mean_noun  {
    my ( $self, $may_mean_noun ) = @_;
    $self->{_may_mean_noun} = $may_mean_noun if
defined($may_mean_noun);
    return $self->{_may_mean_noun};
}
sub can_refer_noun  {
    my ( $self, $can_refer_noun ) = @_;
    $self->{_can_refer_noun} = $can_refer_noun if
defined($can_refer_noun);
    return $self->{_can_refer_noun};
}
sub may_also_be_used_for  {
    my ( $self, $may_also_be_used_for ) = @_;
    $self->{_may_also_be_used_for} = $may_also_be_used_for if
defined($may_also_be_used_for);
    return $self->{_may_also_be_used_for};
}
sub context_definition  {
    my ( $self, $context_definition ) = @_;
    $self->{_context_definition} = $context_definition if
defined($context_definition);
    return $self->{_context_definition};
}

sub print {
    my ($self) = @_;

    #print Person info
    printf( "%s %s %s %s\n", $self->mainEntry, $self->partofSpeech,
$self->definition, $self->synonims,$self->antonyms  );
}
```

```
1;
```

SentenceParser.pl

```
#!/usr/local/bin/perl

GLOBAL VARIABLES DECLARATION; Use of our rules class theasaurus
use Tk;
use Thesaurus;
# Main Window
my $mw = new MainWindow;
my @thesaurus_array;
my $value = 0;
my $currently_parsing="";
my $type_determined;
my $position_in_sentence;

#Global Variables
my $age = 10;
my $occupied = 1;
my $textarea1 = $mw -> Frame();
my $textarea2 = $mw -> Frame();
my $textarea3 = $mw -> Frame();
my $textarea4 = $mw -> Frame();
my $textarea5 = $mw -> Frame();
my $textarea6 = $mw -> Frame();
my $textarea7 = $mw -> Frame();
my $txt_dictionary = $textarea1 -> Text(-width=>50, -height=>20);
my $txt_thesaurus = $textarea2 -> Text(-width=>50, -height=>20);
my $txt_sentence_input = $textarea3 -> Text(-width=>10, -height=>20);
my $txt_sentence_compiled = $textarea4 -> Text(-width=>10, -
height=>20);
my $txt_prolog = $textarea5 -> Text(-width=>10, -height=>20);
my $txt_neural = $textarea6 -> Text(-width=>10, -height=>20);
my $txt_context = $textarea7 -> Text(-width=>10, -height=>20);
Different text areas allow to visualize more easily the results in the
API.
The parsing of the dictionary input, thesaurus input, encyclopedia
input, the logic clauses built, the contet derived.

#Declare that there is a menu
my $mbar = $mw -> Menu();
$mw -> configure(-menu => $mbar);

#The Main Buttons
my $file = $mbar -> cascade(-label=>"File", -underline=>0, -tearoff =>
0);
my $others = $mbar -> cascade(-label =>"Others", -underline=>0, -
tearoff => 0);
```

```
my $help = $mbar -> cascade(-label =>"Help", -underline=>0, -tearoff =>
0);

## File Menu ##
$file -> command(-label => "New", -underline=>0,
            -command=>sub { $txt -> delete('1.0','end');} );
$file -> checkbutton(-label =>"Open", -underline => 0,
            -command => [\&menuClicked, "Open"]);
$file -> command(-label =>"Save", -underline => 0,
            -command => [\&menuClicked, "Save"]);
$file -> separator();
$file -> command(-label =>"Exit", -underline => 1,
            -command => sub { exit } );

## Others Menu ##
my $insert = $others -> cascade(-label =>"Insert", -underline => 0, -
tearoff => 0);
$insert -> command(-label =>"Name",
      -command => sub { $txt->insert('end',"Name : Binny V A\n");});
$insert -> command(-label =>"Website", -command=>sub {
      $txt->insert('end',"Website :
http://www.geocities.com/binnyva/\n");});
$insert -> command(-label =>"Email",
      -command=> sub {$txt->insert('end',"E-Mail :
binnyva\@hotmail.com\n");});
$others -> command(-label =>"Insert All", -underline => 7,
      -command => sub { $txt->insert('end',"Name : Binny V A
Website : http://www.geocities.com/binnyva/
E-Mail : binnyva\@hotmail.com");
      });

## Help ##
$help -> command(-label =>"About", -command => sub {
      $txt->delete('1.0','end');
      $txt->insert('end',
      "About
----------
This script was created to make a menu for a\nPerl/Tk tutorial.
Made by Raphael Rubin
Website : http://www.raphael-rubin.com
E-Mail : rr284\@cornell.edu"); });
#GUI Building Area
my $frm_name = $mw -> Frame();
my $lab = $frm_name -> Label(-text=>"Name:");
my $ent = $frm_name -> Entry();
#Age
my $scl = $mw -> Scale(-label=>"Age :",
       -orient=>'v',          -digit=>1,
       -from=>10,             -to=>50,
       -variable=>\$age,      -tickinterval=>10);

#Jobs
my $frm_job = $mw -> Frame();
my $chk = $frm_job -> Checkbutton(-text=>"Occupied",
      -variable=>\$occupied);
$chk -> deselect();
my $lst = $frm_job -> Listbox(-selectmode=>'single');
```

```
#Adding jobs
$lst -> insert('end',"Student","Teacher","Clerk","Business Man",
      "Militry Personal","Computer Expert","Others");

#Gender
my $frm_gender = $mw -> Frame();
my $lbl_gender = $frm_gender -> Label(-text=>"Sex ");
my $rdb_m = $frm_gender -> Radiobutton(-text=>"Male",
            -value=>"Male",  -variable=>\$gender);
my $rdb_f = $frm_gender -> Radiobutton(-text=>"Female",
            -value=>"Female",-variable=>\$gender);

my $but = $mw -> Button(-text=>"Push Me", -command =>\&push_button);

#Text Area
my $srl_y = $textarea1 -> Scrollbar(-orient=>'v',-command=>[yview =>
$txt]);
my $srl_x = $textarea1 -> Scrollbar(-orient=>'h',-command=>[xview =>
$txt]);
$txt_dictionary -> configure(-yscrollcommand=>['set', $srl_y],
            -xscrollcommand=>['set',$srl_x]);
my $srl_y1 = $textarea2 -> Scrollbar(-orient=>'v',-command=>[yview =>
$txt]);
my $srl_x1 = $textarea2 -> Scrollbar(-orient=>'h',-command=>[xview =>
$txt]);

$txt_thesaurus -> configure(-yscrollcommand=>['set', $srl_y1],
            -xscrollcommand=>['set',$srl_x1]);
my $srl_y2 = $textarea3 -> Scrollbar(-orient=>'v',-command=>[yview =>
$txt]);
my $srl_x2 = $textarea3 -> Scrollbar(-orient=>'h',-command=>[xview =>
$txt]);
$txt_sentence_input  -> configure(-yscrollcommand=>['set', $srl_y],
            -xscrollcommand=>['set',$srl_x]);
my $srl_y3 = $textarea4 -> Scrollbar(-orient=>'v',-command=>[yview =>
$txt]);
my $srl_x3 = $textarea4 -> Scrollbar(-orient=>'h',-command=>[xview =>
$txt]);
$txt_sentence_compiled  -> configure(-yscrollcommand=>['set', $srl_y],
            -xscrollcommand=>['set',$srl_x]);
my $srl_y4 = $textarea5 -> Scrollbar(-orient=>'v',-command=>[yview =>
$txt]);
my $srl_x4 = $textarea5 -> Scrollbar(-orient=>'h',-command=>[xview =>
$txt]);
$txt_prolog  -> configure(-yscrollcommand=>['set', $srl_y],
            -xscrollcommand=>['set',$srl_x]);
my $srl_y5 = $textarea6 -> Scrollbar(-orient=>'v',-command=>[yview =>
$txt]);
my $srl_x5 = $textarea6 -> Scrollbar(-orient=>'h',-command=>[xview =>
$txt]);
$txt_neural  -> configure(-yscrollcommand=>['set', $srl_y],
            -xscrollcommand=>['set',$srl_x]);
my $srl_y6 = $textarea7 -> Scrollbar(-orient=>'v',-command=>[yview =>
$txt]);
my $srl_x6 = $textarea7 -> Scrollbar(-orient=>'h',-command=>[xview =>
$txt]);
```

```
$txt_context  -> configure(-yscrollcommand=>['set', $srl_y],
            -xscrollcommand=>['set',$srl_x]);

#Geometry Management
$lab -> grid(-row=>1,-column=>1);
$ent -> grid(-row=>1,-column=>2);
$scl -> grid(-row=>2,-column=>1);
$frm_name -> grid(-row=>1,-column=>1,-columnspan=>2);

$chk -> grid(-row=>1,-column=>1,-sticky=>'w');
$lst -> grid(-row=>2,-column=>1);
$frm_job -> grid(-row=>2,-column=>2);

$lbl_gender -> grid(-row=>1,-column=>1);
$rdb_m -> grid(-row=>1,-column=>2);
$rdb_f -> grid(-row=>1,-column=>3);
$frm_gender -> grid(-row=>3,-column=>1,-columnspan=>2);

$but -> grid(-row=>4,-column=>1,-columnspan=>2);

$txt_dictionary -> grid(-row=>1,-column=>1);
$txt_thesaurus -> grid(-row=>1,-column=>1);
$txt_sentence_input -> grid(-row=>1,-column=>1);
$txt_sentence_compiled -> grid(-row=>1,-column=>1);
$txt_prolog -> grid(-row=>1,-column=>1);
$txt_neural -> grid(-row=>1,-column=>1);
$txt_context -> grid(-row=>1,-column=>1);
Layout of the dialog panels in the grid.

