SNePS and WordNet

Using WordNet as a Source of Background Knowledge in Contextual Vocabulary Acquistion

Dmitriy Dligach
Department of Computer Science and Engineering; SNePS Research Group
State University of New York at Buffalo, Buffalo NY 14260

Abstract

Contextual Vocabulary Acquisition (CVA) is an interdisciplinary project which focuses on developing a computational theory of vocabulary acquisition. Currently, the background knowledge that is necessary for performing CVA is collected from human readers and manually encoded into SNePS (the Knowledge Representation system used in the CVA project). Therefore, using an external knowledge base such as CYC or WordNet as a source of background knowledge presents an interesting research problem.

This paper discusses an approach to using WordNet as a source of background knowledge for CVA. First, WordNet noun and verb relations are analyzed in order to determine how they can be represented in SNePS. Next, an implemented WordNet interface program is described that is capable of converting WordNet ontologies to SNePSUL (SNePS user language). Finally, an example illustrating the usage of this program in CVA is presented. A discussion of possible future research directions concludes the paper.

1. Introduction

Brief SNePS and CVA Overview

This project is a part of a broader research effort conducted by the members of the SNePS and CVA research groups. The purpose of the following two sections is to provide a quick overview of SNePS and CVA and cite the literature that would be helpful to the reader who is unfamiliar with these projects.

SNePS

SNePS is a propositional, semantic-network knowledge representation and reasoning (KRR) system. From the outset, SNePS was conceived as a semantic network suitable for representing natural language discourses, and the current version of the SNePS user language (SNePSUL) offers a high level of expressibility comparable to that of natural languages. Like any network, SNePS consists of a system of nodes interconnected by arcs. What distinguishes SNePS from other semantic networks is that each proposition is represented by a node (and not by an arc) and that nodes represent intensional concepts. Thus, every SNePS node represents a single concept and every arc represents the relation between the concepts that this arc connects. The content of the SNePS network is often viewed as the mind of an artificial agent, who is affectionately referred to by the member of the SNePS research group as Cassie (see Shapiro & Rapaport 1987).

CVA

Contextual Vocabulary Acquistion (CVA) is an interdisciplinary project whose main objective is developing a computational theory of vocabulary acquisition from context. The CVA project currently develops in two major directions: AI and Education. The AI component focuses on creating an artificial agent capable of determining the meaning of an unknown word from its context (Ehrlich 1995; Rapaport & Ehrlich 2000). The goal of the educational component is transforming CVA algorithms into an educational curriculum (Rapaport & Kibby 2002)

The CVA approach to learning from context is SNePS based. It follows this process: First, Cassie reads a passage containing an unknown word: it is translated into the language of SNePS logic and programmed into SNePS. Next, background knowledge that a human reader uses to learn the meaning of the unknown word is identified. This is usually accomplished by means of collecting verbal protocols from human readers who are unfamiliar with the meaning of the word in question. Then, this background knowledge is also translated into a SNePS representation and programmed into SNePS. After that, the SNePS reasoning engine draws the appropriate inferences from the

information available in the network. At this point, Cassie's mind contains, in the form of a semantic network, all the information that is necessary to determine a meaning for the unknown word. Finally, a (part-of-speech-specific) CVA algorithm is executed. The algorithm traverses the semantic network and collects all the information that is relevant to the definition of the unknown word. This information is organized into such categories as class memberships, actions, properties, synonyms, etc., and output to the user.

Motivation and Issues

The motivation behind this project is to automate (at least partially) the process of teaching Cassie the background knowledge that she needs in order to perform CVA. Currently, this background knowledge is acquired by means of collecting verbal protocols from humans and encoding it manually into SNePS. However, several large external (non-SNePS based) repositories of knowledge exist, such as CYC and WordNet, and these repositories can potentially be used as sources of background knowledge for CVA. There are two important issues in using an external knowledge base (KB) as a source of background knowledge:

1. Identifying relevant knowledge: Cassie must be able to identify the kinds of knowledge that are relevant to computing a meaning for the given unknown word. In other words, Cassie needs to be able to query an external KB to extract only the knowledge that she needs for CVA. Of course, this is not necessary if the entire content of an external KB is imported into Cassie's mind. However, given the fact that the sizes of both the CYC and WordNet KBs are quite large, an option of importing the entire KB was ruled out to avoid a considerable increase in running time of the CVA algorithms.

2. Representing knowledge: Cassie must learn the language in which the knowledge is expressed in the external KB. Alternatively, the external knowledge can be converted to SNePSUL, i.e. to the language that Cassie can understand. The second option can be implemented by means of a program external to SNePS, which would retrieve the knowledge from an external KB and convert it to SNePSUL. Such a program can, if necessary, be incorporated into SNePS.

CYC vs. WordNet

For the purpose of this project, two alternative external KBs were considered: CYC, a repository of commonsense knowledge developed by CYCorp, and WordNet, an electronic lexical database developed by the Cognitive Science Laboratory at Princeton University. Below, the pros and cons of using CYC vs. WordNet as a source of background knowledge for CVA will be outlined.

There are several reasons why WordNet is a better choice as an external source of background knowledge.

1. Identifying relevant knowledge in WordNet is more straightforward than in CYC: the content of the CYC KB is topically grouped into microtheories (or contexts), that is, into sets of assertions which share the same assumptions; searching for the relevant background knowledge in CYC requires identifying the microtheories that are relevant to the given context. Unfortunately, finding the right microtheory is more of an art than science and thus is not easily automated (see more on this in section 8). At the same time, given a word form (a single word or a collocation) and a semantic relation such as hypernymy or meronymy, the WordNet KB can be searched to retrieve the word forms that stand in this semantic relation to the word form in question. For example, given the word form *unicorn* and the semantic relation of hypernymy, we can find out from WordNet that *unicorn* belongs to the class of *imaginary beings.* Therefore, the lexical content of the passage containing an unknown word, that is, the words and collocations the passage is composed of, provide sufficient parameters that the WordNet KB can be searched by.

2. Representing WordNet ontologies in SNePS is more straightforward than representing CYC ontologies: the facts that only a finite number of semantic relations are defined in WordNet and that these relations can be mapped to the standard CVA case-frames make it possible to convert WordNet ontologies to SNePSUL. The CYC ontologies are different in this respect: CYC allows for an unrestricted number of predicates, and thus a CYC-to-SNePS dictionary of a variable and unpredictably large size would be necessary for the task of converting CYC ontologies to the representations that are standard to CVA. Additionally, the task of mapping CYC predicates to the standard CVA case-frames is complicated by the fact that although some of the CYC predicates correspond to the existing CVA case-frames (e.g., *isa* corresponds to the *member/class* case-frame; *genls* corresponds to the *subclass/superclass* case-frame), most CYC predicates do not have such mappings.

It must be noted that there is a downside to using WordNet ontologies as a source of background knowledge: although retrieving data and converting it to its SNePS representation is more straightforward in the case of WordNet than in the case of CYC, the knowledge obtained from WordNet is limited to the types of semantic relations that are built in WordNet. Because WordNet was never intended to be a source of commonsense knowledge, many facts, the knowledge of which is necessary for determining a meaning for a word from context, are more likely to be found in CYC than in WordNet. On the other hand, OpenCyc – the version of CYC that is currently freely distributed by CYCorp – is far from being comprehensive. For example, the query (#$isa ?X #$BiologicalSpecies), which asks CYC to produce all entities that are biological species, only comes up with a dog and a lion, whereas WordNet in response to the equivalent query, produces a large list of animals.

Goals

Based on the analysis in the previous section, the WordNet KB was adopted as the source of background knowledge for CVA. The goals of this project are, therefore,

1. To develop an approach to representing WordNet ontologies in SNePS
2. Design and implement a WordNet-to-SNePS interface, which would allow the retrieval and conversion of WordNet ontologies to SNePSUL
3. Demonstrate how this interface can be used in CVA

What is to Follow

The remaining portion of this report contains

1. A brief overview of WordNet
2. A discussion of an approach to representing WordNet ontologies in SNePS
3. A description of an automated interface which is capable of retrieving data from WordNet and converting it to SNePSUL
4. A description of the CVA demo that makes use of the WordNet-to-SNePS interface
5. A description of possible future research directions

2. WordNet Overview

Searchable information in a conventional dictionary (most often word forms) is usually organized based on lexical principles – in order to look up a word, one has to perform an alphabetic search. However, it is often useful to be able to look for information conceptually, that is, based on its relationship to another piece of information. For example, we may be interested in finding some of the components which constitute a book. A conventional dictionary would not be very helpful: in order to find information in it, we already have to know the spelling of the word form we are looking for.

In 1985, a group of researchers at Princeton University (Miller 1990) began to develop a new type of dictionary, which later became known as WordNet. The goal of the WordNet designers was to create an electronic dictionary, which would allow its users to perform a semantic search. A typical task such a dictionary can be used for is the following:

Given a word form W and a relation R, find all word forms that stand in the relation R to the word form W.

For example, we can search for all word forms that stand in meronymy, (a part-of) relation to the word *book*. This type of search can be performed in WordNet. The result would tell us that a book has the following parts: pages, binding, etc.

Because our "mental dictionary" can perform a similar task, it was decided to adopt modern psycholinguistic principles as a basis for the design of WordNet. In other words,

WordNet implements an ability that all humans possess – the ability to store and search concepts based on relationships among them.

Because the WordNet KB stores concepts and relationships among them, the basic building block of WordNet KB is a concept, and not a word form, as is the case with the conventional dictionary. This basic building block is called a *synset*. In WordNet, each concept is expressed by means of a synset – a set of one or more word forms – which identifies that concept. The members of a synset are by convention enclosed in curly brackets. For example, the word form *book* is polysemous – it is used to refer to (among others) the following senses:

1. *physical objects consisting of a number of pages bound together*
2. *the sacred writings of Islam revealed by God to the prophet Muhammad during his life at Mecca and Medina*
3. *the sacred writings of the Christian religions*

The following three synsets are used to identify each of the above senses:

1. *{book, volume}*
2. *{Koran, Quran, al-Qur'an, Book}*
3. *{Bible, Christian Bible, Book, Good Book, Holy Scripture, Holy Writ, Scripture, Word of God, Word}*

Because a synset (and not a word form) is the basic building block of WordNet, all relations in WordNet are defined for synsets, and not for individual word forms. Thus, the meronymy search, which was mentioned before, actually referred only to the first meaning of the word *book*. Obviously, if we perform a meronymy search for senses (2) and (3), we should get a different result. Indeed, for sense (2), we get the synset {*sura}* (one of the sections in the Koran), while for sense (3), we get the synsets {*Old Testament}* and *{New Testament}*. Sometimes, the same synset is used to express multiple concepts. In such cases, a short textual gloss is used to differentiate concepts.

3. Representing Synsets

Concept/Word Case Frame

It was decided that a WordNet concept would be represented in SNePS as a base node. Because synset members are verbal expressions of the lexicalized concept they represent, an asserted SNePS node is created for each synset member with two arcs coming out of it: the *concept* arc points to the base node and the *word* arc points to an expression used to articulate the concept. Such a representation parallels the semiotic notion of a sign, which is viewed as a fusion of a concept (signified) and its sound image/verbal expression (signifier).

Thus, the proposition *m:*

(assert concept c word w)

has the following semantics: [[m]] is the proposition that concept (or: synset) [[c]] is expressed by word [[w]].

For example, for the concept

rock (n.) - a lump or mass of hard consolidated mineral matter

which is expressed in WordNet by means of the synset {rock, stone}, the following SNePS representation is used (the approach to forming base node names will be explained in following section):

```
(assert concept #rock-stone-n-8824564
      word "rock")
(assert concept *rock-stone-n-8824564
      word "stone")
```

At the same time, a different concept

rock (v.) - move back and forth or sideways

which is expressed in WordNet by means of the synset {rock, sway, shake}, is represented in SNePS as follows:

```
(assert concept #rock-sway-shake-v-1821183
      word "rock")
(assert concept *rock-sway-shake-v-1821183
      word "sway")
(assert concept *rock-sway-shake-v-1821183
      word "shake")
```

Base-Node Name Formation

There are several considerations that are relevant to the issue of naming base nodes, which represent WordNet concepts in SNePS.

1. The base-node name representing a concept must uniquely identify that concept. Resolving this issue is necessary in order to avoid base-node name “collisions” – a situation when the same base node represents more than one WordNet concept. If a need to import large portions of the WordNet KB into SNePS were to arise in the future, there would be a one-to-one relationship between SNePS base nodes and WordNet concepts.

2. We must be able to generate base node names dynamically, e.g., by means of a computer program. Finding a way to do so will facilitate the creation of a WordNet to SNePS converter program.

The following possible solutions to the problem of forming base node names were considered:

1. To simply *concatenate all members of the synset*. Unfortunately, this solution is flawed because in some cases identical synsets represent more then one concept. For example, WordNet contains the following information for the noun *stone:*

> *1. rock, stone -- (a lump or mass of hard consolidated mineral matter; "he threw a rock at me")*
> *2. rock, stone -- (material consisting of the aggregate of minerals like those making up the Earth's crust; "that mountain is solid rock"; "stone is abundant in New England and there are many quarries")*

[Output A]

The two senses of the noun *stone* (and therefore two different concepts) are represented by the same synset *{rock, stone}.*

2. In WordNet, each sense of a word has a unique identifier called sense key (Beckwith, Miller, & Tengi 1993). Using sense keys for base node names will not always work either, because we might end up with two different base node names representing the same concept. For example, WordNet search for the noun *rock* produces:

> *1. rock, stone -- (a lump or mass of hard consolidated mineral matter; "he threw a rock at me")*
> *2. rock, stone -- (material consisting of the aggregate of minerals like those making up the Earth's crust; "that mountain is solid rock"; "stone is abund ant in New England and there are many quarries")*

[Output B]

Senses (1) and (2) of [Output A] represent the same concepts as the senses (1) and (2) respectively of [Output B]. However, if we used sense keys as base node names, the base node name we would generate for the first sense of the noun *stone* would be different from the base node name for the noun *rock*, despite of the fact that they should represent the same concept expressed by the synset {rock, stone}. The same is true for the second senses of the nouns *stone* and *rock*.

3. It turns out that each synset can be uniquely identified by its offset in the database file for the given part of speech (nouns, verbs, and adjectives are stored in different database files). Therefore, the following strategy for naming base nodes can be used:

Suppose we are interested in some word X, which (as a noun) has three senses defined in WordNet. These senses are expressed by synsets {X, Y}, {X, Y} and {X, Z, W}, whose offsets in the noun database file are respectively 12345, 45678, and 98987987.

The base node names for these senses will be formed by concatenating synset members and attaching to them

a) a part of speech identifier (noun in this case)
b) synset offset

So, the base node names will look like:

Sense 1: X-Y-n-12345
Sense 2: X-Y-n-45678
Sense 3: X-Z-W-n-97987987

Additionally, because synset offsets are available at run time by means of a WordNet KB search, base-node names can be generated "on the fly" as necessary.

For example, using this method, the following assertions can be generated when searching for the first sense of either the noun *stone* or the noun *rock*.

(assert concept #rock-stone-n-8824564
word "rock")
*(assert concept *rock-stone-n-8824564*
word "stone")

This solution is free from the limitations of the first two solutions and therefore was adopted as the approach for generating base-node names.

4. Nouns in WordNet

Noun Relations in WordNet

A typical dictionary definition usually provides a superordinate for the term in question along with the term's distinguishing features (Miller 1990). For example, the word *tree* is defined by the Merriam-Webster dictionary as "a woody perennial plant having a single usually elongate main stem generally with few or no branches on its lower part". A superordinate (woody plant) and distinguishing features (elongate main stem, branches) can be singled out from this definition.

Superordinates (hypernyms) and distinguishing features (currently, only meronyms) are also available in WordNet. WordNet currently stores information about the following noun relations (Miller 1990):

1. Antonymy
2. Hypernymy
3. Hyponymy
4. Synonymy
5. Holonymy
6. Meronymy

Antonymy and synonym are lexical relations, while hypernymy, hyponymy, meronymy, and synonymy are semantic relations.

As was discussed before, the WordNet KB contains concepts and semantic relations between them. Because concepts are represented as synsets, relations are represented as pointers between synsets. It is natural to represent these pointers as SNePS case-frames. In general, the fact that some relation R holds between two synsets S1 and S2 can be represented by a SNePS assertion:

(assert arc1 S1 arc2 S2)

where arc1/arc2 is the case-frame, which the relation R is mapped to.

Each relation along with its SNePS representation will be discussed in the following sections.

Representing Antonymy

The WordNet antonymy relation can be straightforwardly mapped to the SNePS antonym/antonym case-frame used in the CVA project. For example, one of the senses of the noun *day*

day (n.) - the time after sunrise and before sunset while it is light outside

which corresponds to the WordNet synset {day, daytime, daylight}, is represented in SNePS as follows:

```
(assert concept #day-daytime-daylight-n-14305860
      word "day")
(assert concept *day-daytime-daylight-n-14305860
      word "daytime")
(assert concept *day-daytime-daylight-n-14305860
      word "daylight")

(assert concept #night-nighttime-dark-n-14307923
      word "night")
(assert concept *night-nighttime-dark-n-14307923
```

word "nighttime")
*(assert concept *night-nighttime-dark-n-14307923*
word "dark")

*(assert antonym *day-daytime-daylight-n-14305860*
*antonym *night-nighttime-dark-n-14307923)*

The first three assertions represent the synset {day, daytime, daylight}; the next three assertions represent the synset {night, nighttime, dark}; finally, the last assertion represents the fact that the synsets {day, daytime, daylight} and {night, nighttime, dark} are in the antonymy relation with each other.

Representing Hyponymy and Hypernymy

WordNet hypernymy and hyponymy relations are defined as follows: "a concept represented by the synset {x, x1, …} is said to be a hyponym of the concept represented by the synset {y, y1, …} if native speakers of English accept sentences constructed from such frames as 'An x is a (kind of) y'" (Miller 1990). Hypernymy and hyponymy can be mapped to the SNePS subclass/superclass case-frame For example, one of the senses of the noun *tree*

tree (n.) - a tall perennial woody plant having a main trunk and branches forming a distinct elevated crown; includes both gymnosperms and angiosperms.

which corresponds to the WordNet synset {tree}, is represented in SNePS as follows:

(assert concept #tree-n-12352501
word "tree")

(assert concept #woody_plant-ligneous_plant-n-12351578
word "woody_plant")
*(assert concept *woody_plant-ligneous_plant-n-12351578*
word "ligneous_plant")

*(assert subclass *tree-n-12352501*
*superclass *woody_plant-ligneous_plant-n-12351578)*

The above assertions can be interpreted to say that the WordNet concept {tree} is a subordinate (or subclass) of the concept {woody_plant, ligneous_plant}.

Representing Meronymy and Holonymy

WordNet meronymy is defined as follows: "a concept represented by the synset {x, x1, …} is a meronym of a concept represented by the synset {y, y1, …} if native speakers of

English accept sentences constructed from such frames as ‘A y has an x (as a part)’ or ‘An x is a part of y’” (Miller 1990).

Both meronyms and holonyms in WordNet have three subtypes: substance, part, and member. All three subtypes of meronyms and hypernyms are best mapped to the SNePS part/whole case-frame. For example, a meronymy WordNet search for the noun *tree* produces the following results:

Meronyms of noun tree

1 of 3 senses of tree

Sense 1
tree
HAS SUBSTANCE: sapwood
HAS SUBSTANCE: heartwood, duramen
HAS PART: stump, tree stump
HAS PART: crown, capitulum, treetop
HAS PART: limb, tree branch
HAS PART: trunk, tree trunk, bole
HAS PART: burl

This information is represented in SNePS as the following set of assertions:

```
(assert concept #tree-n-12352501
      word "tree")

(assert concept #sapwood-n-12346093
      word "sapwood")

(assert whole *tree-n-12352501
      part *sapwood-n-12346093)

(assert concept #heartwood-duramen-n-12346309
      word "heartwood")
(assert concept *heartwood-duramen-n-12346309
      word "duramen")

(assert whole *tree-n-12352501
      part *heartwood-duramen-n-12346309)

(assert concept #tree-n-12352501
      word "tree")

(assert concept #stump-tree_stump-n-12359618
      word "stump")
```

*(assert concept *stump-tree_stump-n-12359618*
word "tree_stump")

*(assert whole *tree-n-12352501*
*part *stump-tree_stump-n-12359618)*

(assert concept #crown-capitulum-treetop-n-12375616
word "crown")
*(assert concept *crown-capitulum-treetop-n-12375616*
word "capitulum")
*(assert concept *crown-capitulum-treetop-n-12375616*
word "treetop")

*(assert whole *tree-n-12352501*
*part *crown-capitulum-treetop-n-12375616)*

(assert concept #limb-tree_branch-n-12410108
word "limb")
*(assert concept *limb-tree_branch-n-12410108*
word "tree_branch")

*(assert whole *tree-n-12352501*
*part *limb-tree_branch-n-12410108)*

(assert concept #trunk-tree_trunk-bole-n-12411931
word "trunk")
*(assert concept *trunk-tree_trunk-bole-n-12411931*
word "tree_trunk")
*(assert concept *trunk-tree_trunk-bole-n-12411931*
word "bole")

*(assert whole *tree-n-12352501*
*part *trunk-tree_trunk-bole-n-12411931)*

(assert concept #burl-n-12412160
word "burl")

*(assert whole *tree-n-12352501*
*part *burl-n-12412160)*

5. Verbs in WordNet

Verb Relations Available in WordNet

Three types of verb relations are available in WordNet (Fellbaum 1990). These relations are

1. Lexical Entailment
2. Troponymy
3. Causation

The SNePS representation of lexical entailment, troponymy, and causation will be discussed in the following three sections.

Lexical Entailment and Its SNePS Representation

Below is an excerpt from "English Verbs as a Semantic Net" (Fellbaum 1990:272), in which the notion of lexical entailment between verbs is defined:

> In logic, entailment, or strict implication, is properly defined for propositions; a proposition P entails a proposition Q if and only if there is no conceivable state of affairs that could make P true and Q false. Entailment is a semantic relation because it involves reference to the states of affairs that P and Q represent. The term will be generalized here to refer to the relation between two verbs V1 and V2 that holds when the sentence *Someone V1 logically entails the sentence Someone V2*; this use of entailment can be called lexical entailment. Thus, for example, *snore* lexically entails *sleep* because the sentence *He is snoring* entails *He is sleeping*; the second sentence necessarily holds if the first one does.

Some examples of lexical entailment are

buy entails *pay*
snore entails *sleep*
drive entails *ride*

Because there is no direct mapping between lexical entailment and any standard SNePS case-frame, the new *verb-concept/entails* case-frame is introduced to represent lexical entailment in SNePS. For example, the following set of assertions represents the fact that {buy, purchase} entails {pay} and that {buy, purchase} entails {choose, take, select, pick_out}:

```
(assert concept #buy-purchase-v-2143689
      word "buy")
(assert concept *buy-purchase-v-2143689
      word "purchase")

(assert concept #pay-v-2186811
      word "pay")

(assert verb-concept *buy-purchase-v-2143689
```

*entails *pay-v-2186811)*

(assert concept #choose-take-select-pick_out-v-652154
word "choose")
*(assert concept *choose-take-select-pick_out-v-652154*
word "take")
*(assert concept *choose-take-select-pick_out-v-652154*
word "select")
*(assert concept *choose-take-select-pick_out-v-652154*
word "pick_out")

*(assert verb-concept *buy-purchase-v-2143689*
*entails *choose-take-select-pick_out-v-652154)*

Troponymy and Its SNePS Representation

To extend the notion of hyponymy from nouns to verbs, the creators of WordNet introduced a new semantic relation called *troponymy.* Verbs V1 and V2 are troponyms if *V1 is to V2 in some particular manner*. The notions of troponymy and hypernymy in application to verbs are used interchangeably in the WordNet literature.

For example: *scribble* and *write* are troponyms because WordNet defines *scribble* as *write down quickly without much attention to detail*; i.e., *to scribble* is to *write* in some particular manner (specifically *quickly and without much attention to detail*).

Troponymy and lexical entailment are related: troponymy is a kind of entailment, because every troponym V1 of a more general verb V2 also entails V2. An example from Fellbaum 1990:275:

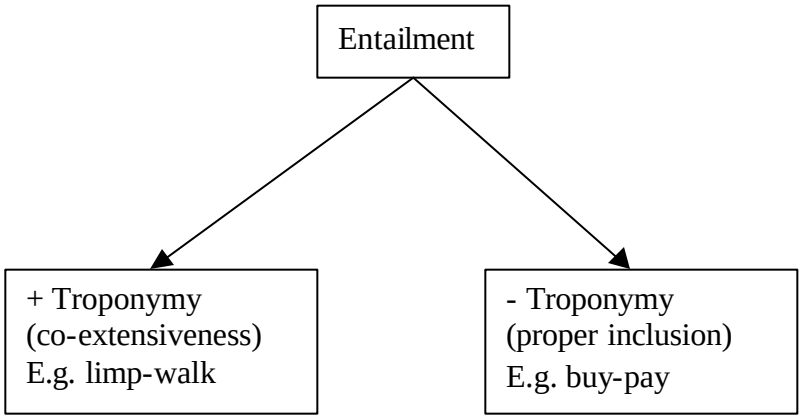
"Consider the pair *limp-walk*. The verbs in this pair are related by troponymy: *to limp* is also *to walk in a certain manner*; *limp* is a troponym of *walk*. The verbs are also in an entailment relation: the statement *He is limping* entails *He is walking*, and walking can be said to be a part of limping."

Whether a pair of verbs in an entailment relation are also in a troponymy relation is determined by the temporal relationship between these verbs. For every pair of troponyms V1 andV2, it must necessarily hold that V1 and V2 are temporally co-extensive. For example, the activities corresponding to the verbs *to scribble* and *to write* are temporally co-extensive, because one must necessarily be writing every moment that one is scribbling.

In order to discuss temporal relationships between troponymy and entailment, the WordNet creators introduced the notion of *Temporal Inclusion*, which they defined as follows:

"A verb V1 will be said to *temporally include* a verb V2 if there is some stretch of time during which the activities denoted by the two verbs co-occur, but no time during which V2 occurs and V1 does not. If there is a time during which V1 occurs but V2 does not, V1 will be said to *properly include* V2." (Fellbaum 1990)

For example, such verb pairs as buy-pay and get_a_medical_check-up/visit_the_doctor[1] are bound by proper inclusion. The pairs bound by proper inclusion cannot be related by troponymy, because troponym verbs must be temporally co-extensive. These ideas are summarized in the following diagram:



Verbs in an entailment relation are also in a troponymy relation if these verbs are temporally co-extensive.

The notion of troponymy is best represented in SNePS by the subclass/superclass case-frame. For example, to demonstrate that the synset {limp, hobble, hitch} has a more general troponym synset {walk}, the following set of assertions is used:

(assert concept #limp-hobble-hitch-v-1861283
word "limp")
*(assert concept *limp-hobble-hitch-v-1861283*
word "hobble")
*(assert concept *limp-hobble-hitch-v-1861283*
word "hitch")

(assert concept #walk-v-1849285
word "walk")

*(assert subclass *limp-hobble-hitch-v-1861283*
*superclass *walk-v-1849285)*

[1] The convention for dealing with empty spaces in WordNet is to replace every empty space with an underscore.

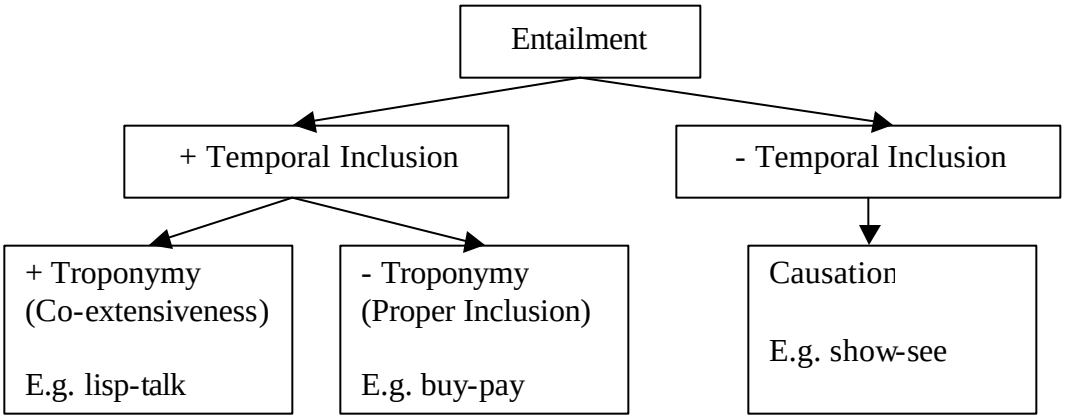
Causation and Its SNePS Representation

The causative relation between two concepts (currently only in application to verbs) is the relation when one of the concepts can be identified as *causative* and the other one as *resultative.* In contrast to other WordNet relations, subjects of the verbs in a causative/resultative pair have two distinctly different referents. The subject of the resultative verb must be an object of the causative verb.

Some examples of causative/resultative pairs are:

give/have
show/see
kill/die

Causation is related to lexical entailment in that it is a specific case of entailment: if V1 necessarily causes V2, then V1 also entails V2. However, causation is different from troponymy in that it lacks temporal inclusion. The relationship between entailment, troponymy and causation is summarized in the following diagram:



It can be seen from this diagram that not all verbs related by entailment are also related by temporal inclusion: while verbs in troponymy relation are always temporally co-extensive, verbs in causative relation are not. To summarize, the causation relation between verbs is a specific kind of entailment characterized by the absence of temporal inclusion.

To represent causation in SNePS, the *cause/effect* case frame is introduced. For a pair of verbs that are bound by causation relation, the synset pointed to by the *cause* arc corresponds to the causative part of the pair, while the synset pointed to by the *effect* arc corresponds to the resultative part of the pair. For example, the following assertions represent the fact that a pair of synsets {show}/{see} are bound by causation:

(assert concept #show-v-2075197
word "show")

(assert concept #see-v-2067665
word "see")

*(assert cause *show-v-2075197*
*effect *see-v-2067665)*

6. WordNet-to-SNePS (WNS) Interface

Overview

The process of retrieving data from the WordNet KB and converting this data to SNePSUL is automated by means of a computer program called *wns*. The program is implemented in the C language and makes use of the WordNet API that simplifies the basic operations of searching the WordNet KB and is distributed as a part of the WordNet installation package.

The functionality of *wns* includes the ability to perform a WordNet KB search: given a word form (a one-word noun, a one-word verb, or a collocation with empty spaces replaced by the underscore character) *wns* will

1. retrieve the synsets corresponding to every sense of this word form that is available in the WordNet KB
2. retrieve and output the textual gloss corresponding to every sense of this word that is available the WordNet KB
3. retrieve the information about the following relations for this word form:

    - Noun Antonymy
    - Noun Hypernymy
    - Noun Synonymy
    - Noun Holonymy (all three types)
    - Noun Meronymy (all three types)
    - Verb Antonymy
    - Verb Hypernymy (troponymy)
    - Verb Synonymy
    - Verb Entailment
    - Verb Causation

4. convert and output the information retrieved in steps (1) and (3) and convert it to SNePSUL.

The *wns* program has a built-in morphological processor and thus is capable of handling inflected word forms.