$srl_y -> grid(-row=>1,-column=>2,-sticky=>"ns");
$srl_x -> grid(-row=>2,-column=>1,-sticky=>"ew");
$srl_y1 -> grid(-row=>1,-column=>2,-sticky=>"ns");
$srl_x1 -> grid(-row=>2,-column=>1,-sticky=>"ew");
$srl_y2 -> grid(-row=>1,-column=>2,-sticky=>"ns");
$srl_x2 -> grid(-row=>2,-column=>1,-sticky=>"ew");
$srl_y3 -> grid(-row=>1,-column=>2,-sticky=>"ns");
$srl_x3 -> grid(-row=>2,-column=>1,-sticky=>"ew");
$srl_y4 -> grid(-row=>1,-column=>2,-sticky=>"ns");
$srl_x4 -> grid(-row=>2,-column=>1,-sticky=>"ew");
$srl_y5 -> grid(-row=>1,-column=>2,-sticky=>"ns");
$srl_x5 -> grid(-row=>2,-column=>1,-sticky=>"ew");
$srl_y6 -> grid(-row=>1,-column=>2,-sticky=>"ns");
$srl_x6 -> grid(-row=>2,-column=>1,-sticky=>"ew");

$textarea1 -> grid(-row=>5,-column=>1,-columnspan=>2);
$textarea2 -> grid(-row=>5,-column=>5,-columnspan=>2);
$textarea3 -> grid(-row=>2,-column=>10,-columnspan=>2);
$textarea4 -> grid(-row=>2,-column=>15,-columnspan=>2);
$textarea5 -> grid(-row=>2,-column=>20,-columnspan=>2);
$textarea6 -> grid(-row=>2,-column=>25,-columnspan=>2);
$textarea7 -> grid(-row=>2,-column=>30,-columnspan=>2);


MainLoop;
## Functions
#This function will be executed when the button is pushed
sub push_button {
```

```
my $url_dictionary = "http://dictionary.reference.com/browse/";
my $url_thesaurus = "http://thesaurus.reference.com/browse/";
my $url_encyclopedia = "http://www.reference.com/search?q=";
use DBI;
my $count;

These are all hashes used for parsing.
my %conjugation;
my %title;
my %part_of_speech;
my %exemplary_sentence;
my %section_label;
my %synonims_label;
my %verb_without_object;
my %verb_with_object;
my %ad_jec_tive;
my %no_un;
my %prono_un;
my %con_junction;
my %bof_subhead;
my %prefix_verb;
$prefix_verb{count} = 0;

#end on intialization

my $value = 0;
my $print1 = "false";
#this is actually ARGV[2]
my $word = "drive";
use AI::Proplog;
use Data::Dumper;

Down below is the exhaustive list of all terms used in the English that
are easily identifiable in a sentence, that allow to distinguish its
elements, to parse it.

my @subjective_personal_pronoun;
push(@subjective_personal_pronoun,"I", "you", "she", "he", "it", "we",
"you", "they");
my @objective_personal_pronoun;
push(@objective_personal_pronoun,"me", "you", "her", "him", "it", "us",
"you" , "them");
my @possessive_personal_pronoun;
push(@possessive_personal_pronoun,"mine", "yours" ,"hers", "his",
"its", "ours", "theirs");
my @demonstrative_personal_pronoun;
push(@demonstrative_personal_pronoun,"this", "that", "these", "those");
my @interrogative_pronoun;
push(@interrogative_pronoun, "who", "whom", "which", "what", "whoever",
"whomever", "whichever", "whatever");
my @relative_pronoun;
push(@relative_pronoun,"what", "who", "whom", "that", "which",
"whoever", "whomever", "whichever");
my @indefinite_pronoun;
push(@indefinite_pronoun,"enough", "fewer", "less", "little", "much",
"several", "most", "both", "every", "either", "neither", "nor", "all",
"another", "any", "anybody", "anyone", "something", "anything",
```

```
"everything", "each", "everybody", "everyone", "everything", "few",
"many", "nobody", "none", "one" ,"several", "some", "somebody",
"someone");
my @reflexive_pronoun;
push(@reflexive_pronoun,"myself", "yourself", "herself", "himself",
"itself", "ourselves", "yourselves", "themselves");
#difference with a reflexive_pronoun is that even taken out of a
sentence, it still makes sense
my @intensive_pronoun;
push(@intensive_pronoun,"myself", "yourself", "herself", "himself",
"itself", "ourselves", "yourselves", "themselves");
#composite indicates that composite words and split is a comma
my @reciprocal_pronoun;
push(@reciprocal_pronoun,"each", "other","one", "another");
my @article;
push(@article,"a", "an", "the", "some", "any");
my @adjective_size;
push(@adjective_size,"big", "little", "enormous", "big", "small",
"long", "tall", "short");
my @adjective_shape;
push(@adjective_shape,"round", "circle", "triangle", "rectangle",
"square", "oval");
my @adjective_color;
push (@adjective_color,"blue", "red", "green", "brown", "yellow",
"black", "white");
my @adjective_shape;
push (@adjective_shape,"big", "little", "enormous", "big", "small",
"long", "tall","short");
my @adjective_color;
push(@adjective_color,"big", "little", "enormous", "big", "small",
"long", "tall", "short");


#just multiply one * thousand to get the right amount
my %number = (
one =>1,
two =>2,
three =>3,
four =>4,
five =>5,
six =>6,
seven =>7,
height =>8,
nine =>9,
ten =>10,
eleven =>11,
twelve =>12,
thirteen =>13,
fourteen =>14,
fifteen =>15,
sixteen =>16,
seventeen =>17,
eighteen =>18,
nineteen =>19,
twenty =>20,
thirty =>30,
fourty =>40,
```

```
fifty =>50,
sixty =>60,
seventy =>70,
eighty =>80,
ninety =>90,
hundred =>100,
thousand =>1000,
rule =>"*ty one,*ty two,*ty three,*ty four,*ty five,*ty six,*ty
seven,*ty eight,*ty nine"
);


my %past_participled_adjective = (
      rule =>"verb ed"
);
my @possessive_adjective;
push(@possessive_adjective,"my", "your", "his", "her", "its", "our",
"their");

my @demonstrative_adjective;
push(@demonstrative_adjective,"this", "these", "that", "those");
my @interrogative_adjective;
push(@interrogative_adjective,"which", "what", "whose");
my @indefinite_adjective;
push(@indefinite_adjective,"many", "all", "any", "much");
my @possessive_adjective;
push(@possessive_adjective,"my your his her its their our");
#if ends with y => becomes ier
we mean that exist extensions ier, est, and prefix more, most...
my @superlative_adjective;
push(@psuperlative_adjective,"adjective", "st","most", "adjective");
my @comparative_adjective;
push(@comparative_adjective,"composite","adjective", "er",
"than","%","adjective", "er","%","adjective" ,"ier","%","adjective"
,"ier", "than","%","more", "adjective","%","more", "adjective",
"than","%","as", "adjective", "as");
my %comparative_irregular_adjective = (
good =>"better",
bad =>"worse",
little =>"less",
much =>"more",
many =>"more",
some =>"more",
far =>"further"
);
my %superlative_irregular_adjective = (
good =>"best",
bad =>"worst",
little =>"least",
much =>"most",
many =>"most",
some =>"most",
far =>"furthest"
);
```

```
my @irregular_verb_present;
push(@irregular_verb_present,"is", "are", "am");
#Infinitive Simple Past Past Participle
my %irregular_verb = (
arise =>"arose,arisen",
awake =>    "awakened awoke,awakened awoken",
backslide =>      "backslid,backslidden backslid",
be => "was were,been",
bear=>      "bore,born borne",
beat=>      "beat,beaten beat",
become=>"became,become",
begin =>"began,begun",
bend=>"bent,bent",
bet => "bet betted,bet betted",
bid_farewell =>"bid bade,bidden",
bid_amount=>"bid,bid",
bind=>"bound,bound",
bite=>"bit,bitten",
bleed =>"bled,bled",
blow =>"blew,blown",
break =>"broke,broken",
breed=>"bred,bred",
bring =>"brought,brought",
broadcast=>"broadcast broadcasted,broadcast broadcasted",
browbeat => "browbeat,browbeaten browbeat",
build=>"built,built",
burn=>"burned burnt,burned burnt",
burst=>"burst,burst",
bust=>"busted bust,busted bust",
buy=>"bought,bought",
cast=>"cast,cast",
catch =>"caught,caught",
choose=> "chose,chosen",
cling =>"clung,clung",
clothe=>"clothed clad,clothed clad",
come=>"came,come",
cost=>"cost,cost",
creep=>"crept,crept",
crossbreed=>"crossbred,crossbred",
cut=>"cut,cut",
daydream =>"daydreamed daydreamt,daydreamed daydreamt",
deal=>"dealt,dealt",
dig=>"dug,dug",
disprove=>"disproved,disproved disproven",
dive_jump_head_first =>"dove dived,dived",
dive_scuba_diving =>"dived dove,dived",
do =>"did,done",
dream=>"dreamed dreamt,dreamed dreamt",
drink=>"drank,drunk",
drive=>"drove,driven",
dwell=>"dwelt dwelled,dwelt dwelled",
eat=> "ate,eaten",
fall=>"fell,fallen",
feed =>"fed,fed",
feel =>"felt,felt",
fight =>"fought,fought",
find =>"found,found",
```