The executable *wns* is currently stored in

/projects/rapaport/Wordnet/WordNet-2.0/src/wns-1.0/

and is available for public use.

High-Level Design

wns operates in accordance with the following algorithm.

The algorithm is simplified and is intended to provide only a high-level understanding of how of *wns* works; the source code of *wns* is well documented, so please see the code in appendix 3 for details.

```
Begin

Read the word form W passed as an argument

For every WordNet noun relation R and word form W
{
        perform a noun WordNet search for word form W and relation R

        for every sense S of word form W retrieved from WordNet KB
        {
                convert information about relation R and word form W for sense S
                        to SNePSUL and output this information
        }
}

For every WordNet verb relation R and word form W
{
        perform a verb WordNet search for word form W and relation R

        for every sense S of word form W retrieved from WordNet KB
        {
                convert information about relation R and word form W for sense S
                        to SNePSUL and output this information
        }
}

End
```

There are two ways to run the *wns* program:

1. Pass a single word form as an argument: wns –word <some_word_form>

For example:

wns –word head
wns –word computer_program

The abridged output produced by executing *wns –word head* is shown in the Appendix section of this report (for each type of search, only the first three senses are left).

2. Pass a file name: wns –file <some_file_name>

For example:

wns –file context.txt

The *wns* program includes a tokenizer. The tokenizer goes through the file that is passed as an argument to the program and performs a WordNet search for each word form in the file. The job of the tokenizer is to get rid of all the "extraneous" characters, such as commas, periods, spaces. As a result, the program is capable of processing a passage that is taken from an arbitrary text: the passage can be saved in a separate text file and the name of the file can be passed to the program. In this manner, *wns* can be used to import the entire content of the WordNet KB to SNePS if such a need arises.

Conversion Rules

Because all WordNet relations are defined for synsets and not for individual word forms and the output of the *wns* program reflects this state of affairs, a suite of conversion rules is necessary to adapt the output of *wns* for use in CVA. This task is accomplished by means of the rules which semantically interpret the *wns*-produced output and convert it to SNePS assertions relating individual word forms. These assertions conform to the CVA standards and thus can be picked up by the CVA word definition algorithms.

The first rule is used to convert the *wns* representation of synsets into the CVA-compliant representation that uses the synonym/synonym case frame:

```
(add forall ($concept $word1 $word2)
   &ant (
       (build concept *concept word *word1)
       (build concept *concept word *word2)
      )
   cq   (build synonym (build lex *word1)
            synonym (build lex *word2)))
```

The following three rules will convert the antonym/antonym, subclass/superclass, and part/whole relations which hold between concepts (synsets) to the CVA-compliant relations that hold between individual words (synsets members):

```
(add forall ($concept1 $concept2 $word1 $word2)
    &ant (
        (build antonym *concept1 antonym *concept2)
        (build concept *concept1 word *word1)
        (build concept *concept2 word *word2)
       )
    cq   (build antonym (build lex *word1)
             antonym (build lex *word2)))

(add forall ($sub-concept $super-concept $sub-word $super-word)
    &ant (
        (build subclass *sub-concept superclass *super-concept)
        (build concept *sub-concept word *sub-word)
        (build concept *super-concept word *super-word)
       )
    cq   (build subclass (build lex *sub-word)
             superclass (build lex *super-word)))

(add forall($whole-concept $part-concept $whole-word $part-word)
    &ant (
        (build whole *whole-concept part *part-concept)
        (build concept *whole-concept word *whole-word)
        (build concept *part-concept word *part-word)
       )
    cq   (build part  (build lex *part-word)
             whole (build lex *whole-word)))
```

The following two rules will convert entailment and causation relations to the representations that use the standard CVA agent/act/action case frame.

```
(add forall ($action-concept $action-word $entails-concept $entails-word)
    &ant (
        (build verb-concept *action-concept entails *entails-concept)
        (build concept *action-concept word *action-word)
        (build concept *entails-concept word *entails-word)
        )
    cq   (build forall ($x)
             ant (build agent *x
                      act (build action (build lex *action-word)))
             cq  (build agent *x
                      act (build action (build lex *entails-word)))))
```

```
(add forall ($cause-concept $cause-word $effect-concept $effect-word)
   &ant (
      (build cause *cause-concept effect *effect-concept)
      (build concept *cause-concept word *cause-word)
      (build concept *effect-concept word *effect-word)
      )
   cq  (build forall ($subject $object)
         ant (build agent *subject
               act (build action (build lex *cause-word)
                     object *object))
         cq  (build agent *object
               act (build action (build lex *effect-word)))))
```

7. CVA Demo

The demo created for the purpose of this project will be referred to as the *augur-wns* demo to distinguish it from the original *augur* demo created by Chris Becker (Becker 2004) to computationally determine the meaning of the verb *augur* from context and thus test the capabilities of his verb-definition algorithm.

The purpose of the *augur-wns* demo is to show how the automated WordNet-to-SNePS interface (*wns)* can be used to retrieve some background knowledge necessary for CVA from WordNet.

The following passage served as the context:

Suddenly the tempest redoubled. The poor young woman could ***augur*** *nothing favorable as she listened to the threatening heavens, the changes of which were interpreted in those credulous days according to the ideas or the habits of individuals.*

The original *augur* demo already made a selective use of some WordNet ontologies, which Chris Becker manually converted to SNePSUL. Specifically, the demo contains a partial chain of superordinates for the nouns *tempest, woman, days,* and *quality,* and the verbs *interpret, listen,* and *redouble.* For the purpose of the *augur-wns* demo, the full chain of superordinates for these words was automatically retrieved and converted to SNePSUL using the *wns* program. Because the *wns* program retrieves all the senses of the sought word that are defined in WordNet, the correct sense had to be manually extracted from the output of *wns* and pasted to the *augur-wns* demo (see the "Future Research Directions" section for more on this). In addition to this, in order to make the output of *wns* recognizable by the verb-definition algorithm, the suite of conversion rules described in the previous section was loaded to ensure that the synset relations generated by *wns* are converted to the CVA-compliant rules. This step is accomplished by the line

(intext "conv-rules") in the *augur-wns* demo file (the rules were stored in the file named *conv-rules* in the same directory where the demo was run).

Below is the verb-definition algorithm's output for the original *augur* demo:

** * * Defining the verb augur * * **

Arguments of the verb:
(agent object)

Transitivity:
transitive

** * * Basic Findings * * **

possible cause of augur is:
bad omen,
heavens,
instance of change,
tempest,

possible actions performed by agent of augur is:
augur,
interpret,
listen,

possible actions performed on object of augur is:
augur,

possible actions performed with instrument of augur is:
augur,

possible action that is the cause of augur is:
listen,
interpret,

possible property of verb is:
unknown,

possible membership of agent is:
human,

possible superordinate of agent is:
animate thing,

possible property of agent is:
disposed to believe on little evidence,
credulous,
poor,
young,

possible superclass of object is:
quality,

possible property of object is:
not favorable,

** * * generalizations from this context * * **

A {human} can augur
Something that is a subclass of {quality}

A {human} can augur
Something with the properties {not favorable}

A {animate thing} can augur
Something that is a subclass of {quality}

A {animate thing} can augur
Something with the properties {not favorable}

Something with the properties {disposed to believe on little evidence, credulous, poor, young} can augur
Something that is a subclass of {quality}

Something with the properties {disposed to believe on little evidence, credulous, poor, young} can augur
Something with the properties {not favorable}

The definition of *augur* produced by the *augur-wns* demo is a superset of the definition produced by the original *augur* demo:

** * * Defining the verb augur * * **

Arguments of the verb:
(agent object)

Transitivity: transitive

** * * Basic Findings * * **

possible cause of augur is:
bad omen,
heavens,
instance of change,
tempest,

possible actions performed by agent of augur is:
augur,
interpret,
listen,

possible actions performed on object of augur is:
augur,

possible actions performed with instrument of augur is:
augur,

possible action that is the cause of augur is:
listen,

interpret,

possible property of verb is:
unknown,

possible membership of agent is:
human,

possible superordinate of agent is:
entity,
physical_object,
object,
animate_thing,
living_thing,
being,
organism,

possible property of agent is:
disposed to believe on little evidence,
credulous,
poor,
young,

possible superclass of object is:
quality,

possible property of object is:
not favorable,

** * * generalizations from this context * * **

A {human} can augur
Something that is a subclass of {quality}

A {human} can augur
Something with the properties {not favorable}

A {entity, physical_object, object, animate_thing, living_thing, being, organism} can augur
Something that is a subclass of {quality}

A {entity, physical_object, object, animate_thing, living_thing, being, organism} can augur
Something with the properties {not favorable}

Something with the properties {disposed to believe on little evidence, credulous, poor, young} can augur
Something that is a subclass of {quality}

Something with the properties {disposed to believe on little evidence, credulous, poor, young} can augur
Something with the properties {not favorable}

The explanation of the fact that the *wns-augur* demo produces the output that is the superset of the original *augur* demo is rooted in that the chains of superordinates, which were manually adapted from WordNet for the *augur* demo, are incomplete, while the *wns* program extracts whatever is available in the WordNet KB for the given word form.

8. Future Research Directions

WordNet-related Research

1. The *wns* program currently retrieves from WordNet all the senses for the word form passed to it as an argument. However, when a context for this word form is provided, only a single sense is applicable. In other words, a word-sense disambiguation technique needs to be developed, which would allow a decision about which sense of all the senses that are available in WordNet KB for the given word form is applicable in the given context.

An interesting project would be to investigate the applicability of the CVA techniques to the problem of word-sense disambiguation. The following strategy is conceivable:

Suppose we are given a word form W in some context. We could run a CVA algorithm on W and try to match its output against the *wns*-produced ontologies for each of the senses of W. A match would point to the sense of W that is used in this context.

2. Because WordNet verb relations have many temporal aspects (see the section on WordNet verb relations), the conversion rules, which interpreted the *wns*-produced assertions for CVA use, could include the concept of time. For example, the rule that interprets the *cause/effect* case frame could include the information that the causative and resultative verbs lack temporal inclusion.

3. Since WordNet verb relations are interrelated (e.g., troponymy is a case of entailment), the conversion rules could be updated to reflect this fact. For example, if an assertion links two synsets as being in a troponymy relation, this assertion can be interpreted in the same way as if these two synsets were in an entailment relation.

CYC-related Research

1. An interesting research direction would be to develop a theory of how a human reader identifies the background knowledge that is necessary for CVA. The major problem with using the CYC KB as a source of background knowledge is the lack of understanding of how a human reader decides what kind of background knowledge is necessary for CVA. Developing such a theory would make searching CYC for the relevant background knowledge possible.

2. Another approach to using CYC as a source of background knowledge for CVA could be to re-implement the CVA algorithms in CYCL/SUBL (CYC programming languages). This would obviate the need to search for and import the background knowledge to SNePS.

3. A problem with using CYC as a source of background knowledge for CVA is having to map CYC predicates to the standard CVA case-frames. A case-frame independent CVA algorithm, that is an algorithm that does not rely on a set of standard case-frames to search for, would obviate the need for such a mapping. A case-frame independent CVA algorithm could conceivably search the semantic network around an unknown word for *all* assertions that involves this word (and not just the assertions that use the standard case-frames).