```
fit_change_size =>"fitted fit,fitted fit",
fit_right_size =>"fitted fit,fitted fit",
flee =>"fled,fled",
fling =>"flung,flung",
fly =>"flew,flown",
forbid =>"forbade,forbidden",
forecast =>"forecast,forecast",
forego =>"forewent,foregone",
foresee=>"foresaw,foreseen",
foretell =>"foretold,foretold",
forget =>"forgot,forgotten forgot",
forgive=>"forgave,forgiven",
forsake=>"forsook,forsaken",
freeze=>"froze,frozen",
frostbite=>"frostbit,frostbitten",
get=>" got,gotten got",
give=>"gave,given",
go=>"went,gone",
grind=>"ground,ground",
grow=>"grew,grown",
hand-feed =>"hand-fed,hand-fed",
handwrite =>"handwrote,handwritten",
hang =>"hung,hung",
have=>"had,had",
hear =>"heard,heard",
hew =>"hewed,hewn hewn",
hide =>"hid,hidden",
hit  =>"hit,hit",
hold =>"held,held",
hurt =>"hurt,hurt",
inbreed =>"inbred,inbred",
inlay =>"inlaid,inlaid",
input  =>"input inputted,input inputted",
interbreed =>"interbred,interbred",
interweave  =>"interwove interweaved,interwoven interweaved",
interwind =>"interwound,interwound",
jerry-build  =>"jerry-built,jerry-built",
keep  =>"kept,kept",
kneel =>"knelt kneeled,knelt kneeled",
knot =>"knitted knit,knitted knit",
know =>"knew,known",
lay  =>"laid,laid",
lead  =>"led,led",
lean  =>"leaned leant,leaned leant",
leap  =>"leaped leapt,leaped leapt",
learn =>"learned learnt,learned learnt",
leave =>"left,left",
lend =>"lent,lent",
let =>"let,let",
lie_down =>"lay,lain",
lie_truth =>"lied,lied",
light =>"lit lighted,lit lighted",
lip-read =>"lip-read,lip-read",
lose =>"lost,lost",
make =>"made,made",
mean =>"meant,meant",
meet =>"met,met",
```

```
miscast=>"miscast,miscast",
misdeal =>"misdealt,misdealt",
misdo =>" misdid,misdone",
mishear =>"misheard,misheard",
mislay =>"mislaid,mislaid",
mislead =>"misled,misled",
mislearn =>"mislearned mislearnt,mislearned mislearnt",
misread =>"misread,misread",
misset =>"misset,misset",
misspeak =>"misspoken,misspoken",
misspell =>"misspelled misspelt,misspelled misspelt",
misspend =>"misspent,misspent",
mistake =>"mistook,mistaken",
misteach  =>"mistaught,mistaught",
misunderstand =>"misunderstood,misunderstood",
miswrite =>"miswrote,miswritten",
mow =>"mowed,mowed mown",
offset =>"offset,offset",
outbid  =>"outbid,outbid",
outbreed =>"outbred,outbred",
outdo  =>"outdid,outdone",
outdraw  =>"outdrew,outdrawn",
outdrink =>"outdrank,outdrunk",
outdrive =>"outdrove,outdriven",
outfight =>"outfought,outfought",
outfly  =>"outflew,outflown",
outgrow =>"outgrew,outgrown",
outleap  =>"outleaped outleapt,outleaped outleapt",
outlie_truth =>"outlied,outlied",
outride =>"outrode,outridden",
outrun  =>"outran,outrun",
outsell=>"outsold,outsold",
outshine =>"outshined outshone,outshined outshone",
outshoot =>"outshot,outshot",
outsing =>"outsang,outsung",
outsit =>"outsat,outsat",
outsleep =>"outslept,outslept",
outssmell  =>"outsmelled outsmelt,outsmelled outsmelt",
outspeak  =>"outspoke,outspoken",
outspeed  =>"outsped,outsped",
outspend =>"outspent,outspent",
outswear =>"outswore,outsworn",
outswim =>"outswam,outswum",
outthink  =>"outthought,outthought",
outthrow =>"outthrew,outthrown",
outwrite =>"outwrote,outwritten",
overbid =>"overbid,overbid",
overbreed =>"overbred,overbred",
overbuild  =>"overbuilt,overbuilt",
overbuy  =>"overbought,overbought",
overcome =>"overcame,overcome",
overdo =>"overdid,overdone",
overdraw =>"overdrew,overdrawn",
overdrink =>"overdrank,overdrunk",
overeat =>"overate,overeaten",
overfeed =>"overfed,overfed",
overhang =>"overhung,overhung",
```

```
overhear =>"overheard,overheard",
overlay =>"overlaid,overlaid",
overpay =>"overpaid,overpaid",
override =>"overrode,overridden",
overrun =>"overran,overrun",
oversee =>"oversaw,overseen",
oversell =>"oversold,oversold",
oversew =>"oversewed,oversewn oversewed",
overshoot =>"overshot,overshot",
oversleep =>"overslept,overslept",
overspeak =>"overspoke,overspoken",
overspend =>"overspent,overspent",
overspill =>"overspilled overspilt,overspilled overspilt",
overtake =>"overtook,overtaken",
overthink =>"overthought,overthought",
overthrow =>"overthrew,overthrown",
overwind =>"overwound,overwound",
overwrite =>"overwrote,overwritten",
partake =>"partook,partaken",
pay =>"paid,paid",
plead =>"pleaded pled,pleaded pled",
prebuild =>"prebuilt,prebuilt",
predo =>"predid,predone",
premake =>"premade,premade",
prepay =>"prepaid,prepaid",
presell =>"presold,presold",
preset =>"preset,preset",
preshrink =>"preshrank,preshrunk",
proofread =>"proofread,proofread",
prove =>"proved,proven proved",
put =>"put,put",
quick-freeze =>"quick-froze,quick-frozen",
quit =>"quit quitted,quit quitted",
read =>"read,read",
reawake =>"reawoke,reawaken",
rebid =>"rebid,rebid",
rebind =>"rebound,rebound",
rebroadcast =>"rebroadcast rebroadcasted,rebroadcast rebroadcasted",
rebuild =>"rebuilt,rebuilt",
recast =>"recast,recast",
recut =>"recut,recut",
redeal =>"redealt,redealt",
redo  =>"redid,redone",
redraw  =>"redrew,redrawn",
refit_replace =>"refit refitted,refit refitted",
refit_retailor =>"refitted refit,refitted refit",
regrind =>"reground,reground",
regrow =>"regrew,regrown",
rehang =>"rehung,rehung",
rehear =>"reheard,reheard",
reknit =>"reknitted reknit,reknitted reknit",
relay_tiles =>"relaid,relaid",
relay_pass_along =>"relayed,relayed",
relearn =>"relearned relearnt,relearned relearnt",
relight =>"relit relighted,relit relighted",
remake =>"remade,remade",
repay =>"repaid,repaid",
```

```
reread =>"reread,reread",
rerun =>"reran,rerun",
resell =>"resold,resold",
resend =>"resent,resent",
reset =>"reset,reset",
resew =>"resewed,resewn resewed",
retake =>"retook,retaken",
reteach =>"retaught,retaught",
retear =>"retore,retorn",
retell =>"retold,retold",
rethink =>"rethought,rethought",
retread =>"retread,retread",
retrofit =>"retrofitted retrofit,retrofitted retrofit",
rewake =>"rewoke rewaked,rewaken rewaked",
rewear =>"rewore,reworn",
reweave =>"rewove reweaved,rewoven reweaved",
rewed =>"rewed rewedded,rewed rewedded",
rewet =>"rewet rewetted,rewet rewetted",
rewin =>"rewon,rewon",
rewind =>"rewound,rewound",
rewrite =>"rewrote      rewritten",
rid =>"rid,rid",
ride =>"rode,ridden",
ring =>"rang,rung",
rise =>"rose,risen",
roughcast =>"roughcast,roughcast",
run =>"ran,run",
sand-cast =>"sand-cast,sand-cast",
saw =>"sawed,sawed sawn",
say=>"said,said",
see=>"saw,seen",
seek=>"sought,sought",
sell=>"sold,sold",
send=>"sent,sent",
set=>"set,set",
sew=>"sewed,sewn sewed",
shake=>"shook,shaken",
shave=>"shaved,shaved shaven",
shear=>"sheared,sheared shorn",
shed=>"shed shed",
shine=>"shined shone,shined shone",
shit=>"shat,shitted",
shoot=>"shot,shot",
show =>"showed,shown showed",
shrink =>"shrank shrunk,shrunk",
shut =>"shut,shut",
sight-read =>"sight-read,sight-read",
sing=>"sang,sung",
sink =>"sank sunk,sunk",
sit =>"sat,sat",
slay_kill =>"slew slayed,slain slayed",
slay_amuse =>"slayed,slayed",
sleep =>"slept,slept",
slide=>"slid,slid",
sling=>"slung,slung",
slink=>"slinked slunk,slinked slunk",
slit=>"slit,slit",
```

```
smell=>"smelled smelt,smelled smelt",
sneak =>"sneaked snuck,sneaked snuck",
sow=>"sowed,sown sowed",
speak=>"spoke,spoken",
speed=>"sped speeded,sped speeded",
spell =>"spelled spelt,spelled spelt",
spend =>"spent,spent",
spill =>"spilled spilt,spilled spilt",
spin=>"spun,spun",
spit =>"spit spat,spit spat",
split =>"split,split",
spoil=>"spoiled spoilt,spoiled spoilt",
spoon-feed=>"spoon-fed,spoon-fed",
spread=>"spread,spread",
spring=>"sprang sprung,sprung",
stand =>"stood,stood",
steal=>"stole,stolen",
stick=>"stuck,stuck",
sting=>"stung,stung",
stink=>"stunk stank,stunk",
strew=>"strewed,strewn strewed",
stride=>"strode,stridden",
strike_delete =>"struck,stricken",
strike_hit =>"struck,struck stricken",
string=>"strung,strung",
strive=>"strove strived,striven strived",
sublet=>"sublet,sublet",
sunburn=>"sunburned sunburnt,sunburned sunburnt",
swear=>"swore,sworn",
sweat=>"sweat sweated,sweat sweated",
sweep=>"swept,swept",
swell=>"swelled,swollen swelled",
swim=>"swam,swum",
swing=>"swung,swung",
take=>"took,taken",
teach=>"taught,taught",
tear=>"tore,torn",
telecast=>"telecast,telecast",
tell=>"told,told",
test-drive =>"test-drove,test-driven",
test-fly =>"test-flew,test-flown",
think =>"thought,thought",
throw=>"threw,thrown",
thrust=>"thrust,thrust",
tread=>"trod,trodden trod",
typecast=>"typecast,typecast",
typeset=>"typeset,typeset",
typewrite=>"typewrote,typewritten",
unbend=>"unbent,unbent",
unbind=>"unbound,unbound",
unclothe =>"unclothed unclad,unclothed unclad",
underbid =>"underbid,underbid",
undercut=>"undercut,undercut",
underfeed=>"underfed,underfed",
undergo=>"underwent,undergone",
underlie=>"underlay,underlain",
undersell=>"undersold,undersold",
```

```
underspend=>"underspent,underspent",
understand=>"understood,understood",
undertake=>"undertook,undertaken",
underwrite=>"underwrote,underwritten",
undo=>"undid,undone",
unfreeze=>"unfroze,unfrozen",
unhang=>"unhung,unhung",
unhide=>"unhid,unhidden",
unknit=>"unknitted unknit,unknitted unknit",
unlearn=>"unlearned unlearnt,unlearned unlearnt",
unsew =>"unsewed,unsewn unsewed",
unsling =>"unslung,unslung",
unspin =>"unspun,unspun",
unstick =>"unstuck,unstuck",
unstring =>"unstrung,unstrung",
unweave =>"unwove unweaved,unwoven unweaved",
unwind =>"unwound,unwound",
uphold =>"upheld,upheld",
upset =>"upset,upset",
wake => "woke waked,woken waked",
waylay =>"waylaid,waylaid",
wear =>"wore,worn",
weave =>"wove weaved,woven weaved",
wed =>"wed wedded,wed wedded",
weep =>"wept wept",
wet =>"wet wetted,wet wetted",
whet => "whetted,whetted",
win =>"won,won",
wind =>"wound,wound",
withdraw =>"withdrew,withdrawn",
withhold =>"withheld,withheld",
withstand =>"withstood,withstood",
wring =>"wrung,wrung",
write =>"wrote,written"
);

my @pronoun;
push(@pronoun,"subjective_personal_pronoun",
"objective_personal_pronoun", "possessive_personal_pronoun",
"demonstrative_pronouns", "interrogative_pronouns",
"relative_pronouns", "indefinite_pronouns", "reflexive_pronouns",
"intensive_pronouns", "reciprocal_pronouns");

my @adjective;
push(@adjective,"past_participled_adjective", "possessive_adjective",
"demonstrative_adjective", "interrogative_adjective",
"indefinite_adjective", "possessive_adjective",
"superlative_adjective", "comparative_adjective",
"comparative_irregular_adjective", "superlative_irregular_adjective",
"adjective_size",  "adjective_shape", "adjective_color",
"adjective_shape", "adjective_color");

my @preposition;
push(@preposition,"about","above" ,"across", "after", "against",
"along", "among" ,"around", "at", "before", "behind", "below",
"beneath", "beside", "between", "beyond", "but", "by", "despite",
"down", "during", "except", "for", "from", "in", "inside", "into",
```

```
"like", "near", "of", "off", "on", "onto", "out", "outside", "over",
"past", "since", "through", "throughout", "till", "to", "toward",
"under", "underneath", "until", "up", "upon", "with", "within",
"without");

my @negation;
push(@negation,"not", "nt");

my @conjunction;
push(@conjunction,"coordinating_conjunction",
"subordinating_conjunction", "correlative_conjunction");

my @coordinating_conjunction;
push(@coordinating_conjunction,"and", "but", "or", "nor", "for", "so",
"yet");

my @subordinating_conjunction;
push(@subordinating_conjunction,"after", "although", "as", "because",
"before", "how", "if", "once", "since", "than", "that", "though",
"till", "until", "when", "where", "whether", "while");
my @correlative_conjunction;
push(@correlative_conjunction,"both","%", "and","%","either
or","%","neither nor","%","whether", "%", "or","%", "not only", "%",
"but also");


#just call the main categories, the subcategories make it more flexible

my @verb;
push(@verb, "irregular_verb_present", "irregular_verb");
my    %sentence_structure = (
        pronoun    => 110,
        adjective => 210   ,
        article => 310,
        verb  => 410 ,
        preposition => 510,
        conjunction => 610  ,
        number   => 710,
        noun    => 810
);
Above figure all parts of speech. Below, the boxes which serve object
correlation and will feed the prolog clauses and ultimately neural
networks.

my @array_string =( "object(adj,def)" ,
"object(noun,def)","object(number,
def)","object(pronoun,def)","subject(pronoun,def)"
,"verb(tense,def)","verb(sub,def)","verb(obj,def)", "verb(adverb,def)",
"verb(qualifier, def)","verb(concurrent, def)","verb(verb, def)"
,"verb(condition,def)","noun(adjective,def)","noun(noun,def)","noun(pro
noun,def)"  );


my @typical_commercial = ("Now", "Instead", "best", "miss", "lowest");
my $sentence= "IT'S a common juggling act on the doorstep: rummaging
for the house keys with one hand while balancing a bag of groceries
with the other";
```

```
$txt_sentence_input  -> insert('end',"$sentence\n");

package MyParser;
use base qw(HTML::Parser);
use LWP::Simple ();
use DBI;
#use >> to append
#open(MYFILE2, '>>business.txt') or print "die";
my $key;
my $prefix;
my @conjugation_table;
my @synonims_table;   #replace commas and white spaces with dots
#Always keep track of the pattern /s c/ in three caracters and if the
first is a a-zA-Z and the last also, skip
my @sentences_table;  #further decompose sentence into
my %related_forms;      #ad_jec_tive => ...,   no_un=> ..., adverb=>
...,
my $comment;
my $do_not_print= 0;

Parsing of the starting element for each identified part of speech

sub start_noun{
      my ($self,$tag,$attr, $attrseq, $origtext,$comment) = @_;
}
sub start_adjective{
}
sub start_adverb{
}

sub start_thesaurus{
      my ($self,$tag,$attr, $attrseq, $origtext,$comment) = @_;
}

sub start_encyclopedia{
      my ($self,$tagname,$attr, $attrseq, $origtext,$comment) = @_;
######################START################################
if ($tagname eq 'a') {
       if ($attr->{ title }  !~  ".com"  && $attr->{ title } !~ "http"
&& $attr->{ title }  !~  "arrow your search" ) {
#print "$attr->{ title }\n";

}
}


if  ($tagname eq "h2" ){
      print "H2 \n";
}

#   no    .com   and no attribute href


          #secondary holds values for
if ($origtext =~ "secondary" ){
$conjugation{count}+= 1;
      #print "text: $origtext\n";
```

```
  }

if ($origtext =~ "pg" ){
$part_of_speech{count}+= 1;
      #print "text: $origtext\n";
  }

if ($origtext =~ "sectionLabel" ){
$section_label{count}+= 1;
 $do_not_print= 0;
  }

if ($origtext =~ "class=\"dn\""){
$synonims_label{count}+= 1;
  }

if ($origtext =~ "ital-inline" ){
$exemplary_sentence{count}+= 1;
  }

if ($origtext =~ "rom-inline" ){
$do_not_print= 1;
$ad_jec_tive{count}=0;
  }

##########################END#############################
}



sub start {

  my $value;
      my ($self,$tag,$attr, $attrseq, $origtext,$comment) = @_;

if ( $currently_parsing eq "thes" ){
start_thesaurus($self,$tag,$attr, $attrseq, $origtext,$comment);
}

if ( $currently_parsing eq "encyclopedia" ){
start_encyclopedia($self,$tag,$attr, $attrseq, $origtext,$comment);
//these functions are called as the start() method only will be
launched by the Package MyParser.
}

if($type_determined eq "noun"){
start_noun($self,$tag,$attr, $attrseq, $origtext,$comment);
}
if($type_determined eq "adjective"){
start_adjective($self,$tag,$attr, $attrseq, $origtext,$comment);
}
if($type_determined eq "adverb"){
start_adverb($self,$tag,$attr, $attrseq, $origtext,$comment);
}
```

```
if($type_determined eq "verb"){

        # print "attribute: $attr\n";
         if($tag eq "span"){
        $value = $attr->{'name'};}

#just to get the initial part of the verb: es. for verb escape,  be
careful to respect the parsing order


#<!--BOF_SUBHEAD-->
#<b>es·caped</b>, <b>es·cap·ing</b>, <b>es·capes</b>         IT IS A
COMMENT, so analyze the comment
#<br />
#<!--EOF_SUBHEAD-->


if ($origtext =~ "class=\"me\""){
$prefix_verb{count}+= 1;
#print "found\n";
  }


          #secondary holds values for
if ($origtext =~ "secondary" ){
$conjugation{count}+= 1;
      #print "text: $origtext\n";
  }

if ($origtext =~ "pg" ){
$part_of_speech{count}+= 1;
      #print "text: $origtext\n";
  }

if ($origtext =~ "sectionLabel" ){
$section_label{count}+= 1;
 $do_not_print= 0;
  }

if ($origtext =~ "class=\"dn\""){
$synonims_label{count}+= 1;
  }

if ($origtext =~ "ital-inline" ){
$exemplary_sentence{count}+= 1;
  }

if ($origtext =~ "rom-inline" ){
$do_not_print= 1;
$ad_jec_tive{count}=0;
  }

# <!--BOF_DEF-->
#<ol type="1">     stand in capturing

}
}
```

```
#we need to catch the original commencement of the verb
These functions below catch the text included inside the tag.
sub text_noun {
      my ($self,$text) = @_;
}
sub text_adjective {
      my ($self,$text) = @_;
}
sub text_adverb{
      my ($self,$text) = @_;
}

sub text_encyclopedia{
my ($self,$text) = @_;

if ($text =~ "Venues" ) {
      print "found fvrofierhouregerre";
}


}



sub text_thesaurus{
      my ($self,$text) = @_;
#print"entered thesaurus reference with $word\n";
$_ = $text;
my $count = 0;
#number of entries on the page
if ( /Main Entry/) {
 #we encountered this part, we now that next second line will hold our
value
$value = 2;