Appendix 1

wns program sample run

/home/Mitya/WordNet-2.0/src/wns/>wns -word head

```
; ------------------------------------------------
; ------------------------------------------------
;  antonymy search results for noun "head"
; ------------------------------------------------
; ------------------------------------------------

; --------------- sense 1 ------------------------
; (the front of a military formation or process
; ion; "the head of the column advanced boldly"
; ; "they were at the head of the attack")
; ------------------------------------------------

(assert concept #head-n-7968063
        word "head")

(assert concept #rear-n-7968474
        word "rear")

(assert antonym *head-n-7968063
        antonym *rear-n-7968474)

; --------------- sense 2 ------------------------
; (the top of something; "the head of the stair
; s"; "the head of the page"; "the head of the
; list")
; ------------------------------------------------

(assert concept #head-n-8134688
        word "head")

(assert concept #foot-n-7993741
        word "foot")

(assert antonym *head-n-8134688
        antonym *foot-n-7993741)

; --------------- sense 3 ------------------------
; ((usually plural) an obverse side of a coin t
; hat bears the representation of a person's he
; ad; "call heads or tails!")
; ------------------------------------------------

(assert concept #head-n-3373458
        word "head")

(assert concept #tail-n-4214453
        word "tail")
```

```
(assert antonym *head-n-3373458
        antonym *tail-n-4214453)


; ------------------------------------------------
; ------------------------------------------------
;  hypernymy search results for noun "head"
; ------------------------------------------------
; ------------------------------------------------

; --------------- sense 1 ------------------------
; (the upper part of the human body or the fron
; t part of the body in animals; contains the f
; ace and brains; "he stuck his head out the wi
; ndow")
; ------------------------------------------------

(assert concept #head-caput-n-5221598
        word "head")
(assert concept *head-caput-n-5221598
        word "caput")

(assert concept #external_body_part-n-4924211
        word "external_body_part")

(assert subclass *head-caput-n-5221598
        superclass *external_body_part-n-4924211)

(assert concept #body_part-n-4919813
        word "body_part")

(assert subclass *external_body_part-n-4924211
        superclass *body_part-n-4919813)

(assert concept #part-piece-n-8797461
        word "part")
(assert concept *part-piece-n-8797461
        word "piece")

(assert subclass *body_part-n-4919813
        superclass *part-piece-n-8797461)

(assert concept #thing-n-2056
        word "thing")

(assert subclass *part-piece-n-8797461
        superclass *thing-n-2056)

(assert concept #entity-n-1740
        word "entity")

(assert subclass *thing-n-2056
        superclass *entity-n-1740)

; --------------- sense 2 ------------------------
; (a single domestic animal; "200 head of cattl
```

```
; e")
; ---------------------------------------------

(assert concept #head-n-1245172
        word "head")

(assert concept #domestic_animal-n-1244626
        word "domestic_animal")

(assert subclass *head-n-1245172
        superclass *domestic_animal-n-1244626)

(assert concept #animal-animate_being-beast-brute-creature-fauna-n-
12748
        word "animal")
(assert concept *animal-animate_being-beast-brute-creature-fauna-n-
12748
        word "animate_being")
(assert concept *animal-animate_being-beast-brute-creature-fauna-n-
12748
        word "beast")
(assert concept *animal-animate_being-beast-brute-creature-fauna-n-
12748
        word "brute")
(assert concept *animal-animate_being-beast-brute-creature-fauna-n-
12748
        word "creature")
(assert concept *animal-animate_being-beast-brute-creature-fauna-n-
12748
        word "fauna")

(assert subclass *domestic_animal-n-1244626
        superclass *animal-animate_being-beast-brute-creature-fauna-n-
12748)

(assert concept #organism-being-n-3226
        word "organism")
(assert concept *organism-being-n-3226
        word "being")

(assert subclass *animal-animate_being-beast-brute-creature-fauna-n-
12748
        superclass *organism-being-n-3226)

(assert concept #living_thing-animate_thing-n-3009
        word "living_thing")
(assert concept *living_thing-animate_thing-n-3009
        word "animate_thing")

(assert subclass *organism-being-n-3226
        superclass *living_thing-animate_thing-n-3009)

(assert concept #object-physical_object-n-16236
        word "object")
(assert concept *object-physical_object-n-16236
        word "physical_object")
```

```
(assert subclass *living_thing-animate_thing-n-3009
        superclass *object-physical_object-n-16236)

(assert concept #entity-n-1740
        word "entity")

(assert subclass *object-physical_object-n-16236
        superclass *entity-n-1740)

; --------------- sense 3 ----------------------
; (that which is responsible for one's thoughts
;  and feelings; the seat of the faculty of rea
; son; "his mind wandered"; "I couldn't get his
;  words out of my head")
; ------------------------------------------------

(assert concept #mind-head-brain-psyche-nous-n-5291080
        word "mind")
(assert concept *mind-head-brain-psyche-nous-n-5291080
        word "head")
(assert concept *mind-head-brain-psyche-nous-n-5291080
        word "brain")
(assert concept *mind-head-brain-psyche-nous-n-5291080
        word "psyche")
(assert concept *mind-head-brain-psyche-nous-n-5291080
        word "nous")

(assert concept #cognition-knowledge-noesis-n-20729
        word "cognition")
(assert concept *cognition-knowledge-noesis-n-20729
        word "knowledge")
(assert concept *cognition-knowledge-noesis-n-20729
        word "noesis")

(assert subclass *mind-head-brain-psyche-nous-n-5291080
        superclass *cognition-knowledge-noesis-n-20729)

(assert concept #psychological_feature-n-20333
        word "psychological_feature")

(assert subclass *cognition-knowledge-noesis-n-20729
        superclass *psychological_feature-n-20333)

; ------------------------------------------------
; ------------------------------------------------
;  synonymy search results for noun "head"
; ------------------------------------------------
; ------------------------------------------------

; --------------- sense 1 ----------------------
; (the upper part of the human body or the fron
; t part of the body in animals; contains the f
; ace and brains; "he stuck his head out the wi
; ndow")
; ------------------------------------------------

(assert concept #head-caput-n-5221598
```

```
        word "head")
(assert concept *head-caput-n-5221598
        word "caput")

(assert concept #external_body_part-n-4924211
        word "external_body_part")

(assert subclass *head-caput-n-5221598
        superclass *external_body_part-n-4924211)

; --------------- sense 2 -----------------------
; (a single domestic animal; "200 head of cattl
; e")
; -----------------------------------------------

(assert concept #head-n-1245172
        word "head")

(assert concept #domestic_animal-n-1244626
        word "domestic_animal")

(assert subclass *head-n-1245172
        superclass *domestic_animal-n-1244626)

; --------------- sense 3 -----------------------
; (that which is responsible for one's thoughts
;  and feelings; the seat of the faculty of rea
; son; "his mind wandered"; "I couldn't get his
;  words out of my head")
; -----------------------------------------------

(assert concept #mind-head-brain-psyche-nous-n-5291080
        word "mind")
(assert concept *mind-head-brain-psyche-nous-n-5291080
        word "head")
(assert concept *mind-head-brain-psyche-nous-n-5291080
        word "brain")
(assert concept *mind-head-brain-psyche-nous-n-5291080
        word "psyche")
(assert concept *mind-head-brain-psyche-nous-n-5291080
        word "nous")

(assert concept #cognition-knowledge-noesis-n-20729
        word "cognition")
(assert concept *cognition-knowledge-noesis-n-20729
        word "knowledge")
(assert concept *cognition-knowledge-noesis-n-20729
        word "noesis")

(assert subclass *mind-head-brain-psyche-nous-n-5291080
        superclass *cognition-knowledge-noesis-n-20729)


; -----------------------------------------------
; -----------------------------------------------
;  holonymy search results for noun "head"
; -----------------------------------------------
```

```
; ------------------------------------------------

; --------------- sense 1 -----------------------
; (the upper part of the human body or the fron
; t part of the body in animals; contains the f
; ace and brains; "he stuck his head out the wi
; ndow")
; ------------------------------------------------

(assert concept #head-caput-n-5221598
        word "head")
(assert concept *head-caput-n-5221598
        word "caput")

(assert concept #body-organic_structure-physical_structure-n-4916110
        word "body")
(assert concept *body-organic_structure-physical_structure-n-4916110
        word "organic_structure")
(assert concept *body-organic_structure-physical_structure-n-4916110
        word "physical_structure")

(assert whole *body-organic_structure-physical_structure-n-4916110
        part *head-caput-n-5221598)

(assert concept #animal-animate_being-beast-brute-creature-fauna-n-
12748
        word "animal")
(assert concept *animal-animate_being-beast-brute-creature-fauna-n-
12748
        word "animate_being")
(assert concept *animal-animate_being-beast-brute-creature-fauna-n-
12748
        word "beast")
(assert concept *animal-animate_being-beast-brute-creature-fauna-n-
12748
        word "brute")
(assert concept *animal-animate_being-beast-brute-creature-fauna-n-
12748
        word "creature")
(assert concept *animal-animate_being-beast-brute-creature-fauna-n-
12748
        word "fauna")

(assert whole *animal-animate_being-beast-brute-creature-fauna-n-12748
        part *head-caput-n-5221598)

; --------------- sense 2 -----------------------
; (the tip of an abscess (where the pus accumul
; ates))
; ------------------------------------------------

(assert concept #head-n-13501568
        word "head")

(assert concept #abscess-n-13501259
        word "abscess")
```

```
(assert whole *abscess-n-13501259
        part *head-n-13501568)

; --------------- sense 3 ----------------------
; (a V-shaped mark at one end of an arrow point
; er; "the point of the arrow was due north")
; ----------------------------------------------

(assert concept #point-head-n-6402917
        word "point")
(assert concept *point-head-n-6402917
        word "head")

(assert concept #arrow-pointer-n-6399245
        word "arrow")
(assert concept *arrow-pointer-n-6399245
        word "pointer")

(assert whole *arrow-pointer-n-6399245
        part *point-head-n-6402917)


; ----------------------------------------------
; ----------------------------------------------
;  meronymy search results for noun "head"
; ----------------------------------------------
; ----------------------------------------------

; --------------- sense 1 ----------------------
; (the upper part of the human body or the fron
; t part of the body in animals; contains the f
; ace and brains; "he stuck his head out the wi
; ndow")
; ----------------------------------------------

(assert concept #head-caput-n-5221598
        word "head")
(assert concept *head-caput-n-5221598
        word "caput")

(assert concept #muzzle-n-2364732
        word "muzzle")

(assert whole *head-caput-n-5221598
        part *muzzle-n-2364732)

(assert concept #ear-n-5014060
        word "ear")

(assert whole *head-caput-n-5221598
        part *ear-n-5014060)

(assert concept #basilar_artery-arteria_basilaris-n-5031199
        word "basilar_artery")
(assert concept *basilar_artery-arteria_basilaris-n-5031199
        word "arteria_basilaris")
```

```
(assert whole *head-caput-n-5221598
        part *basilar_artery-arteria_basilaris-n-5031199)

(assert concept #brain-encephalon-n-5166469
        word "brain")
(assert concept *brain-encephalon-n-5166469
        word "encephalon")

(assert whole *head-caput-n-5221598
        part *brain-encephalon-n-5166469)

(assert concept #skull-n-5223094
        word "skull")

(assert whole *head-caput-n-5221598
        part *skull-n-5223094)

(assert concept #face-human_face-n-5280660
        word "face")
(assert concept *face-human_face-n-5280660
        word "human_face")

(assert whole *head-caput-n-5221598
        part *face-human_face-n-5280660)

(assert concept #temple-n-5282670
        word "temple")

(assert whole *head-caput-n-5221598
        part *temple-n-5282670)

; --------------- sense 2 -----------------------
; (the striking part of a tool; "the head of th
; e hammer")
; -----------------------------------------------

(assert concept #head-n-3373261
        word "head")

(assert concept #face-n-3193650
        word "face")

(assert whole *head-n-3373261
        part *face-n-3193650)
```

Appendix 2

CVA Demo

```
* (demo "augur-wn.demo")


File /home/csgrad/ddligach/IndProj/Demo/augur-wn.demo is now the source
of input.




 CPU time : 0.01


* ;
;=========================================================================

; FILENAME:     augur-wn.demo

; DATE:         4/15/2004

; PROGRAMMER:   Chris Becker

; PROGRAMMER:   Dmitriy Dligach

;
;=========================================================================


; Turn off inference tracing.

; This is optional;

;^(setq snip:*infertrace* nil)


; Load the verb definition algorithm:

^(

--> load "/projects/rapaport/CVA/verbalgorithm3.0/defun_verb.cl")

;   Loading /projects/rapaport/CVA/verbalgorithm3.0/defun_verb.cl
```

```
;     Loading
;        /projects/rapaport/CVA/verbalgorithm3.0/ConstructFindLists.cl
;     Loading
;        /projects/rapaport/CVA/verbalgorithm3.0/ConstructFoundLists.cl
;     Loading /projects/rapaport/CVA/verbalgorithm3.0/Processing.cl
;     Loading /projects/rapaport/CVA/verbalgorithm3.0/Output.cl
t



 CPU time : 0.06

*
; Clear the SNePS network:
(resetnet t)

Net reset


 CPU time : 0.00

*
;turn on full forward inferencing:
^(
--> setq snip:*infertrace* nil)
nil
```

```
 CPU time : 0.00


*
; ;enter the "snip" package:
^(
--> in-package snip)
#<The snip package>




 CPU time : 0.00


*
^(
--> defun broadcast-one-report (rep)
        (let (anysent)
          (do.chset (ch *OUTGOING-CHANNELS* anysent)
                (when (isopen.ch ch)
                   (setq anysent (or (try-to-send-report rep ch)
anysent)))))
        nil)
broadcast-one-report




 CPU time : 0.00