}
if ( !/nbsp/ &&  !/Main Entry/&& $value == 2){
#print "Main Entry: $text\n";
$txt_sentence_compiled -> insert('end',"Main Entry: $text\n");
$thesaurus_array[$position_in_sentence]->{_mainEntry}=
$thesaurus_array[$position_in_sentence]->{_mainEntry} . "," . $text;
$value = 0;
}
if ( /Part of Speech/ &&  $value == 0 ) {
 #we encountered this part, we now that next second line will hold our
value
 $value = 3;
}
if ( !/nbsp/ &&  !/Part of Speech/&& $value == 3){
#print "Part of Speech: $text\n";
$txt_sentence_compiled -> insert('end',"Part of Speech: $text\n");
$thesaurus_array[$position_in_sentence]->{_partofSpeech}=
$thesaurus_array[$position_in_sentence]->{_partofSpeech} . "," . $text;
$value = 0;
}
```

```
if ( /Definition/  &&  $value == 0) {
 #we encountered this part, we now that next second line will hold our
value
 $value = 4;
}
if ( !/nbsp/ &&  !/Definition/&& $value == 4){
#print "Definition: $text\n";
$txt_sentence_compiled -> insert('end',"Definition: $text\n");
$thesaurus_array[$position_in_sentence]->{_context_word}=
$thesaurus_array[$position_in_sentence]->{_context_word} . "," . $text;
$value = 0;
}

if ( /Synonims/  &&  $value == 0) {
 #we encountered this part, we now that next second line will hold our
value
 $value = 5;
}
if ( !/nbsp/ &&  !/Synonims/&& $value == 5){
#print "Synonims: $text\n";
$txt_sentence_compiled -> insert('end',"Synonims: $text\n");
$thesaurus_array[$position_in_sentence]->{_synonims}=
$thesaurus_array[$position_in_sentence]->{_synonims} . "," . $text;
$value = 0;
}
if ( /Antonyms/  &&  $value == 0) {
 #we encountered this part, we now that next second line will hold our
value
 $value = 6;
}
if ( !/nbsp/ &&  !/Antonyms/&& $value == 6){
#print "Antonyms: $text\n";
$txt_sentence_compiled -> insert('end',"Antonims: $text\n");
$thesaurus_array[$position_in_sentence]->{_antonyms}=
$thesaurus_array[$position_in_sentence]->{_antonyms} . "," . $text;
$value = 0;
}

#prolog();


}



sub text {
      my ($self,$text) = @_;

if ( $currently_parsing eq "thes" ){
text_thesaurus($self,$text);
}

if ( $currently_parsing eq "encyclopedia" ){
text_encyclopedia($self,$text);
}
```

```
if($type_determined eq "noun"){
text_noun($self,$text);
}
if($type_determined eq "adjective"){
text_adjective($self,$text);
}
if($type_determined eq "adverb"){
text_adverb($self,$text);
}


if($type_determined eq "verb" ){

if ($bof_subhead{count} > 0) {

if($text =~ /[\w]+/ && $text =~ $prefix ){
print "conjugation2: $text\n";
$thesaurus_array[$position_in_sentence]->{_conjugation2}=
$thesaurus_array[$position_in_sentence]->{_conjugation2} . "," . $text;
}
#$txt_dictionary -> insert('end',"conjugation2: $text\n");
$bof_subhead{count} += 1;

if ($bof_subhead{count} > 6){
      $bof_subhead{count}=0;

}

}

$myLength = length($text);
$text_new = "";
if($prefix_verb{count}> 0 && $prefix_verb{count}< 2 ){
   for ($i = 0; $i <$myLength ; $i++) {

   if(substr($text, $i, 1) !~ /[a-zA-Z]/) {
     goto fin; }

 if(substr($text, $i, 1) =~ /[a-zA-Z]/) {
$text_new=$text_new . substr($text, $i, 1); }
}
fin:
      $prefix_verb{count}+=1;
      print "Prefix: $text_new\n";
$thesaurus_array[$position_in_sentence]->{_prefix}=
$thesaurus_array[$position_in_sentence]->{_prefix} . "," . $text;

Here we instanciate the prefix of the word, provided it is a verb.
$txt_dictionary -> insert('end',"Prefix: $text_new\n");

      $prefix = $text;
}

if( $section_label{count} > 0 && $synonims_label{count} > 0 ){
if($text =~ /[a-zA-Z]/){
@words = split(",", $text);
```

```
for $word (@words) {
print "Synonims: $word \n";
$thesaurus_array[$position_in_sentence]->{_synonims_verb}=
$thesaurus_array[$position_in_sentence]->{_synonims_verb} . "," .
$text;
$txt_dictionary -> insert('end',"Synonims: $word \n");
}
$synonims_label{count} = 0;}

#do not forget to reset section_label
}

if($conjugation{count}>0 && $section_label{count}== 0 ){
#es-+cap-+ing-+ly
$myLength = length($text);
#print "length $myLength\n";   #length is one longer than the actual
length
#index and length from index
$text_new = "";
for ($i = 0; $i <$myLength ; $i++) {
if(substr($text, $i, 1) =~ /[a-zA-Z]/) {
$text_new=$text_new . substr($text, $i, 1); }
}
print "conjugation forms : $text_new\n";
$thesaurus_array[$position_in_sentence]->{_conjugation_forms}=
$thesaurus_array[$position_in_sentence]->{_conjugation_forms} . "," .
$text;
$txt_dictionary -> insert('end',"conjugation forms : $text_new\n");
unshift(@conjugation_table,$text_new);
$conjugation{count}-=1;
}
#section label for related forms
if($conjugation{count}>0 && $section_label{count}> 0  &&
      $part_of_speech{count}== 0) {
#do not forget to process
$myLength = length($text);
$text_new = "";
for ($i = 0; $i <$myLength ; $i++) {
if(substr($text, $i, 1) =~ /[a-zA-Z]/) {
$text_new=$text_new . substr($text, $i, 1); }
}
$key=$text_new;
}
if($conjugation{count}>0 && $section_label{count}> 0  &&
      $part_of_speech{count}> 0) {
#do not forget to process
$related_forms{$key}=$text;
$thesaurus_array[$position_in_sentence]-
>{_related_forms}=$thesaurus_array[$position_in_sentence]-
>{_related_forms} . "," . "$key: $text";
print"related form: $key $text\n";


$txt_dictionary -> insert('end',"related form: $key $text\n");
$conjugation{count}=0;
$part_of_speech{count}= 0;
}
```

```
if($part_of_speech{count}>0){
    if ($text =~ /verb \(used without object\)/){
     $verb_without_object{count}+=1;
     $verb_with_object{count}=0;
     $no_un{count}=0;
     $ad_jec_tive{count}=0;
      if($exemplary_sentence{count} >0){
  $exemplary_sentence{count}=0;}
      $part_of_speech{count}=0;
    }
    if ($text =~ /verb \(used with object\)/){
     $verb_with_object{count}+=1;
     $verb_without_object{count}=0;
     $no_un{count}=0;
     $ad_jec_tive{count}=0;
      if($exemplary_sentence{count} >0){
  $exemplary_sentence{count}=0;}
      $part_of_speech{count}=0;
    }
    if ($text =~ /adjective/){
     $ad_jec_tive{count}+=1;
     $verb_with_object{count}=0;
     $verb_without_object{count}=0;
     $no_un{count}=0;
      if($exemplary_sentence{count} >0){
  $exemplary_sentence{count}=0;}
           $part_of_speech{count}=0;
    }
    if ($text =~ /noun/){
     $no_un{count}+=1;
          $verb_with_object{count}=0;
     $verb_without_object{count}=0;
     $ad_jec_tive{count}=0;
      if($exemplary_sentence{count} >0){
  $exemplary_sentence{count}=0;}
           $part_of_speech{count}=0;
    }
#-no_un
#–ad_jec_tive

}


if ($exemplary_sentence{count}>0 ){
 if($verb_without_object{count}>0){
#$verb_without_object{count}-=1;
           #print "verb without object\n";
if(($text =~ $word || $text =~ $prefix) && $text !~ /;/){
print "new sentence for verb without object: $text\n";
$thesaurus_array[$position_in_sentence]-
>{_sentence_verb_without_object}=
```

```
$thesaurus_array[$position_in_sentence]-
>{_sentence_verb_without_object} . "," . $text;

$txt_dictionary -> insert('end',"new sentence for verb without object:
$text\n");
}
elsif ($text =~ $word || $text =~ $prefix){
@verb_sentences = split(";", $text);
for $verb_sentence (@verb_sentences) {
print "new sentence for verb without object:: $verb_sentence \n";
$thesaurus_array[$position_in_sentence]-
>{_sentence_verb_without_object}=
$thesaurus_array[$position_in_sentence]-
>{_sentence_verb_without_object} . "," . $text;
$txt_dictionary -> insert('end',"new sentence for verb without object::
$verb_sentence \n");
}
}

}

 if($verb_with_object{count}>0){
#$verb_with_object{count}-=1;
            #print "verb with object\n";
            if(($text =~ $word || $text =~ $prefix )&& $text !~ /;/){
print "new sentence for verb with object: $text\n";
$thesaurus_array[$position_in_sentence]->{_sentence_verb_with_object}=
$thesaurus_array[$position_in_sentence]->{_sentence_verb_with_object} .
"," . $text;
$txt_dictionary -> insert('end',"new sentence for verb with object:
$text\n");
}
elsif ($text =~ $word || $text =~ $prefix){
@verb_sentences = split(";", $text);
for $verb_sentence (@verb_sentences) {
print "new sentence for verb with object: $verb_sentence \n";
$txt_dictionary -> insert('end',"new sentence for verb with object:
$verb_sentence \n");
}
}

}

       if($no_un{count}>0){
#process these, split...
            $no_un{count}-=1;
            #print "no_un\n";
if( ($text =~ $word || $text =~ $prefix) && $text !~ /;/){
print "new sentence for noun: $text\n";
$thesaurus_array[$position_in_sentence]->{_sentence_for_noun}=
$thesaurus_array[$position_in_sentence]->{_sentence_for_noun} . "," .
$text;
$txt_dictionary -> insert('end',"new sentence for noun: $text\n");
}
elsif ($text =~ $word || $text =~ $prefix){
@no_un_sentences = split(";", $text);
for $no_un_sentence (@no_un_sentences) {
```

```
print "new sentence for noun:: $no_un_sentence \n";
$thesaurus_array[$position_in_sentence]->{_sentence_for_noun}=
$thesaurus_array[$position_in_sentence]->{_sentence_for_noun} . "," .
$text;

$txt_dictionary -> insert('end',"new sentence for noun::
$no_un_sentence \n");
}

}

}



       if($ad_jec_tive{count}>0){
            #$ad_jec_tive{count}-=1;
            #print "ad_jec_tive\n";
if($text =~ $word || $text =~ $prefix ){
print "new sentence for ad_jec_tive: $text\n";
$thesaurus_array[$position_in_sentence]->{_sentence_for_adjective}=
$thesaurus_array[$position_in_sentence]->{_sentence_for_adjective} .
"," . $text;
$txt_dictionary -> insert('end',"new sentence for ad_jec_tive:
$text\n");}
}


}


}
}

sub comment_noun {
my ($self, $comment,$text) = @_;
}
sub comment_adjective {
my ($self, $comment,$text) = @_;
}
sub comment_adverb {
my ($self, $comment,$text) = @_;
}
sub comment_thesaurus {
my ($self, $comment,$text) = @_;
}

sub comment_encyclopedia {
my ($self, $comment,$text) = @_;
}


sub comment {
my ($self, $comment,$text) = @_;
if ( $currently_parsing eq "thes" ){
comment_thesaurus($self, $comment,$text);
}
```

```
if($type_determined eq "noun"){
comment_noun();
}
if($type_determined eq "adjective"){
comment_adjective();
}
if($type_determined eq "adverb"){
comment_adverb();
}


if($type_determined eq "verb" || ($type_determined ne "adjective" &&
$type_determined ne "noun" )  ){

if ($comment  =~ BOF_SUBHEAD){
$bof_subhead{count}+= 1;
}


}
}

sub end_noun{
my ($self, $tag, $origtext) = @_;
}
sub end_adjective{
my ($self, $tag, $origtext) = @_;
}
sub end_adverb{
my ($self, $tag, $origtext) = @_;
}
sub end_thesaurus{
my ($self, $tag, $origtext) = @_;
}

sub end_encyclopedia{
my ($self, $tag, $origtext) = @_;
}



sub end {
my ($self, $tag, $origtext) = @_;
if ( $currently_parsing eq "thes" ){
end_thesaurus($self, $tag, $origtext);
}

if($type_determined eq "noun"){
end_noun($self, $tag, $origtext);
}
if($type_determined eq "adjective"){
end_adjective($self, $tag, $origtext);
}
if($type_determined eq "adverb"){
end_adverb($self, $tag, $origtext);
}
```

```
if($type_determined eq "verb" || ($type_determined ne "adjective" &&
$type_determined ne "noun" )  ){

# reset appropriate flag if we see </H1> or </TITLE>
if ($origtext =~ "secondary"){ $print1 = "false"; }



}
}


package main;

# Test the parser


my $verb_tense = new AI::Proplog;

my $count1 = 0;
my $word_found = "false";
my @sentence_clauses;
my $word_global;

@words = split(" ", $sentence);
 for $word (@words) {
$word_global = $word;

//Sentence composition

#now we browse through the list     of all parts of speech
my @keys = keys %sentence_structure


foreach $key (sort(keys %sentence_structure)){

if ("pronoun" eq $key){
for $pro (@pronoun) {
if ("subjective_personal_pronoun" eq $pro){ for $pro_sub
(@subjective_personal_pronoun){ if( ($word_global =~ $pro_sub && $word
=~ /'/ ) || $word_global eq $pro_sub     ){ print "subject or object
$pro_sub found \n";
$thesaurus_array[$position_in_sentence]->{_type}=
$thesaurus_array[$position_in_sentence]->{_type} . "," . $text;

$word_found = "true"; $type_determined =
"subjective_personal_pronoun";goto begin_rules;}  }    }
if ("objective_personal_pronoun" eq $pro){ for $pro_sub
(@objective_personal_pronoun){  if(  $word_global eq $pro_sub     ){
print "subject or object $pro_sub found\n";   $word_found =
"true";$type_determined = "objective_personal_pronoun";goto
begin_rules;}  }    }
if ("possessive_personal_pronoun" eq $pro){ for $pro_sub
(@possessive_personal_pronoun){  if( $word_global eq $pro_sub     ){
print "subject or object $pro_sub\n";   $word_found =
```

```
"true";$type_determined = "possessive_personal_pronoun";goto
begin_rules;}  }   }
if ("demonstrative_personal_pronouns" eq $pro){ for $pro_sub
(@demonstrative_personal_pronouns){  if(  $word_global eq $pro_sub
){ print "subject or object: $pro_sub\n";   $word_found =
"true";$type_determined = "demonstrative_personal_pronoun";goto
begin_rules;}  }   }
if ("interrogative_pronouns" eq $pro){ for $pro_sub
(@interrogative_pronouns){  if( $word_global eq $pro_sub     ){ print
"subject or object $pro_sub\n";   $word_found = "true";$type_determined
= "interrogative_pronoun";goto begin_rules;}  }   }
if ("relative_pronouns" eq $pro){ for $pro_sub (@relative_pronouns){
if( $word_global eq $pro_sub     ){ print "subject or object
$pro_sub\n";   $word_found = "true";$type_determined =
"relative_pronoun";goto begin_rules;}  }   }
if ("indefinite_pronouns" eq $pro){ for $pro_sub
(@indefinite_pronouns){  if(  $word_global eq $pro_sub     ){ print
"subject or object $pro_sub\n";   $word_found = "true";$type_determined
= "indefinite_pronoun";goto begin_rules;}  }   }
if ("reflexive_pronouns" eq $pro){ for $pro_sub (@reflexive_pronouns){
if( $word_global eq $pro_sub     ){ print "subject or object
$pro_sub\n";   $word_found = "true";$type_determined =
"reflexive_pronoun";goto begin_rules;}  }   }
if ("intensive_pronouns" eq $pro){ for $pro_sub (@intensive_pronouns){
if(  $word_global eq $pro_sub     ){ print "subject or object
$pro_sub\n";   $word_found = "true";$type_determined =
"intensive_pronoun";goto begin_rules;}  }   }
if ("reciprocal_pronouns" eq $pro){ for $pro_sub
(@reciprocal_pronouns){  if(  $word_global eq $pro_sub     ){ print
"subject or object $pro_sub\n";   $word_found = "true";$type_determined
= "reciprocal_pronoun";goto begin_rules;}  }   }

}

       }
if ("adjective" eq $key){
for $adj (@adjective) {
if ("past_participled_adjective" eq $adj){ }  #do not implement, we do
not know if it is a verb, implement the rule
if ("possessive_adjective" eq $adj){for $pro_sub
(@possessive_adjective){  if( $word_global eq $pro_sub     ){ print
"adjective $pro_sub\n";   $word_found = "true";$type_determined =
"possessive_adjective";goto begin_rules;}  }   }
if ("demonstrative_adjective" eq $adj){for $pro_sub
(@demonstrative_adjective){  if( $word_global eq $pro_sub     ){ print
"adjective $pro_sub\n";   $word_found = "true";$type_determined =
"demonstrative_adjective";goto begin_rules;}  }   }
if ("interrogative_adjective" eq $adj){for $pro_sub
(@interrogative_adjective){  if( $word_global eq $pro_sub     ){ print
"adjective $pro_sub\n";   $word_found = "true";$type_determined =
"interrogative_adjective";goto begin_rules;}  }   }
if ("indefinite_adjective" eq $adj){for $pro_sub
(@indefinite_adjective){  if( $word_global eq $pro_sub     ){ print
"adjective $pro_sub\n";   $word_found = "true";$type_determined =
"indefinite_adjective";goto begin_rules;}  }   }
if ("superlative_adjective" eq $word_global){} #implement the rule
if ("comparative_adjective" eq $word_global){}
```

```
if ("comparative_irregular_adjective" eq $word_global){}
if ("superlative_irregular_adjective" eq $word_global){}
if ("adjective_size" eq $adj){for $pro_sub (@adjective_size){  if(
$word_global eq $pro_sub     ){ print "adjective $pro_sub\n";
$word_found = "true";$type_determined = "adjective_size";goto
begin_rules;}  }   }
if ("adjective_shape" eq $adj){for $pro_sub (@adjective_shape){  if(
$word_global eq $pro_sub     ){ print "adjective $pro_sub\n";
$word_found = "true";$type_determined = "adjective_shape";goto
begin_rules;}  }   }
if ("adjective_color" eq $adj){for $pro_sub (@adjective_color){  if(
$word_global eq $pro_sub     ){ print "adjective $pro_sub\n";
$word_found = "true";$type_determined = "adjective_color";goto
begin_rules;}  }   }

}
      }
if ("article" eq $key){
for $art (@article) {
if ($word_global eq $art){print "article: $art\n"; $type_determined =
"article";$word_found = "true";goto begin_rules; }