*
```

```
;^(in-package sneps)

; load all pre-defined relations:
(intext "/projects/rapaport/CVA/STN2/demos/rels")
File /projects/rapaport/CVA/STN2/demos/rels is now the source of input.


 CPU time : 0.00

*

(a1 a2 a3 a4 after agent against antonym associated before cause class
 direction equiv etime event from in indobj instr into lex location
 manner member mode object on onto part place possessor proper-name
 property rel skf sp-rel stime subclass superclass subset superset
 synonym time to whole kn_cat)

 CPU time : 0.05

*

End of file /projects/rapaport/CVA/STN2/demos/rels


 CPU time : 0.05

*
; define WordNet relations
```

```
(define concept word)


(concept word)


 CPU time : 0.00


*

; define new relations:

(define similar)


(similar)


 CPU time : 0.00


*

; load all pre-defined path definitions:

(intext "/projects/rapaport/CVA/STN2/demos/paths")

File /projects/rapaport/CVA/STN2/demos/paths is now the source of
input.




 CPU time : 0.00


*

before implied by the path (compose before

                            (kstar (compose after- ! before)))

before- implied by the path (compose (kstar (compose before- ! after))

                             before-)
```

```
 CPU time : 0.01

*
after implied by the path (compose after
                           (kstar (compose before- ! after)))
after- implied by the path (compose (kstar (compose after- ! before))
                            after-)


 CPU time : 0.00

*
class implied by the path (compose class
                           (kstar (compose subclass- ! superclass)))
class- implied by the path (compose
                            (kstar (compose superclass- ! subclass))
                            class-)


 CPU time : 0.00

*
subclass implied by the path (compose subclass
                              (kstar (compose superclass- ! subclass)))
subclass- implied by the path (compose
                               (kstar (compose subclass- ! superclass))
                               subclass-)
```

```
 CPU time : 0.00

*
member implied by the path (compose member
                            (kstar (compose equiv- ! equiv)))
member- implied by the path (compose (kstar (compose equiv- ! equiv))
                             member-)


 CPU time : 0.00

*

End of file /projects/rapaport/CVA/STN2/demos/paths


 CPU time : 0.01

*
;;---------------------------------------------------------------
;;---------------------------------------------------------------

; THE CONTEXT:
; ============

; Suddenly the tempest redoubled. The poor young woman could augur
```

```
; nothing favorable as she listened to the threatening heavens, the

; changes of which were interpreted in those credulous days according
to

; the ideas or the habits of individuals.

;

; from: http://www.gutenberg.net/etext98/htdsn10.txt

;

;;-----------------------------------------------------------------


; PARAPHRASED:

;

; The tempest redoubled.

; (ant:) The woman listened to the heavens.

; (cq:) The woman augured a thing that was not favorable.

; In those days, people interpreted a change in the heavens.

; In credulous days, people are credulous.

; The act of interpretation, in those (credulous) days,

;       is done by the relation of "manner"

;       with respect to "ideas" and "habits" possessed by the agent,

;;-----------------------------------------------------------------


; OBJECTS:


; tempest

; woman

; (no)thing

; heavens

; change

; days
```

```
; individuals

;;-----------------------------

; PROPERTIES OF OBJECTS

; tempest
;       act: redouble
;       equiv: heavens
;
; woman
;       property: young
;       property: poor
;       act: augur
;       act: listen(heavens)
;
; thing
;       property: not favorable
; heavens
;       equiv: tempest
;       property: threatening
;       possesses: change
; change
;
; individuals
;       act: interpret change
;;-----------------------------
```

```
; BACKGROUND KNOWLEDGE:

; ======================

;;-------------------------------------------------------------

; General rules:

;

; If action A is performed by agent Y on object Z,

; and action B is performed by agent Y on object Z,

; then A and B are similar.

;

; If action A and action B are similar, and A causes C,

; then B causes C as well.

;

; If X and Y are equiv, and X has property A, then Y has property A.

; If X and Y are equiv, and X does act A, then Y does act A.

; If X and Y are equiv, and X is a member of class A,

;       then Y is a member of class A.

; If X and Y are equiv, and X is a subclass of A, then Y is a subclass
of A.

;;-------------------------------------------------------------



;;-------------------------------------------------------------

;;; If action A is performed by agent Y on object Z,

;;; and action B is performed by agent Y on object Z,

;;; then A and B are similar.

;;-------------------------------------------------------------



;(show

(describe (add forall ($A $B $Y $Z)

        &ant (  (build
```

```
                        agent *Y
                        act (build action *A object *Z)
                    )
                   (build
                        agent *Y
                        act (build action *B object *Z)
                    )
                )
          cq (build similar *A similar *B)
))

(m1! (forall v4 v3 v2 v1)
 (&ant (p4 (act (p3 (action v2) (object v4))) (agent v3))
  (p2 (act (p1 (action v1) (object v4))) (agent v3)))
 (cq (p5 (similar v2 v1))))

(m1!)

 CPU time : 0.01

* ;)

;;-------------------------------------------------------------
;;; If action I and action J are similar, and the act containing
;;; action I causes K, then the act containing action J causes K as
well.
;;;
;;; Note: the actions do not need to have the same agent or object.
;;-------------------------------------------------------------
```

```
;; $I and $J are actions
;; $W and $V are agents
;; $U and $T are objects

;(show
(describe (add forall ($I $J $K $W $V $U $T)
        &ant (
                (build similar *I similar *J)

                (build  cause (build    agent *W
                                        act (build
                                                action *I
                                                object *U))
                        effect *K) )

        cq (build cause (build agent *V act (build action *J object
*T))
                  effect *K)
))

(m2! (forall v11 v10 v9 v8 v7 v6 v5)
 (&ant
  (p9 (cause (p8 (act (p7 (action v5) (object v10))) (agent v8)))
   (effect v7))
  (p6 (similar v6 v5)))
 (cq
  (p12 (cause (p11 (act (p10 (action v6) (object v11))) (agent v9)))
   (effect v7))))
```

```
(m2!)

 CPU time : 0.02

* ;)

;;; GENERIC RULES FOR "EQUIV" CASE FRAME
;;-------------------------------------------------------------
;;; These rules add some meaning to what it means for
;;; two things to be equiv.

;;-------------------------------------------------------------
;;; Equivalent things will share the same properties
;;;
;;; If X and Y are equiv, and X has property A,
;;; then Y also has property A.
;;-------------------------------------------------------------

;(show
(describe (add forall ($x1 $y1 $a1)
        &ant (
                (build equiv *x1 equiv *y1)
                (build object *x1 property *a1)
              )

        cq (build object *y1 property *a1)
))
```

```
(m3! (forall v14 v13 v12)
 (&ant (p14 (object v12) (property v14)) (p13 (equiv v13 v12)))
 (cq (p15 (object v13) (property v14))))

(m3!)

 CPU time : 0.01

* ;)

;;-------------------------------------------------------------
;;; Equivalent things will do the same actions
;;;
;;; If X and Y are equiv, and X does act A,
;;; then Y also does act A.
;;; Note, this is for an intransitive act
;;-------------------------------------------------------------

(describe (add forall ($x2 $y2 $a2)
        &ant (
                (build equiv *x2 equiv *y2)
                (build agent *x2 act (build action *a2))
             )
        cq (build agent *y2 act (build action *a2))
))

(m4! (forall v17 v16 v15)
```

```
 (&ant (p18 (act (p17 (action v17))) (agent v15))
  (p16 (equiv v16 v15)))
 (cq (p19 (act (p17)) (agent v16))))

(m4!)

 CPU time : 0.02

*
;;-------------------------------------------------------------
;;; Equivalent things will share the same class membership
;;;
;;; If X and Y are equiv, and X is a member of class A,
;;; then Y is also a member of class A.
;;-------------------------------------------------------------
(describe (add forall ($x3 $y3 $a3)
        &ant (  (build equiv *x3 equiv *y3)
                (build member *x3 class *a3) )
        cq (build member *y3 class *a3)
))

(m5! (forall v20 v19 v18)
 (&ant (p21 (class v20) (member v18)) (p20 (equiv v19 v18)))
 (cq (p22 (class v20) (member v19))))

(m5!)

 CPU time : 0.02
```

```
*
;;-------------------------------------------------------
;;; ;;; Equivalent things will share the same superclass
;;;
;;; If X and Y are equiv, and X is a subclass of A,
;;; then Y is also a subclass of A.
;;-------------------------------------------------------

(describe (add forall ($x4 $y4 $a4)
        &ant (  (build equiv *x4 equiv *y4)
                (build subclass *x4 superclass *a4) )
        cq (build subclass *y4 superclass *a4)
))

(m6! (forall v23 v22 v21)
 (&ant (p24 (subclass v21) (superclass v23)) (p23 (equiv v22 v21)))
 (cq (p25 (subclass v22) (superclass v23))))

(m6!)

 CPU time : 0.05

*
;;-------------------------------------------------------
; BACKGROUND KNOWLEDGE AND RULES PROVIDED IN THE CONTEXT
;;-------------------------------------------------------
```

```
;;-------------------------------------------------------
;;; Tempest and heavens refer to the same thing.
;;; We can therefore say that tempest is equiv to heavens
;;-------------------------------------------------------

(describe
        (add
                equiv (build lex "tempest") = tempest
                equiv (build lex "heavens") = heavens
        )
)

(m9! (equiv (m8 (lex heavens)) (m7 (lex tempest))))

(m9!)

 CPU time : 0.02

*
;;-------------------------------------------------------
;;; The tempest redoubled.
;;-------------------------------------------------------

(describe
        (add
                agent *tempest
                act (build action (build lex "redouble") = redouble)
        )
```

```
)

(m13! (act (m11 (action (m10 (lex redouble)))))
 (agent (m8 (lex heavens))))
(m12! (act (m11)) (agent (m7 (lex tempest))))

(m13! m12!)

 CPU time : 0.04

*
;;-------------------------------------------------------------
;;; The woman is young
;;-------------------------------------------------------------

(describe
        (add
                object (build lex "the woman") = theWoman
                property (build lex "young")
        )
)

(m16! (object (m14 (lex the woman))) (property (m15 (lex young))))

(m16!)

 CPU time : 0.02
```

```
*
;;-------------------------------------------------------
;;; The woman is poor
;;-------------------------------------------------------

(describe
        (add
                object *theWoman
                property (build lex "poor")
        )
)

(m18! (object (m14 (lex the woman))) (property (m17 (lex poor))))

(m18!)

 CPU time : 0.01

*
;;-------------------------------------------------------
;;; The woman is a member of the class human
;;-------------------------------------------------------

(describe (assert
                member *theWoman
                class (build lex "human") = human))

(m20! (class (m19 (lex human))) (member (m14 (lex the woman))))
```

```
(m20!)


 CPU time : 0.00


*
;;-------------------------------------------------------------
;;; The object of "augur" is "nothing favorable".
;;;
;;; However, since it would be silly to create an object
;;; called "nothing" with the property "favorable".
;;;
;;; Other possibilities could have been to represent it as
;;; "A thing that is not favorable".
;;;
;;; However, I settled on creating an object that is just the
;;; concept "nothing favorable" with the property "not favorable"
;;; (or just "unfavorable")
;;-------------------------------------------------------------


(describe
        (add
                object (build lex "nothing favorable") =
nothingFavorable
                property (build lex "not favorable") = unfavorable
        )
)


(m23! (object (m21 (lex nothing favorable)))
```

```
 (property (m22 (lex not favorable))))

(m23!)

 CPU time : 0.04

*
;;-------------------------------------------------------------
;;; The heavens are threatening
;;-------------------------------------------------------------

(describe
        (add
                object *heavens
                property (build lex "threatening") = threatening
        )
)

(m26! (object (m7 (lex tempest))) (property (m24 (lex threatening))))
(m25! (object (m8 (lex heavens))) (property (m24)))

(m26! m25!)

 CPU time : 0.04

*
;;-------------------------------------------------------------
;;; "days" are credulous
```

```
;;;
;;; Later there is a rule that asserts that "days" being credulous
;;; means that people are credulous.
;;-------------------------------------------------------------

(describe (assert
               object (build lex "days") = days
               property (build lex "credulous") = credulous))

(m29! (object (m27 (lex days))) (property (m28 (lex credulous))))

(m29!)

 CPU time : 0.00

*
;;-------------------------------------------------------------
;;; I will define superordinates for the objects above
;;;
;;; The ontologies used here come from WordNet 2.0:
;;; They were retrieved using the automated
;;; WordNet-to-SNePS interface (wns)
;;-------------------------------------------------------------

;;; tempest
;;;       => windstorm
;;;           => storm, violent storm
;;;               => atmospheric phenomenon
```

```
;;;                    => physical phenomenon
;;;                        => natural phenomenon
;;;                            => phenomenon

; ------------------------------------------------
; ------------------------------------------------
;  hypernymy search results for noun "tempest"
; ------------------------------------------------
; ------------------------------------------------

; --------------- sense 2 ------------------------
; ((literary) a violent wind; "a tempest swept
; over the island")
; ------------------------------------------------

(assert concept #tempest-n-10776284
        word "tempest")

(m30!)

 CPU time : 0.00

*
(assert concept #windstorm-n-10784367
        word "windstorm")

(m31!)
```

```
 CPU time : 0.00

*
(assert subclass *tempest-n-10776284
        superclass *windstorm-n-10784367)

(m32!)

 CPU time : 0.00

*
(assert concept #storm-violent_storm-n-10722776
        word "storm")

(m33!)

 CPU time : 0.00

* (assert concept *storm-violent_storm-n-10722776
        word "violent_storm")

(m34!)

 CPU time : 0.00

*
(assert subclass *windstorm-n-10784367
        superclass *storm-violent_storm-n-10722776)
```

```
(m35!)

 CPU time : 0.00

*
(assert concept #atmospheric_phenomenon-n-10687119
        word "atmospheric_phenomenon")

(m36!)

 CPU time : 0.00

*
(assert subclass *storm-violent_storm-n-10722776
        superclass *atmospheric_phenomenon-n-10687119)

(m37!)

 CPU time : 0.00

*
(assert concept #physical_phenomenon-n-10681095
        word "physical_phenomenon")

(m38!)

 CPU time : 0.00
```

```
*
(assert subclass *atmospheric_phenomenon-n-10687119
        superclass *physical_phenomenon-n-10681095)

(m39!)

 CPU time : 0.00

*
(assert concept #natural_phenomenon-n-10670756
        word "natural_phenomenon")

(m40!)

 CPU time : 0.00

*
(assert subclass *physical_phenomenon-n-10681095
        superclass *natural_phenomenon-n-10670756)

(m41!)

 CPU time : 0.00

*
(assert concept #phenomenon-n-29881
        word "phenomenon")
```

```
(m42!)

 CPU time : 0.01

*
(assert subclass *natural_phenomenon-n-10670756
        superclass *phenomenon-n-29881)

(m43!)

 CPU time : 0.00

*
;;-------------------------------------------------------------
;;; WOMAN
;;;     is a kind of:
;;; human => animate thing => physical object => entity
;;-------------------------------------------------------------

;;; woman, adult female
;;;       => female, female person
;;;           => person, individual, someone, somebody, mortal, human,
soul
;;;               => organism, being
;;;                   => living thing, animate thing
;;;                       => object, physical object
;;;                           => entity
```

```
; ------------------------------------------------
; ------------------------------------------------
;  hypernymy search results for noun "woman"
; ------------------------------------------------
; ------------------------------------------------

; --------------- sense 1 ----------------------
; (an adult female person (as opposed to a man)
; ; "the woman kept house while the man hunted"
; )
; ------------------------------------------------

(assert concept #woman-adult_female-n-10084064
        word "woman")

(m44!)

 CPU time : 0.00

* (assert concept *woman-adult_female-n-10084064
        word "adult_female")

(m45!)

 CPU time : 0.00

*
(assert concept #female-female_person-n-9013489
```

```
        word "female")


(m46!)


 CPU time : 0.00


* (assert concept *female-female_person-n-9013489
        word "female_person")


(m47!)


 CPU time : 0.00


*
(assert subclass *woman-adult_female-n-10084064
        superclass *female-female_person-n-9013489)


(m48!)


 CPU time : 0.00


*
(assert concept #person-individual-someone-somebody-mortal-human-soul-
n-6026
        word "person")


(m49!)


 CPU time : 0.00
```

```
* (assert concept *person-individual-someone-somebody-mortal-human-
soul-n-6026

        word "individual")


(m50!)


 CPU time : 0.00


* (assert concept *person-individual-someone-somebody-mortal-human-
soul-n-6026

        word "someone")


(m51!)


 CPU time : 0.00


* (assert concept *person-individual-someone-somebody-mortal-human-
soul-n-6026

        word "somebody")


(m52!)


 CPU time : 0.00


* (assert concept *person-individual-someone-somebody-mortal-human-
soul-n-6026

        word "mortal")


(m53!)
```

```
 CPU time : 0.00


* (assert concept *person-individual-someone-somebody-mortal-human-
soul-n-6026

        word "human")


(m54!)


 CPU time : 0.00


* (assert concept *person-individual-someone-somebody-mortal-human-
soul-n-6026

        word "soul")


(m55!)


 CPU time : 0.00


*

(assert subclass *female-female_person-n-9013489

        superclass *person-individual-someone-somebody-mortal-human-
soul-n-6026)


(m56!)


 CPU time : 0.00


*

(assert concept #organism-being-n-3226
```

```
        word "organism")


(m57!)


 CPU time : 0.00


* (assert concept *organism-being-n-3226

        word "being")


(m58!)


 CPU time : 0.00


*

(assert subclass *person-individual-someone-somebody-mortal-human-soul-
n-6026

        superclass *organism-being-n-3226)


(m59!)


 CPU time : 0.00


*

(assert concept #living_thing-animate_thing-n-3009

        word "living_thing")


(m60!)


 CPU time : 0.00
```

```
* (assert concept *living_thing-animate_thing-n-3009
        word "animate_thing")

(m61!)

 CPU time : 0.00

*
(assert subclass *organism-being-n-3226
        superclass *living_thing-animate_thing-n-3009)

(m62!)

 CPU time : 0.00

*
(assert concept #object-physical_object-n-16236
        word "object")

(m63!)

 CPU time : 0.00

* (assert concept *object-physical_object-n-16236
        word "physical_object")

(m64!)
```

```
 CPU time : 0.00

*
(assert subclass *living_thing-animate_thing-n-3009
        superclass *object-physical_object-n-16236)

(m65!)

 CPU time : 0.00

*
(assert concept #entity-n-1740
        word "entity")

(m66!)

 CPU time : 0.00

*
(assert subclass *object-physical_object-n-16236
        superclass *entity-n-1740)

(m67!)

 CPU time : 0.01

*
```

```
;;-------------------------------------------------------

;;; DAYS

;;;     is a kind of:

;;; days => time period => fundamental quantity => amount =>
abstraction

;;-------------------------------------------------------


;;; days, years

;;;       => life

;;;           => time period, period of time, period

;;;               => fundamental quantity, fundamental measure

;;;                   => measure, quantity, amount

;;;                       => abstraction


; ------------------------------------------------

; ------------------------------------------------

;  hypernymy search results for noun "days"

; ------------------------------------------------

; ------------------------------------------------


; --------------- sense 1 ----------------------

; (the time during which someone's life continu

; es; "the monarch's last days"; "in his final

; years")

; ------------------------------------------------


(assert concept #days-years-n-14283933

        word "days")
```

```
(m68!)

 CPU time : 0.00

* (assert concept *days-years-n-14283933
        word "years")

(m69!)

 CPU time : 0.00

*
(assert concept #life-n-14283784
        word "life")

(m70!)

 CPU time : 0.00

*
(assert subclass *days-years-n-14283933
        superclass *life-n-14283784)

(m71!)

 CPU time : 0.00

*
```

```
(assert concept #time_period-period_of_time-period-n-14257468
        word "time_period")

(m72!)

 CPU time : 0.00

* (assert concept *time_period-period_of_time-period-n-14257468
        word "period_of_time")

(m73!)

 CPU time : 0.00

* (assert concept *time_period-period_of_time-period-n-14257468
        word "period")

(m74!)

 CPU time : 0.00

*
(assert subclass *life-n-14283784
        superclass *time_period-period_of_time-period-n-14257468)

(m75!)

 CPU time : 0.00
```

```
*
(assert concept #fundamental_quantity-fundamental_measure-n-12810936
        word "fundamental_quantity")

(m76!)

 CPU time : 0.00

* (assert concept *fundamental_quantity-fundamental_measure-n-12810936
        word "fundamental_measure")

(m77!)

 CPU time : 0.00

*
(assert subclass *time_period-period_of_time-period-n-14257468
        superclass *fundamental_quantity-fundamental_measure-n-
12810936)

(m78!)

 CPU time : 0.00

*
(assert concept #measure-quantity-amount-n-29305
        word "measure")
```

```
(m79!)

 CPU time : 0.03

* (assert concept *measure-quantity-amount-n-29305
        word "quantity")

(m80!)

 CPU time : 0.00

* (assert concept *measure-quantity-amount-n-29305
        word "amount")

(m81!)

 CPU time : 0.01

*
(assert subclass *fundamental_quantity-fundamental_measure-n-12810936
        superclass *measure-quantity-amount-n-29305)

(m82!)

 CPU time : 0.00

*
(assert concept #abstraction-n-20486
```

```
        word "abstraction")

(m83!)

 CPU time : 0.00

*
(assert subclass *measure-quantity-amount-n-29305
        superclass *abstraction-n-20486)

(m84!)

 CPU time : 0.00

*
;;-------------------------------------------------------------
;;; NOTHING FAVORABLE
;;; e.g unfavorable
;;;     is a kind of:
;;; quality => attribute => abstraction
;;-------------------------------------------------------------

(describe (assert
                subclass *nothingFavorable
                superclass (build lex quality) = quality))

(m86! (subclass (m21 (lex nothing favorable)))
 (superclass (m85 (lex quality))))
```

```
(m86!)

 CPU time : 0.00

*
;;; quality
;;;       => attribute
;;;           => abstraction

; ------------------------------------------------
; ------------------------------------------------
;  hypernymy search results for noun "quality"
; ------------------------------------------------
; ------------------------------------------------

; --------------- sense 1 ------------------------
; (an essential and distinguishing attribute of
;  something or someone; "the quality of mercy
; is not strained"--Shakespeare)
; ------------------------------------------------

(assert concept #quality-n-4521520
        word "quality")

(m87!)

 CPU time : 0.01
```

```
*
(assert concept #attribute-n-27563
        word "attribute")

(m88!)

 CPU time : 0.00

*
(assert subclass *quality-n-4521520
        superclass *attribute-n-27563)

(m89!)

 CPU time : 0.00

*
(assert concept #abstraction-n-20486
        word "abstraction")

(m90!)

 CPU time : 0.00

*
(assert subclass *attribute-n-27563
        superclass *abstraction-n-20486)
```

```
(m91!)

 CPU time : 0.00

*
;;-------------------------------------------------------------
;;; CREDULOUS
;;;     is a measure of(?):
;;; credibility => quality  => attribute => abstraction
;;;
;;-------------------------------------------------------------

;;; Question for future work:
;;; How can this best be represented?

()



 CPU time : 0.00

*
;;; credibility, credibleness, believability
;;;       => quality
;;;           => attribute
;;;               => abstraction

; ------------------------------------------------
```

```
; ---------------------------------------------
;  hypernymy search results for noun "credibility"
; ---------------------------------------------
; ---------------------------------------------

; --------------- sense 1 ----------------------
; (the quality of being believable or trustwort
; hy)
; ---------------------------------------------

(assert concept #credibility-credibleness-believability-n-4565438
        word "credibility")

(m92!)

 CPU time : 0.00

* (assert concept *credibility-credibleness-believability-n-4565438
        word "credibleness")

(m93!)

 CPU time : 0.00

* (assert concept *credibility-credibleness-believability-n-4565438
        word "believability")

(m94!)
```

```
 CPU time : 0.00

*
(assert concept #quality-n-4521520
        word "quality")

(m95!)

 CPU time : 0.00

*
(assert subclass *credibility-credibleness-believability-n-4565438
        superclass *quality-n-4521520)

(m96!)

 CPU time : 0.00

*
(assert concept #attribute-n-27563
        word "attribute")

(m97!)

 CPU time : 0.00

*
```

```
(assert subclass *quality-n-4521520
        superclass *attribute-n-27563)

(m98!)

 CPU time : 0.00

*
(assert concept #abstraction-n-20486
        word "abstraction")

(m99!)

 CPU time : 0.00

*
(assert subclass *attribute-n-27563
        superclass *abstraction-n-20486)

(m100!)

 CPU time : 0.00

*
;;-------------------------------------------------------------
;;; INTERPRET
;;;     is one way to:
;;; explicate  => inform  => communicate  => interact  => act
```

```
;;-------------------------------------------------------

;;; interpret
;;;       => explain, explicate
;;;           => inform
;;;               => communicate, intercommunicate
;;;                   => interact
;;;                       => act, move

; ------------------------------------------------
; ------------------------------------------------
;  hypernymy search results for verb "interpret"
; ------------------------------------------------
; ------------------------------------------------

; --------------- sense 2 ----------------------
; (give an interpretation or explanation to)
; ------------------------------------------------

(assert concept #interpret-v-907397
        word "interpret")

(m101!)

 CPU time : 0.01

*
(assert concept #explain-explicate-v-908384
```

```
        word "explain")

(m102!)

 CPU time : 0.00

* (assert concept *explain-explicate-v-908384
        word "explicate")

(m103!)

 CPU time : 0.00

*
(assert subclass *interpret-v-907397
        superclass *explain-explicate-v-908384)

(m104!)

 CPU time : 0.00

*
(assert concept #inform-v-804489
        word "inform")

(m105!)

 CPU time : 0.00
```

```
*
(assert subclass *explain-explicate-v-908384
        superclass *inform-v-804489)

(m106!)

 CPU time : 0.00

*
(assert concept #communicate-intercommunicate-v-716346
        word "communicate")

(m107!)

 CPU time : 0.00

* (assert concept *communicate-intercommunicate-v-716346
        word "intercommunicate")

(m108!)

 CPU time : 0.00

*
(assert subclass *inform-v-804489
        superclass *communicate-intercommunicate-v-716346)
```

```
(m109!)

 CPU time : 0.00

*
(assert concept #interact-v-2305904
        word "interact")

(m110!)

 CPU time : 0.00

*
(assert subclass *communicate-intercommunicate-v-716346
        superclass *interact-v-2305904)

(m111!)

 CPU time : 0.01

*
(assert concept #act-move-v-2296591
        word "act")

(m112!)

 CPU time : 0.00
```

```
* (assert concept *act-move-v-2296591
        word "move")

(m113!)

 CPU time : 0.00

*
(assert subclass *interact-v-2305904
        superclass *act-move-v-2296591)

(m114!)

 CPU time : 0.00

*
;;-------------------------------------------------------------
;;; LISTEN
;;;     is one way to:
;;; perceive
;;; concentrate => think
;;-------------------------------------------------------------

;;; listen
;;;       => perceive, comprehend

;;; listen, hear, take heed
;;;       => concentrate, focus, center, centre, pore, rivet
```

```
;;;           => think, cogitate, cerebrate

; ------------------------------------------------
; ------------------------------------------------
;  hypernymy search results for verb "listen"
; ------------------------------------------------
; ------------------------------------------------

; --------------- sense 1 ----------------------
; (hear with intention; "Listen to the sound of
;  this cello")
; ------------------------------------------------

(assert concept #listen-v-2107615
        word "listen")

(m115!)

 CPU time : 0.00

*
(assert concept #perceive-comprehend-v-2045668
        word "perceive")

(m116!)

 CPU time : 0.01
```

```
* (assert concept *perceive-comprehend-v-2045668
        word "comprehend")

(m117!)

 CPU time : 0.00

*
(assert subclass *listen-v-2107615
        superclass *perceive-comprehend-v-2045668)

(m118!)

 CPU time : 0.00

*
; --------------- sense 2 -----------------------
; (listen and pay attention; "Listen to your fa
; ther"; "We must hear the expert before we mak
; e a decision")
; -----------------------------------------------

(assert concept #listen-hear-take_heed-v-2108709
        word "listen")

(m119!)

 CPU time : 0.00
```

```
* (assert concept *listen-hear-take_heed-v-2108709
        word "hear")

(m120!)

 CPU time : 0.00

* (assert concept *listen-hear-take_heed-v-2108709
        word "take_heed")

(m121!)

 CPU time : 0.01

*
(assert concept #concentrate-focus-center-centre-pore-rivet-v-698305
        word "concentrate")

(m122!)

 CPU time : 0.00

* (assert concept *concentrate-focus-center-centre-pore-rivet-v-698305
        word "focus")

(m123!)
```

```
 CPU time : 0.00

* (assert concept *concentrate-focus-center-centre-pore-rivet-v-698305
        word "center")

(m124!)

 CPU time : 0.00

* (assert concept *concentrate-focus-center-centre-pore-rivet-v-698305
        word "centre")

(m125!)

 CPU time : 0.00

* (assert concept *concentrate-focus-center-centre-pore-rivet-v-698305
        word "pore")

(m126!)

 CPU time : 0.01

* (assert concept *concentrate-focus-center-centre-pore-rivet-v-698305
        word "rivet")

(m127!)
```

```
 CPU time : 0.00


*
(assert subclass *listen-hear-take_heed-v-2108709
        superclass *concentrate-focus-center-centre-pore-rivet-v-
698305)


(m128!)


 CPU time : 0.00


*
(assert concept #think-cogitate-cerebrate-v-608615
        word "think")


(m129!)


 CPU time : 0.00


* (assert concept *think-cogitate-cerebrate-v-608615
        word "cogitate")


(m130!)


 CPU time : 0.00


* (assert concept *think-cogitate-cerebrate-v-608615
        word "cerebrate")
```

```
(m131!)

 CPU time : 0.00

*
(assert subclass *concentrate-focus-center-centre-pore-rivet-v-698305
        superclass *think-cogitate-cerebrate-v-608615)

(m132!)

 CPU time : 0.00

*
;;-------------------------------------------------------------
;;; REDOUBLE
;;;     is one way to:
;;; escalate => increase => change
;;-------------------------------------------------------------

;;; listen, hear, take heed
;;;       => concentrate, focus, center, centre, pore, rivet
;;;           => think, cogitate, cerebrate

; ------------------------------------------------
; ------------------------------------------------
;  hypernymy search results for verb "redouble"
; ------------------------------------------------
; ------------------------------------------------
```

```
; --------------- sense 1 ----------------------
; (double in magnitude, extent, or intensity; "
; The enemy redoubled their screaming on the ra
; dio")
; ------------------------------------------------

(assert concept #redouble-v-281537
        word "redouble")

(m133!)

 CPU time : 0.00

*
(assert concept #escalate-intensify-step_up-v-281258
        word "escalate")

(m134!)

 CPU time : 0.00

* (assert concept *escalate-intensify-step_up-v-281258
        word "intensify")

(m135!)

 CPU time : 0.00
```

```
* (assert concept *escalate-intensify-step_up-v-281258
        word "step_up")

(m136!)

 CPU time : 0.00

*
(assert subclass *redouble-v-281537
        superclass *escalate-intensify-step_up-v-281258)

(m137!)

 CPU time : 0.00

*
(assert concept #increase-v-147655
        word "increase")

(m138!)

 CPU time : 0.00

*
(assert subclass *escalate-intensify-step_up-v-281258
        superclass *increase-v-147655)
```

```
(m139!)

 CPU time : 0.00

*
(assert concept #change-alter-modify-v-121430
        word "change")

(m140!)

 CPU time : 0.00

* (assert concept *change-alter-modify-v-121430
        word "alter")

(m141!)

 CPU time : 0.00

* (assert concept *change-alter-modify-v-121430
        word "modify")

(m142!)

 CPU time : 0.00

*
(assert subclass *increase-v-147655
```

```
        superclass *change-alter-modify-v-121430)

(m143!)

 CPU time : 0.00

*
;;;-------------------------------------------------------------

;;; end WordNet rules
;;; now load rules to convert wordnet case-frames
;;; to "standard" SNePS/CVA case-frames

 (intext "conv-rules")
File conv-rules is now the source of input.



 CPU time : 0.00

*  concept is already defined.
 word is already defined.

(concept word)

 CPU time : 0.01

*
```

```
(verb-concept entails)

 CPU time : 0.00

*  cause is already defined.
 effect is already defined.

(cause effect)

 CPU time : 0.00

*

(m269! m268! m266! m263! m262! m260! m257! m256! m254! m251! m250!
 m249! m248! m247! m245! m244! m243! m242! m240! m239! m238! m236!
 m235! m233! m230! m229! m227! m224! m221! m218! m215! m212! m211!
 m209! m206! m205! m203! m200! m197! m196! m194! m191! m189! m186!
 m183! m180! m179! m178! m177! m176! m175! m173! m172! m171! m170!
 m169! m168! m167! m166! m165! m163! m162! m161! m159! m158! m156!
 m153! m150! m147! m144! m142! m141! m140! m138! m136! m135! m134!
 m133! m131! m130! m129! m127! m126! m125! m124! m123! m122! m121!
 m120! m119! m117! m116! m115! m113! m112! m110! m108! m107! m105!
 m103! m102! m101! m99! m97! m95! m94! m93! m92! m90! m88! m87! m83!
 m81! m80! m79! m77! m76! m74! m73! m72! m70! m69! m68! m66! m64! m63!
 m61! m60! m58! m57! m55! m54! m53! m52! m51! m50! m49! m47! m46! m45!
 m44! m42! m40! m38! m36! m34! m33! m31! m30!)

 CPU time : 25.11
```

```
*

(m270! m142! m141! m140! m138! m136! m135! m134! m133! m131! m130!
 m129! m127! m126! m125! m124! m123! m122! m121! m120! m119! m117!
 m116! m115! m113! m112! m110! m108! m107! m105! m103! m102! m101! m99!
 m97! m95! m94! m93! m92! m90! m88! m87! m83! m81! m80! m79! m77! m76!
 m74! m73! m72! m70! m69! m68! m66! m64! m63! m61! m60! m58! m57! m55!
 m54! m53! m52! m51! m50! m49! m47! m46! m45! m44! m42! m40! m38! m36!
 m34! m33! m31! m30!)

 CPU time : 1.16

*

(m669! m668! m667! m666! m665! m664! m663! m662! m661! m660! m659!
 m658! m657! m656! m655! m654! m653! m652! m651! m649! m648! m647!
 m646! m645! m644! m643! m642! m641! m640! m639! m638! m637! m636!
 m635! m634! m633! m632! m631! m630! m629! m628! m627! m626! m625!
 m624! m623! m622! m621! m620! m619! m618! m617! m616! m615! m614!
 m613! m612! m611! m610! m609! m608! m607! m606! m605! m604! m603!
 m602! m601! m600! m599! m598! m597! m596! m595! m594! m593! m592!
 m591! m590! m589! m588! m587! m586! m585! m584! m583! m582! m581!
 m580! m578! m577! m576! m575! m574! m573! m572! m571! m570! m569!
 m568! m566! m565! m563! m562! m561! m560! m559! m558! m557! m556!
 m555! m554! m553! m552! m550! m549! m548! m547! m546! m545! m544!
 m543! m542! m541! m540! m538! m537! m536! m535! m534! m533! m532!
 m531! m530! m529! m528! m527! m526! m525! m524! m523! m522! m521!
```

m520! m519! m518! m517! m516! m515! m514! m513! m512! m511! m510!

m509! m508! m507! m506! m505! m504! m503! m502! m501! m500! m499!

m498! m497! m496! m495! m494! m493! m492! m490! m489! m488! m487!

m486! m485! m484! m483! m482! m481! m480! m479! m478! m477! m476!

m475! m474! m472! m471! m470! m469! m468! m467! m466! m465! m464!

m463! m462! m461! m460! m459! m458! m457! m456! m455! m454! m453!

m452! m451! m450! m449! m448! m447! m446! m445! m444! m443! m442!

m441! m440! m439! m438! m437! m436! m435! m434! m433! m432! m431!

m430! m429! m428! m427! m426! m425! m424! m423! m422! m421! m420!

m419! m418! m417! m416! m415! m414! m413! m412! m411! m410! m409!

m408! m407! m406! m405! m404! m403! m402! m401! m400! m399! m398!

m397! m396! m395! m394! m393! m392! m391! m390! m389! m388! m387!

m386! m385! m384! m383! m382! m381! m380! m379! m378! m377! m376!

m375! m374! m373! m372! m371! m370! m369! m368! m367! m366! m365!

m364! m363! m362! m361! m360! m359! m358! m357! m356! m354! m353!

m352! m351! m350! m349! m347! m346! m345! m344! m343! m341! m340!

m339! m338! m336! m335! m334! m333! m332! m331! m329! m328! m327!

m326! m325! m324! m323! m322! m321! m320! m319! m318! m317! m316!

m315! m314! m313! m312! m311! m310! m309! m308! m307! m306! m305!

m304! m303! m302! m301! m300! m299! m298! m297! m296! m295! m294!

m293! m292! m291! m290! m289! m288! m287! m286! m285! m284! m283!

m282! m281! m280! m279! m278! m277! m276! m275! m274! m273! m272!

m271! m143! m142! m141! m140! m139! m138! m137! m136! m135! m134!

m133! m132! m131! m130! m129! m128! m127! m126! m125! m124! m123!

m122! m121! m120! m119! m118! m117! m116! m115! m114! m113! m112!

m111! m110! m109! m108! m107! m106! m105! m104! m103! m102! m101!

m100! m99! m98! m97! m96! m95! m94! m93! m92! m91! m90! m89! m88! m87!