}

       }
if ("verb" eq $key){
#it might end with ing, so strip it if matches

for $vrb (@verb) {
if ("irregular_verb_present" eq $vrb){for $pro_sub
(@irregular_verb_present){  if( $word_global eq $pro_sub     ){ print
"verb $pro_sub\n";   $word_found = "true";$type_determined =
"verb";goto begin_rules;}  }   }
if ("irregular_verb" eq $vrb){
##############################
#beware you got exceptions to that rule
##############################
my @keys = keys %irregular_verb;
my $key_verb;
foreach $key_verb ((keys %irregular_verb)){
my @verbs_ = split(",", $irregular_verb{$key_verb});

my $regular_form = $key_verb;
my $preterit = $verbs_[0];
my $plus_perfect = $verbs_[1];

 for $verb_value (@verbs) {
my $dummy_verb = $word_global;
my $word_global =~ s/ing//;
if ($verb_value eq $dummy_verb  ){ print "verb irregular
$verb_value\n";   $word_found = "true";$type_determined = "verb";goto
begin_rules;}
elsif ($verb_value =~ $word_global) {
print "verb irregular $verb_value\n";   $word_found =
"true";$type_determined = "verb";goto begin_rules;
}
}
```

```
}
}
}
}

if ("preposition" eq $key){
for $prep (@preposition) {
if ($prep eq $word_global){print "preposition: $word_global\n";
$word_found = "true";$type_determined = "preposition";goto
begin_rules;}

}

      }
if ("conjunction" eq $key){
for $conj (@conjunction) {
if ("coordinating_conjunction" eq $pro){for $pro_sub
(@coordinating_conjunction){  if( $word_global eq $pro_sub     ){ print
"conjunction $word_global\n";   $word_found = "true";$type_determined =
"coordinating_conjunction";goto begin_rules;}  }   }

if ("subordinating_conjunction" eq $pro){for $pro_sub
(@subordinating_conjunction){  if( $word_global eq $pro_sub     ){
print "conjunction $word_global\n";   $word_found =
"true";$type_determined = "subordinating_conjunction";goto
begin_rules;}  }   }

if ("correlative_conjunction" eq $pro){for $pro_sub
(@correlative_conjunction){  if( $word_global eq $pro_sub     ){ print
"conjunction $word_global\n";   $word_found = "true";$type_determined =
"correlative_conjunction";goto begin_rules;}  }   }


}

      }
if ("number" eq $key){

my @keys = keys %number;
foreach $key (sort(values %number)){
if ($key eq $word_global || $number{$key} eq $word_global  ){}

}


      }
}
my $parser = MyParser->new;
#browsing through the lists above, we determine the partofspeech:
type_determined
######################################################################
#first pass through the thesaurus to get basic information

$thesaurus_array[$position_in_sentence] = Thesaurus->new();
$position_in_sentence+=1;
$currently_parsing = "thes";
$parser->parse( LWP::Simple::get($url_thesaurus . $word_global));
```

```
######################################################################
#second pass through the dictionary
$currently_parsing = "dict";
$parser->parse( LWP::Simple::get($url_dictionary . $word_global));
######################################################################
#third pass through the encyclopedia: context
$currently_parsing = "encyclopedia";
$parser->parse( LWP::Simple::get($url_encyclopedia . $word_global));

begin_rules:


print "begin rules with $word_global\n";
$txt_sentence_compiled -> insert('end',"begin rules with
$word_global\n");


#    my @array_string =( "object(adj,def)" ,
"object(noun,def)","object(number,
#def)","object(pronoun,def)","subject(pronoun,def)"
,"verb(tense,def)","verb(sub,def)","verb(obj,def)", "verb(adverb,def)",
#"verb(qualifier, def)","verb(concurrent, def)","verb(verb, def)"
#,"verb(condition,def)","noun(adjective,def)","noun(noun,def)","noun(pr
onoun,def)"  );


for ($loop1 = $position_in_sentence; $loop1 >= 0; $loop1--) {
for ($loop2 = $position_in_sentence; $loop2 >= 0; $loop2--) {

#go into more details of ther type of pronoun...go into details of the
adjacency of the matches

if ($thesaurus_array[$loop1]->{_partofSpeech} eq "article" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }
if ($thesaurus_array[$loop1]->{_partofSpeech} eq "noun" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }
if ($thesaurus_array[$loop1]->{_partofSpeech} eq "pronoun" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }
if ($thesaurus_array[$loop1]->{_partofSpeech} eq "verb" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }
if ($thesaurus_array[$loop1]->{_partofSpeech} eq "article" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }
if ($thesaurus_array[$loop1]->{_partofSpeech} eq "article" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }
if ($thesaurus_array[$loop1]->{_partofSpeech} eq "article" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }
if ($thesaurus_array[$loop1]->{_partofSpeech} eq "article" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }
if ($thesaurus_array[$loop1]->{_partofSpeech} eq "article" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }
if ($thesaurus_array[$loop1]->{_partofSpeech} eq "article" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }
if ($thesaurus_array[$loop1]->{_partofSpeech} eq "article" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }
if ($thesaurus_array[$loop1]->{_partofSpeech} eq "article" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }
```

```
if ($thesaurus_array[$loop1]->{_partofSpeech} eq "article" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }
if ($thesaurus_array[$loop1]->{_partofSpeech} eq "article" &&
$thesaurus_array[$loop1]->{_partofSpeech} eq "verb") { }




}
}



$count1+=1;
   if ( $word_global =~ /'S/) {
@words1 = split("'", $word_global);
my $dummy1 = pop @words1;

if ( $dummy1 =~ /[sS]/ && $words1[0] !~ /[Ii][Tt]/ ){
  print "tense is present and one object\n";
$dummy = @array_string[5];
$dummy =~ s/def/be/;
$dummy =~ s/tense/present/;
push(@sentence_clauses,$dummy );
  }
if ( $dummy1 =~ /[sS]/  && $words1[0] =~ /[Ii][Tt]/ ){
  print "tense is present and abstract concept or phenomenon\n";
$dummy = @array_string[5];
$dummy =~ s/def/be/;
$dummy =~ s/tense/present/;
push(@sentence_clauses,$dummy );

  }
if ( $dummy1 =~ /[re]/ && $words[count-1] !~ /[Tt]here/){
  print "tense is present and several objects and/or context
definition\n";       #check for previous word: there ... would mean a
context definition
$dummy = @array_string[5];
$dummy =~ s/def/be/;
$dummy =~ s/tense/present/;
push(@sentence_clauses,$dummy );

  }

}
}
my $temp;

print "\n@sentence_clauses\n";
for ($temp = $position_in_sentence; $temp >= 0; $temp--) {
print $thesaurus_array[$temp]->{_partofSpeech};
}

#pronoun
```

```
#types =>"subjective_personal_pronoun objective_personal_pronoun
possessive_personal_pronoun demonstrative_pronouns
interrogative_pronouns relative_pronouns indefinite_pronouns
reflexive_pronouns intensive_pronouns"

#adjective
#rule => "article number possessive_adjective demonstrative_adjective
indefinite_pronoun",
#types => "past_participled_adjective possessive_adjective
demonstrative_adjective interrogative_adjective indefinite_adjective
possessive_adjective superlative_adjective comparative_adjective
comparative_irregular_adjective superlative_irregular_adjective"

#noun
#types => "person_noun animal_noun place_noun thing_noun
abstract_idea_noun subject_noun direct_object_noun indirect_object_noun
suject_complement_noun object_complement_noun appositive_noun
adjective_noun"

#verb
#types => "verb_adverb adjective_adverb adverb_adverb phrase_adverb
dummy1_verb emotion_verb location_verb physical_verb spiritual_verb
identification_verb mental_verb past_verb present_verb future_verb
regular_verb irregular_verb inarow_verb"

#preposition
#types => "nouns_preposition pronouns_preposition phrases_preposition",
#list => "about above across after against along among around at before
behind below beneath beside between beyond but by despite down during
except for from in inside into like near of off on onto out outside
over past since through throughout till to toward under underneath
until up upon with within without"

#conjunction
#types =>  "coordinating_conjunction subordinating_conjunction
correlative_conjunction"

#article

#number

#Be(tense,present)
#Be(sub,it)
#Be(obj,act)
#act(adj,juggling)
#act(noun,doorstep)
#rummage(tense,dummy1)
#rummage(obj, keys)
#keys(noun,house)
#rummage(qualifier, hand)
#hand(number, one)
#rummage(concurrent, balance)
#balance(tense,dummy1)
#balance(obj, bag)
#bag(noun,groceries)
#balance(qualifier, other)
```

```
# and so forth
$verb_tense->a( intro_req => 'intro_cs');
$verb_tense->a( intro_req => qw(introI introII) );
$verb_tense->a( math_req  => qw(calc_req finite_req alg_req) );
$verb_tense->a( calc_req  => qw(basic_calc adv_calc) );
$verb_tense->a( basic_calc => qw(calcI calcII) );
$verb_tense->a( basic_calc => qw(calcA calcB calcC) );
$verb_tense->a( adv_calc   => 'lin_alg');
$verb_tense->a( adv_calc   => 'honors_linalg');
$verb_tense->a( finite_req => qw(fin_structI stat) );
$verb_tense->a( alg_req    => 'fin_structII');
$verb_tense->a( alg_req    => 'abs_alg');
$verb_tense->a( alg_req    => 'abs_alg');