m86! m84! m83! m82! m81! m80! m79! m78! m77! m76! m75! m74! m73! m72!

```
 m71! m70! m69! m68! m67! m66! m65! m64! m63! m62! m61! m60! m59! m58!
 m57! m56! m55! m54! m53! m52! m51! m50! m49! m48! m47! m46! m45! m44!
 m43! m42! m41! m40! m39! m38! m37! m36! m35! m34! m33! m32! m31! m30!)

 CPU time : 5.47

*

(m670! m142! m141! m140! m138! m136! m135! m134! m133! m131! m130!
 m129! m127! m126! m125! m124! m123! m122! m121! m120! m119! m117!
 m116! m115! m113! m112! m110! m108! m107! m105! m103! m102! m101! m99!
 m97! m95! m94! m93! m92! m90! m88! m87! m83! m81! m80! m79! m77! m76!
 m74! m73! m72! m70! m69! m68! m66! m64! m63! m61! m60! m58! m57! m55!
 m54! m53! m52! m51! m50! m49! m47! m46! m45! m44! m42! m40! m38! m36!
 m34! m33! m31! m30!)

 CPU time : 1.50

*

End of file conv-rules


 CPU time : 33.32

*
;;;-------------------------------------------------------------
```

```
;;-------------------------------------------------------
;;; Representing parts of the passage:
;;-------------------------------------------------------

;;; RULES INFERED FROM THE PASSAGE
;;-------------------------------------------------------
;;;
;;;     "People interpret a change in the heavens."
;;;
;;; This can be rephrased as "for all x, if X is a person
;;; then X interprets a change [in the heavens].
;;;
;;; This gives us a representation for:
;;;
;;;     If people do X, then the woman does X
;;;
;;-------------------------------------------------------

;;; "change is represented as an instance of "change" (i.e. a member of
;;; the class "change"); this instance  would be the
;;; change relating to the "heavens". This would  separate it from
;;; the sense of "change" used previously, and a superclass of the
;;; heirarchy "redouble => escalate => increase => change".
;;;
;;; The only issue that could arise is whether we should have
;;; multiple instances of change for each rule that refers to
;;; some sort of "change", and then have a rule that allows the
;;; relations coming off one instance of change to refer to the
```

```
;;; others as well.
;;-------------------------------------------------------


;(show
(describe (add forall $person
        ant (build member *person class *human)
        cq (build
                agent *person
                act (build
                        action (build lex "interpret")
                        object (build lex "instance of change")
                                = instChange))
))

(m676! (act (m675 (action (m564 (lex interpret)))))
 (agent (m14 (lex the woman))))
(m674!
 (act (m672 (action (m564)) (object (m671 (lex instance of change)))))
 (agent (m14)))
(m673! (forall v39) (ant (p49 (class (m19 (lex human))) (member v39)))
 (cq (p50 (act (m672)) (agent v39))))
(m20! (class (m19)) (member (m14)))

(m676! m674! m673! m20!)

 CPU time : 0.69
```

```
* ;)

;;-------------------------------------------------------------
;; define what an instance of "change" is
;;-------------------------------------------------------------
(describe (add member *instChange class *change ))

(m677! (member (m671 (lex instance of change))))

(m677!)

 CPU time : 0.07

*
;;-------------------------------------------------------------
;;; If the days are credulous, then all people are credulous
;;;
;;-------------------------------------------------------------

(describe (add forall *person
        &ant (  (build member *person class *human)
                (build object *days property *credulous)
              )
        cq (build object *person property *credulous)
))

(m679! (object (m14 (lex the woman))) (property (m28 (lex credulous))))
(m678! (forall v39)
```

```
 (&ant (p49 (class (m19 (lex human))) (member v39))
  (m29! (object (m27 (lex days))) (property (m28))))
 (cq (p53 (object v39) (property (m28)))))

(m679! m678!)

 CPU time : 0.16

*
;;-------------------------------------------------------------
;;; If a person is credulous, they are
;;;     "disposed to believe on little evidence"
;;;
;;; This just summarizes the meaning of credulous into
;;; one node that can be accessed from the verb algorithm.
;;-------------------------------------------------------------

(describe (add forall *person
        &ant (  (build member *person class *human)
                (build object *person property *credulous)
              )
        cq (build
                object *person
                property (build lex "disposed to believe on little
evidence")
            )
))

(m682! (object (m14 (lex the woman)))
```

```
 (property (m680 (lex disposed to believe on little evidence))))
(m681! (forall v39)
 (&ant (p53 (object v39) (property (m28 (lex credulous))))
  (p49 (class (m19 (lex human))) (member v39)))
 (cq (p54 (object v39) (property (m680)))))
(m29! (object (m27 (lex days))) (property (m28)))

(m682! m681! m29!)

 CPU time : 0.64

*
;;-------------------------------------------------------------
;;; If X does the act of "redouble" then X possesses
;;; and instance of "change".
;;;
;;; Note: In the background knowledge represented above,
;;; "change" is the highest superordinate category of "redouble"
;;-------------------------------------------------------------

;(show
(describe (add forall $h
                ant (build
                        agent *h
                        act (build action *redouble)
                     )

                ;cq (build possessor *h object *change)
```

```
                cq (build possessor *h object *instChange)
))

(m685! (object (m671 (lex instance of change)))
 (possessor (m8 (lex heavens))))
(m684! (object (m671)) (possessor (m7 (lex tempest))))
(m683! (forall v40)
 (ant (p57 (act (m11 (action (m10 (lex redouble))))) (agent v40)))
 (cq (p58 (object (m671)) (possessor v40))))
(m13! (act (m11)) (agent (m8)))
(m12! (act (m11)) (agent (m7)))

(m685! m684! m683! m13! m12!)

 CPU time : 0.77

* ;:format ps)

;;-------------------------------------------------------------
;;;
;;; "The poor young woman could augur nothing favorable
;;;  as she listened to the threatening heavens..."
;;;
;;-------------------------------------------------------------
;;; The main part of the passage consists of two clauses that
;;; are the antecedent - consequent of each other.
;;;
;;;
```

```
;;; ANT: The woman listens to change [in the heavens].
;;; CQ: The woman augurs a nothing favorable.
;;;
;;-------------------------------------------------------------

;;; (possible problem: what is the woman actually listening to?)
;;; the "heavens", the "threatening" heavens, or the "changes" of it?
;;;
;;; I decided to have her listen to the "change" in the heavens.
;;;

;;-------------------------------------------------------------
;;; the first clause:
;;; The woman listens to change [in the heavens].
;;-------------------------------------------------------------

;(show
(describe (add agent *theWoman
                act (build action (build lex "listen")
                           object *instChange)
        ) = Woman_listens_To_Heavens
)

(m690! (similar (m564 (lex interpret)) (m225 (lex listen))))
(m689! (act (m688 (action (m225)))) (agent (m14 (lex the woman))))
(m687!
 (act (m686 (action (m225)) (object (m671 (lex instance of change)))))
 (agent (m14)))
```

```
(m690! m689! m687!)

 CPU time : 0.13

* ;)

;;-------------------------------------------------------------
;;; the second clause:
;;; The woman augurs nothing favorable.
;;-------------------------------------------------------------
;(show
(describe (add agent *theWoman
                  act (build action (build lex "augur") = augur
                                    object *nothingFavorable
                        )
        ) = Woman_augurs_Nothing
)

(m695! (act (m694 (action (m691 (lex augur)))))
 (agent (m14 (lex the woman))))
(m693!
 (act (m692 (action (m691)) (object (m21 (lex nothing favorable)))))
 (agent (m14)))

(m695! m693!)

 CPU time : 0.12
```

```
* ;)

;;-------------------------------------------------------
;;; mark the word "augur" as "unknown"
;;-------------------------------------------------------

(describe (assert object *augur property (build lex "unknown")))

(m697! (object (m691 (lex augur))) (property (m696 (lex unknown))))

(m697!)

 CPU time : 0.00

*
;;-------------------------------------------------------
;;; combine both of the above clauses into cause - effect
;;;
;;; CAUSE: The woman listens to change [in the heavens].
;;; EFFECT: The woman augurs a nothing favorable.
;;-------------------------------------------------------

;(show
(describe (add  cause *Woman_listens_To_Heavens
                effect *Woman_augurs_Nothing))

(p74!
```

```
 (cause
  (p72 (act (p71 (action (m225 (lex listen))) (object v11)))
   (agent v9)))
 (effect
  (m693!
   (act (m692 (action (m691 (lex augur)))
         (object (m21 (lex nothing favorable)))))
   (agent (m14 (lex the woman))))))
(p73! (cause (p72))
 (effect (m695! (act (m694 (action (m691)))) (agent (m14)))))
(p70!
 (cause
  (p68 (act (p67 (action (m564 (lex interpret))) (object v11)))
   (agent v9)))
 (effect (m695!)))
(p69! (cause (p68)) (effect (m693!)))
(m700!
 (cause
  (m687!
   (act (m686 (action (m225))
         (object (m671 (lex instance of change)))))
   (agent (m14))))
 (effect (m695!)))
(m699! (cause (m687!)) (effect (m693!)))
(m698!
 (cause (m690! (similar (m564) (m225)))
  (m689! (act (m688 (action (m225)))) (agent (m14))) (m687!))
 (effect (m695!) (m693!)))
```

```
(p74! p73! p70! p69! m700! m699! m698!)

 CPU time : 0.57

* ;:format ps)

;;-------------------------------------------------------------
;;; INFORMATION FROM PROTOCOLS
;;;
;;; I took protocols on this context from myself before I
;;; knew what "augur" meant.  I will represent some of the
;;; inferences I made here.
;;
;;-------------------------------------------------------------
;;;     if: a person interprets a change,
;;;     and the days are credulous,
;;;     then: the change is interpreted to be a "bad omen"
;;;
;;-------------------------------------------------------------

;(show
(describe (add forall *person
        ant ( (build
                agent *person
                act (build
                        action (build lex "interpret")
                        object *instChange))
```

```
                (build object *days property *credulous) )

        cq  (build      equiv *instChange
                        equiv (build lex "bad omen") = badOmen)
))

(m703! (forall v39)
 (ant
  (p50
   (act (m672 (action (m564 (lex interpret)))
         (object (m671 (lex instance of change)))))
   (agent v39))
  (m29! (object (m27 (lex days))) (property (m28 (lex credulous)))))
 (cq (m702! (equiv (m701 (lex bad omen)) (m671)))))

(m703! m702!)

 CPU time : 0.57

* ;:format ps)

;;-------------------------------------------------------------
;;; If the act of "interpreting change" caused some act Y,
;;; and "change" is equiv to Z, then Z causes Y as well.
;;;
;;;
;;; *this should make "bad omen" a cause of the augur act.
```

```
;;;
;;-------------------------------------------------------

;(show
(describe (add forall (*person $x6 $y6 $z6 $a6 $b6)

        &ant ( (build   cause   (build
                                agent *person
                                act (build
                                        action (build lex "interpret")
                                        object *instChange))
                        effect (build
                                agent *person
                                act (build
                                        action *a6
                                        object *b6)) = aact )

                (build equiv *x6 equiv *z6)
             )

        cq (build cause *z6 effect *aact)
))

(m709! (cause (m701 (lex bad omen)))
 (effect
  (m693!
   (act (m692 (action (m691 (lex augur)))
         (object (m21 (lex nothing favorable)))))
```

```
   (agent (m14 (lex the woman))))))
(m708! (cause (m7 (lex tempest))) (effect (m693!)))
(m707! (cause (m671 (lex instance of change))) (effect (m693!)))
(m706! (cause (m8 (lex heavens))) (effect (m693!)))
(m705!
 (cause
  (m674! (act (m672 (action (m564 (lex interpret))) (object (m671))))
   (agent (m14))))
 (effect (m693!)))
(p95!
 (cause (p94 (act (p67 (action (m564)) (object v11))) (agent (m14))))
 (effect (m693!)))
(p93!
 (cause
  (p92 (act (p71 (action (m225 (lex listen))) (object v11)))
   (agent (m14))))
 (effect (m693!)))
(m704! (forall v45 v44 v43 v42 v41 v39)
 (&ant (p81 (equiv v43 v41))
  (p80 (cause (p50 (act (m672)) (agent v39)))
   (effect (p79 (act (p78 (action v44) (object v45))) (agent v39)))))
 (cq (p82 (cause v43) (effect (p79)))))
(m702! (equiv (m701) (m671)))
(p74! (cause (p72 (act (p71)) (agent v9))) (effect (m693!)))
(p69! (cause (p68 (act (p67)) (agent v9))) (effect (m693!)))
(m9! (equiv (m8) (m7)))

(m709! m708! m707! m706! m705! p95! p93! m704! m702! p74! p69! m9!)
```

```
 CPU time : 2.86

* ;:format ps)

;;-------------------------------------------------------------
;;; A "bad omen" is related to "nothing favorable"
;;;
;;; *This is another thing I came up with when I took
;;; protocols on this passage myself.  I don't know if it actually
;;; adds anything important to this representation.
;;-------------------------------------------------------------

(describe (add object *badOmen property *nothingFavorable))

(m711! (object (m671 (lex instance of change)))
 (property (m21 (lex nothing favorable))))
(m710! (object (m701 (lex bad omen))) (property (m21)))

(m711! m710!)

 CPU time : 0.52

*
;;; End of representations
;;; Ask Cassie what "augur" means:
;;-------------------------------------------------------------
```

```
^(
--> defineVerb 'augur)
SNePS ERROR:  Illegal relation or path: (instrument- act action lex)
Occurred in module find in function checkpath

Do you want to debug it? n
SNePS ERROR:  Illegal relation or path: (with- action lex)
Occurred in module find in function checkpath

Do you want to debug it? n


 * * * Defining the verb augur * * *


 Arguments of the verb:
   (agent object)

 Transitivity:   transitive


 * * * Basic Findings * * *


   possible cause of augur is:
       bad omen,
       heavens,
       instance of change,
```

tempest,

possible actions performed by agent of augur is:

augur,

interpret,

listen,

possible actions performed on object of augur is:

augur,

possible actions performed with instrument of augur is:

augur,

possible action that is the cause of augur is:

listen,

interpret,

possible property of verb is:

unknown,

possible membership of agent is:

human,

possible superordinate of agent is:

entity,

physical_object,

object,

animate_thing,

living_thing,

being,

organism,

possible property of agent is:

disposed to believe on little evidence,

credulous,

poor,

young,

possible superclass of object is:

quality,

possible property of object is:

not favorable,

* * * generalizations from this context * * *

A {human} can augur

Something that is a subclass of {quality}

A {human} can augur

```
         Something with the properties {not favorable}


  A {entity, physical_object, object, animate_thing, living_thing,
being, organism} can augur

         Something that is a subclass of {quality}


  A {entity, physical_object, object, animate_thing, living_thing,
being, organism} can augur

         Something with the properties {not favorable}


  Something with the properties {disposed to believe on little
evidence, credulous, poor, young} can augur

         Something that is a subclass of {quality}


  Something with the properties {disposed to believe on little
evidence, credulous, poor, young} can augur

         Something with the properties {not favorable}

nil




 CPU time : 0.06


*


End of /home/csgrad/ddligach/IndProj/Demo/augur-wn.demo demonstration.
```

Appendix 3

WNS Code

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "wn.h"
#include "wnhelp.h"
#include "license.h"

//some global constants

#define MAX_WORD_LEN 300
#define MAX_WORDS_IN_PASSAGE 200
#define MAX_INPUT_FILE_LEN 300 * 200
#define MAX_BASE_NODE_NAME_LEN 500
#define MAX_POS_NAME_LEN 20