# here we assert a bunch of facts:
# the following things have been taken:
# cs intro, computer org, advanced programming, and theory
$verb_tense->apl( qw(basic_cs math_req advanced_cs engr_rec
natural_science) );
# now do a bottom up search of the fact/rule space to see if the
# basic cs requirements have been met
my $R = $verb_tense->bottom_up('cs_req');
#my $R = $p->top_down('cs_req');
print "The basic cs requirements have";
print $R;
print " been met\n";




      my $name = $ent -> get();
#     $txt -> insert('end',"$name\($gender\) is $age years old and is
");

      my $job = "";
      #See whether he is employed
      if ( $occupied == 1 ) {
            my $job_id = $lst -> curselection(); #Get the no of
selected jobs
            if ( $job_id eq "" ) { #If there is no job
                  $job = "a Non worker.";
            }
            else {
                  $job = $lst -> get($job_id) ;#Get the name of the job
#                 $txt -> insert('end',"a $job.");
            }
      }
      else {
#           $txt -> insert('end',"unemployed.");
      }
}

sub menuClicked {
```

```
      my ($opt) = @_;
      $mw->messageBox(-message=>"You have clicked $opt.
This function is not implanted yet.");
}


#################################################################
#####################################################################
```

Here we show the code for the web bot.

I have written different implementations. One of them uses threads which are called by a central function serving as the central thread manager.

Such systems gather speeds of at least, 600KB a second.

Here I chose to use the simple web extractor which browses a man made directory, goes into each website belonging to one of the millions of declared categories.

It doesn't use the same concept as Google's URL Anchor, where supposedly the site is better described or summarized in the anchor, although it should and could.

It goes a two sub levels of depth in websites not belonging to the central directory, extracts sentences and calls thesaurus reference script.

We have isolated the two functionalities for now.

The script above reads the words in the sentences extracted and builds an input data stream to be fed to the prolog logic clauses and the sequential learning algorithm.

AllWebHuntNew.pl

Web Hunt, bot.

```
use Thesaurus;
use AI::Proplog;
use AI::NeuralNet::BackProp;
#use DB_File;
my $text_global;
my $url_global;
my $link = "http://allwebhunt.com/dir-wiki.cfm/Top/Business";
my $match = "http://AllWebHunt.com/dir-wiki.cfm/Top/Business";
my $prolog = new AI::Proplog;
#point every element of a sentence to a hash so that position arrays contain digits
```

```
package MyParser;
use base qw(HTML::Parser);
use LWP::Simple ();
use DBI;
#use >> to append

open(MYFILE16, '>>shopping.txt') or print "die";

my $true_dash = 0;
print "opened $link as a global defined variable\n";
my $sentence;
my $sentence_not_completed = "true";
#we must first go through all URL's of the site
#sub text           { $text_elements++  }
#sub start          { $start_tags++     }
#sub end            { $end_tags++       }
#max of 30 words per sentence
my $thesaurus =  eval { new Thesaurus(); }  or die ($@);

my $front_page = 0;
my @words;
my $sth;

#in order to prevent infinite recursion of the web links

#we need to run through the entire website recursively and extract text starting with
```

```
my $match_url;
#print "match_url is $match_url\n";

#a sentence is a sequence of words separated by white spaces


#we are only interested in the URL's belonging to the NYTIMES
#and not containing any of these :, ?, =
my %already_parsed;
my %match_found;
my %match_found1;
my %match_found2;
my $present= "false";
my $key_value;
my $value_value;
my $value_value1;
my %category;
my $category_global;
sub start {
my $word;
#my $url;
my $thesaurus_parsed;
#open and close file to retrieve thesaurus
my $ not_on_file ;
my $stop = 0;;

#dbmopen(%match_found, "my_database_match", 0644)
#  or die "Cannot create my_database: $!";

        #print "entered start function\n";
          my ($self, $tagname, $attr) = @_;
```

```
        #print "SELF = $self\n";
            if ($tagname eq 'a') {
                my $url = $attr->{ href };



            return if $already_parsed{ $url };
            $_ = $url;
            $url_global = $url;
             #print "$_\n";


        if ( /$match/){
#           print "$_\n";

            $_ = $url;

           print MYFILE "\{$url\{\n";

           print "matched dir-wikWe$url and category is $category_global\n";

                # make a new parser to parse
                # the document referenced by $url
            $already_parsed{ $url }++;


            open (MYFILE4, '>words_parsed.txt')or print "die";
            print MYFILE4 "$url";
            close (MYFILE4);
            my $p = MyParser->new;
       $category_global = $url;
```

```
            $p->parse( LWP::Simple::get($url) );


     }


  elsif( $already_parsed{ $url}<1 && $url !~/dir-wiki/ && $url !~ /reference\.com/ && $url
=~/http/ && $url !~/[aA]llwebhunt/ && $url !~ /\.cfm/  && $url!~ /[\=\?#]/  ) {


#$sentence =~ /the/

#the must appear in sentence


#we count character frequencies in a line: more specifically here, we count /, since we admit only
a total

#of four / in a non wikWesite

#See tutorial http://perldoc.perl.org/perlretut.html

    $url =~ s/(.)/$chars{$1}++;$1/eg;  # final $1 replaces char with itself

 print "/\ appears $chars{'/'} times \n";

if ( $chars{'/'} < 5 ){

$match_found{$url}=1;

  $true_dash = 1; }


  $chars{'/'}=0;


if ( $true_dash == 1 ){                    #2

    #bug fix

$true_dash = 0;

          print MYFILE "\}$url\}  \n";

$already_parsed{ $url }++;


#print " and parsed is ";

#print  $already_parsed{ $url};

#print "\n";

$present = "false";
```

```
my @keys = keys %match_found;

foreach $key (sort(keys %match_found)){
$_ = $key;
$key_value = $key;
#it will always match itself

if( $url=~ $key && $present== "false"){
            $present = "true";
#           break;
print"circuit breaker $url $key\n";
goto second;
#break;
}
}

 if( $present == "false") {              #3
  print "url category 1: $url\n";
   my $p11 = MyParser->new;
  $match_url = $url;
   $p11->parse( LWP::Simple::get($url) );

}
#else, we know it is present but we ignore its level: 1,2,3
elsif ($present == "true" && $url !=  $key_value) {
second:
print "reached second loop because $url matches $key_value\n";
$present = "false";
my @values = values %match_found1;
foreach $value (sort(values %match_found1)){
$value_value = $value;
```

```
$_ = $value;
#check
if(  $url =~ $value ){
$present = "true";
goto third;
break;
}
}

 if( $present == "false") {
  print "url category 2: $url\n";
   $match_found1{$key_value}=  $match_found1{$key_value } . $url;
   my $p12 = MyParser->new;
  $match_url = $url;
   $p12->parse( LWP::Simple::get($url) );

}

elsif ($present == "true" && $url !=  $value_value) {
third:
print "reached third loop because $value_value matches $url$\n";
$present = "false";
my @values = values %match_found2;
foreach $value (sort(values %match_found2)){
$_ = $value;
$value_value1 =$value;
#check
if($value =~ $value_value){
$present = "true";
break;
}
```

```
}

 if( $present == "false") {
  print "url category 3: $url\n";
   $match_found2{$value_value1 }=  $match_found2{$value_value1 } . $url;
   my $p13 = MyParser->new;
  $match_url = $url;
   $p13->parse( LWP::Simple::get($url) );

}
}
}
}
}
}
}



sub text {

        my ($self, $text) = @_;
        $text_global = $text;
#white sapce is \s
if($text =~ /\./){
$text =~ s/\./\sdot\s/;}
if($text =~ /\!/){
$text =~ s/\!/\sexclamation\s/;}
if($text =~ /,/){
$text =~ s/,/\scomma\s/;}
if($text =~ /\'/){
```

```
$text =~ s/\'/\s/;}

if($text =~ /\;/){

$text =~ s/\;/\ssemicolumn\s/;}


$_ = $text;

if( /[a-zA-Z\s]*[\?\!\.\;]$/ &&$thesaurus_parsed !~  /thesaurus\.reference\.com/&&$url_global !~ /thesaurus\.reference\.com/ && $already_parsed{$url_global}>0 && $url_global !~/dir-wiki/ && $url_global!~ /\.cfm/ && $url_global!~ /\?/) {


@words = split(" ", $_);


$thesaurus_string = "http://thesaurus.reference.com/browse/";

 for $word (@words) {

$thesaurus_string = "http://thesaurus.reference.com/browse/";


if($word =~ /\./){

$word =~ s/\./\s/;}


if($word =~ /\w/ ){

    $sentence_not_completed = "true";

    if($word =~ /dot/ && $word =~ /exclamation/ && $word =~ /semicolumn/ ){

    $sentence_not_completed = "false";}

    print "word $word\n";

    $thesaurus_string = $thesaurus_string . $word;

    $thesaurus_parsed = $thesaurus_string;

my $pid = fork();

if ($pid == 0){

exec"perl thesaurusreference.pl $thesaurus_string $word $category_global $sentence_not_completed";}



}
```

```
}
 $thesaurus_parsed = "";

#whenever we encounter a dot, a new sentence is processed in prolog




}
}









sub prolog{

if ($sentence_not_completed == "false") {

#substitute         "http://allwebhunt.com/dir-wiki.cfm/Top/" with "" in category_global

$prolog->a( category_global => qw($sentence));
$sentence = "";
$prolog->a( category_global => qw($sentence));

}
}


sub neural_network{
```

```
# Create a new network with 2 layers, 30 inputs, and 5 outputs.
#prolog creates rules for every content category
#the AWENN will be trained for every one of these matching sentences

        my $net = new AI::NeuralNet::BackProp(2,30,5);

        # Add a small amount of randomness to the network
        $net->random(0.001);

        # Demonstrate a simple learn() call
        my @inputs = ( 0,0,1,1,1,0,1,1,1,0,1,1,1,0,1,1,1 ,0,1,1,1 ,0,1,1,1   );
        my @ouputs = ( 1,0,1,0,1 );

        print $net->learn(\@inputs, \@outputs),"\n";




}



package main;

# Test the parser



my $parser = MyParser->new;
```

```
$parser->parse( LWP::Simple::get($link) );



close (MYFILE);
```