//function prototypes

void SearchForAntonyms(char * searchstr, int pos);
void SearchForHypernyms(char * searchstr, int pos);
void SearchForSynonyms(char * searchstr, int pos);
void SearchForHolonyms(char * searchstr, int pos);
void SearchForMeronyms(char * searchstr, int pos);
void SearchForEntailments(char * searchstr, int pos);
void SearchForCausations(char * searchstr, int pos);

void TraverseAntonymSenses(SynsetPtr synset_ptr);
void TraverseHypernymSenses(SynsetPtr synset_ptr);
void TraverseHolonymSenses(SynsetPtr synset_ptr);
void TraverseMeronymSenses(SynsetPtr synset_ptr);
void TraverseEntailmentSenses(SynsetPtr synset_ptr);
void TraverseCausationSenses(SynsetPtr synset_ptr);

void TraverseAntonyms(SynsetPtr synset_ptr);
void TraverseHypernyms(SynsetPtr synset_ptr);
void TraverseHolonyms(SynsetPtr synset_ptr);
void TraverseMeronyms(SynsetPtr synset_ptr);
void TraverseEntailments(SynsetPtr synset_ptr);
void TraverseCausations(SynsetPtr synset_ptr);

void PrintSynset(SynsetPtr synset_ptr);
char * GetBaseNodeName(SynsetPtr synset_ptr, char * result);
void PrintSynsetStruct(SynsetPtr synset_ptr);
void PrintDefinition(char * definition);
void PrintSearchHeader(char * searchtype, int pos, char * searchstr);
void PrintSenseHeader(int sensenum, char * definition);
int Tokenize(char * passage, char tokenized_passage[][MAX_WORD_LEN]);
int ReadFileIntoString(char * file_name, char * result_str);
void GetPOSName(int pos, char * result);
void ReplaceChar(char * str, char char_to_replace, char replacement_char);

/*****************************************************************************
```

```
 *
 * main()
 *
 ******************************************************************************/

main(int argc, char *argv[])
{
	char tokenized_passage[MAX_WORDS_IN_PASSAGE][MAX_WORD_LEN];
	char file_str[MAX_INPUT_FILE_LEN];
	int loop_index = 0;
	int word_count = 0;

	//initialize wordnet database
	if(wninit())
	{
		printf("error opening WordNet database!\n");
		exit(-1); //couldn't open database
	}

	if(argc < 3)
	{
		//no arguments were passed
		printf("\nWordNet to SNePS interface\n");
		printf("--------------------------\n\n");
		printf("usage:\n\n");
		printf("  wns -file <file name>\n");
		printf("  wns -word <word>\n\n");
		exit(-1); //no command line params passed
	}

	//parse command line
	else if(strcmp(argv[1], "-file") == 0)
	{
		//read input file into string and tokenize it
		ReadFileIntoString(argv[2], file_str);
		word_count = Tokenize(file_str, tokenized_passage);
	}
	else if(strcmp(argv[1], "-word") == 0)
	{
		//store this word in the word table (tokenized_passage)
		word_count = 1;
		strcpy(tokenized_passage[0], argv[2]);
	}
	else
	{
		printf("incorrect command line option!\n");
		exit(-1);
	}

	//loop through the tokens
	for(loop_index = 0; loop_index < word_count; loop_index++)
	{
		//perform noun searches

		SearchForAntonyms(tokenized_passage[loop_index], NOUN);
		SearchForHypernyms(tokenized_passage[loop_index], NOUN);
		SearchForSynonyms(tokenized_passage[loop_index], NOUN);
		SearchForHolonyms(tokenized_passage[loop_index], NOUN);
		SearchForMeronyms(tokenized_passage[loop_index], NOUN);

		//perform verb searches
```

```
                SearchForAntonyms(tokenized_passage[loop_index], VERB);
                SearchForHypernyms(tokenized_passage[loop_index], VERB);
                SearchForSynonyms(tokenized_passage[loop_index], VERB);
                SearchForEntailments(tokenized_passage[loop_index], VERB);
                SearchForCausations(tokenized_passage[loop_index], VERB);
        }

}


/*****************************************************************************
 *
 * The following set of functions will perform various types of wordnet searches
 *
 *****************************************************************************/


/*****************************************************************************
 *
 * SearchForAntonyms()
 *
 * This function will call findtheinfo_ds to search for antonyms of the searchstr
 *
 *****************************************************************************/

void SearchForAntonyms(char * searchstr, int pos)
{
        SynsetPtr search_result_ptr = NULL; //result of findtheinfo(...)

        //print textual header for this search
        PrintSearchHeader("antonymy", pos, searchstr);

        search_result_ptr = findtheinfo_ds(searchstr, pos, ANTPTR, ALLSENSES);
        if(!search_result_ptr)
        {
                //if search did not return anything
                //try to morph the search string
                search_result_ptr = findtheinfo_ds(morphstr(searchstr, pos), pos,
                                        ANTPTR, ALLSENSES);
        }
        if(search_result_ptr)
        {
                //there are search results, output them
                TraverseAntonymSenses(search_result_ptr);
        }

        //free the space allocated by findtheinfo_ds()
        free_syns(search_result_ptr);
}


/*****************************************************************************
 *
 * SearchForHypernyms()
 *
 * This function will cal findtheinfo_ds to search for all hypernyms
 * of the searchstr.
 *
 *****************************************************************************/

void SearchForHypernyms(char * searchstr, int pos)
{
```

```
	SynsetPtr search_result_ptr = NULL; //result of findtheinfo(...)

	PrintSearchHeader("hypernymy", pos, searchstr);

	search_result_ptr = findtheinfo_ds(searchstr, pos, -HYPERPTR, ALLSENSES);
	if(!search_result_ptr)
	{
		//if search did not return anything
		//try to morph the search string
		search_result_ptr = findtheinfo_ds(morphstr(searchstr, pos), pos,
					-HYPERPTR, ALLSENSES);
	}
	if(search_result_ptr)
	{
		//the search produced some results, output them
		TraverseHypernymSenses(search_result_ptr);
	}

	//free the space allocated by findtheinfo_ds()
	free_syns(search_result_ptr);
}


/*****************************************************************************
 *
 * SearchForSynonyms()
 *
 * This function will call findtheinfo_ds() to search for synonyms of the
 * searchstr and its immediate hypernyms
 *
 *****************************************************************************/

void SearchForSynonyms(char * searchstr, int pos)
{
	SynsetPtr search_result_ptr = NULL; //result of findtheinfo(...)

	PrintSearchHeader("synonymy", pos, searchstr);

	search_result_ptr = findtheinfo_ds(searchstr, pos, HYPERPTR, ALLSENSES);
	if(!search_result_ptr)
	{
		//if search did not return anything
		//try to morph the search string
		search_result_ptr = findtheinfo_ds(morphstr(searchstr, pos), pos,
					HYPERPTR, ALLSENSES);
	}
	if(search_result_ptr)
	{
		//the search produced some results, output them
		TraverseHypernymSenses(search_result_ptr);
	}

	//free the space allocated by findtheinfo_ds()
	free_syns(search_result_ptr);
}


/*****************************************************************************
 *
 * SearchForHolonyms()
 *
 * This function will call findtheinfo_ds() three times to search for
```

```
 * each type of holonyms of the searchstr.
 *
 ****************************************************************************/

void SearchForHolonyms(char * searchstr, int pos)
{
	SynsetPtr search_result_ptr = NULL; //result of findtheinfo(...)
	int holonym_type; //"member of", "substance of", or "part of"

	PrintSearchHeader("holonymy", pos, searchstr);

	for(holonym_type = ISMEMBERPTR; holonym_type <= ISPARTPTR; holonym_type++)
	{
		search_result_ptr = findtheinfo_ds(searchstr,
						pos,
						holonym_type,
						ALLSENSES);
		if(!search_result_ptr)
		{
			//if search did not return anything
			//try to morph the search string
			search_result_ptr = findtheinfo_ds(morphstr(searchstr, pos),
							pos,
							holonym_type,
							ALLSENSES);
		}

		if(search_result_ptr)
		{
			//the search produced some results, output them
			TraverseHolonymSenses(search_result_ptr);
		}
	}

	//free the space allocated by findtheinfo_ds()
	free_syns(search_result_ptr);
}


/****************************************************************************
 *
 * SearchForMeronyms()
 *
 * This function will call findtheinfo_ds() three times to search for
 * each type of meronyms of the searchstr.
 *
 ****************************************************************************/

void SearchForMeronyms(char * searchstr, int pos)
{
	SynsetPtr search_result_ptr = NULL; //result of findtheinfo(...)
	int meronym_type; //member, substance, or type meronym

	PrintSearchHeader("meronymy", pos, searchstr);

	for(meronym_type = HASMEMBERPTR; meronym_type <= HASPARTPTR; meronym_type++)
	{
		search_result_ptr = findtheinfo_ds(searchstr,
						pos,
						meronym_type,
						ALLSENSES);
		if(!search_result_ptr)
```

```
		{
			//if search did not return anything
			//try to morph the search string
			search_result_ptr = findtheinfo_ds(morphstr(searchstr, pos),
								pos,
								meronym_type,

ALLSENSES);
		}

		if(search_result_ptr)
		{
			//the search produced some results, output them
			TraverseMeronymSenses(search_result_ptr);
		}
	}

	//free the space allocated by findtheinfo_ds()
	free_syns(search_result_ptr);
}


/*****************************************************************************
 *
 * SearchForEntailments()
 *
 * This function will cal findtheinfo_ds to search for entailment of searchstr.
 *
 *****************************************************************************/

void SearchForEntailments(char * searchstr, int pos)
{
	SynsetPtr search_result_ptr = NULL; //result of findtheinfo(...)

	PrintSearchHeader("entailment", pos, searchstr);

	search_result_ptr = findtheinfo_ds(searchstr, pos, ENTAILPTR, ALLSENSES);
	if(!search_result_ptr)
	{
		//if search did not return anything
		//try to morph the search string
		search_result_ptr = findtheinfo_ds(morphstr(searchstr, pos), pos,
					ENTAILPTR, ALLSENSES);
	}
	if(search_result_ptr)
	{
		//the search produced some results, output them
		TraverseEntailmentSenses(search_result_ptr);
	}

	//free the space allocated by findtheinfo_ds()
	free_syns(search_result_ptr);
}


/*****************************************************************************
 *
 * SearchForCausations()
 *
 * This function will cal findtheinfo_ds to search for causation of searchstr.
 *
 *****************************************************************************/
```

```
void SearchForCausations(char * searchstr, int pos)
{
	SynsetPtr search_result_ptr = NULL; //result of findtheinfo(...)

	PrintSearchHeader("causation", pos, searchstr);

	search_result_ptr = findtheinfo_ds(searchstr, pos, CAUSETO, ALLSENSES);
	if(!search_result_ptr)
	{
		//if search did not return anything
		//try to morph the search string
		search_result_ptr = findtheinfo_ds(morphstr(searchstr, pos), pos,
					CAUSETO, ALLSENSES);
	}
	if(search_result_ptr)
	{
		//the search produced some results, output them
		TraverseCausationSenses(search_result_ptr);
	}

	//free the space allocated by findtheinfo_ds()
	free_syns(search_result_ptr);
}


/*****************************************************************************
 *
 * The following set of function will loop through all senses of the searchstr
 *
 *****************************************************************************/


/*****************************************************************************
 *
 * TraverseAntonymSenses()
 *
 * This function moves through all senses of the search string
 * and calls TraverseAntonyms() for each senses
 *
 *****************************************************************************/

void TraverseAntonymSenses(SynsetPtr synset_ptr)
{
	int sense_counter = 1;
	//save pointer passed to this function
	SynsetPtr cur_sense_ptr = synset_ptr;

   //loop through various senses of the searchstr
   while(cur_sense_ptr)
   {
		//if there are antonyms for this sense, output them
		if(cur_sense_ptr -> ptrlist)
		{
			PrintSenseHeader(sense_counter++, cur_sense_ptr -> defn);

			//traverse and print antonyms for this sense
			TraverseAntonyms(cur_sense_ptr);
		}

		//move to the next sense
      cur_sense_ptr = cur_sense_ptr -> nextss;
```

```
   }
}


/*******************************************************************************
 *
 * TraverseHypernymSenses()
 *
 * This function loops through all senses of the search string
 * and calls TraverseHypernyms() for each sense to print the hierarchy
 *
 *******************************************************************************/

void TraverseHypernymSenses(SynsetPtr synset_ptr)
{
        int sense_counter = 1;
        //save pointer passed to this function
        SynsetPtr cur_sense_ptr = synset_ptr;

   //loop through various senses of the searchstr
   while(cur_sense_ptr)
   {
                PrintSenseHeader(sense_counter++, cur_sense_ptr -> defn);

                //traverse and print hypernym hierarchy for this sense
                TraverseHypernyms(cur_sense_ptr);

                //move to the next sense
      cur_sense_ptr = cur_sense_ptr -> nextss;
   }
}


/*******************************************************************************
 *
 * TraverseHolonymSenses()
 *
 * This function loops through all senses of the search string
 * and calls TraverseHolonyms() for each sense.
 *
 *******************************************************************************/

void TraverseHolonymSenses(SynsetPtr synset_ptr)
{
        int sense_counter = 1;
        //save pointer passed to this function
        SynsetPtr cur_sense_ptr = synset_ptr;

   //loop through all senses of the searchstr
   while(cur_sense_ptr)
   {
                //if there are meronyms for this sense, print them
                if(cur_sense_ptr -> ptrlist)
                {

                        PrintSenseHeader(sense_counter++, cur_sense_ptr -> defn);

                        //traverse and print meronyms for this sense
                        TraverseHolonyms(cur_sense_ptr);
                }

                //move to the next sense
```

```
        cur_sense_ptr = cur_sense_ptr -> nextss;
    }
}

/*****************************************************************************
 *
 * TraverseMeronymSenses()
 *
 * This function loops through all senses of the search string
 * and calls TraverseMeronyms() for each sense.
 *
 *****************************************************************************/

void TraverseMeronymSenses(SynsetPtr synset_ptr)
{
            int sense_counter = 1;
            //save pointer passed to this function
            SynsetPtr cur_sense_ptr = synset_ptr;

    //loop through all senses of the searchstr
    while(cur_sense_ptr)
    {
                        //if there are meronyms for this sense, print them
                        if(cur_sense_ptr -> ptrlist)
                        {

                                    PrintSenseHeader(sense_counter++, cur_sense_ptr -> defn);

                                    //traverse and print meronyms for this sense
                                    TraverseMeronyms(cur_sense_ptr);
                        }

                        //move to the next sense
        cur_sense_ptr = cur_sense_ptr -> nextss;
    }
}


/*****************************************************************************
 *
 * TraverseEntailmentSenses()
 *
 * This function loops through all senses of the search string
 * and calls TraverseEntailments() for each sense to print the ontology
 *
 *****************************************************************************/

void TraverseEntailmentSenses(SynsetPtr synset_ptr)
{
            int sense_counter = 1;
            //save pointer passed to this function
            SynsetPtr cur_sense_ptr = synset_ptr;

    //loop through various senses of the searchstr
    while(cur_sense_ptr)
    {
                        //if there are entailments for this sense, pritn them
                        if(cur_sense_ptr -> ptrlist)
                        {
                                    PrintSenseHeader(sense_counter++, cur_sense_ptr -> defn);

                                    //traverse and print entailments for this sense
```

```
                        TraverseEntailments(cur_sense_ptr);
                }

                //move to the next sense
        cur_sense_ptr = cur_sense_ptr -> nextss;
    }
}


/*****************************************************************************
 *
 * TraverseCausationSenses()
 *
 * This function loops through all senses of the search string
 * and calls TraverseCausations() for each sense to print the ontology
 *
 *****************************************************************************/

void TraverseCausationSenses(SynsetPtr synset_ptr)
{
        int sense_counter = 1;
        //save pointer passed to this function
        SynsetPtr cur_sense_ptr = synset_ptr;

    //loop through various senses of the searchstr
    while(cur_sense_ptr)
    {
                //if there are causations for this sense, pritn them
                if(cur_sense_ptr -> ptrlist)
                {
                        PrintSenseHeader(sense_counter++, cur_sense_ptr -> defn);

                        //traverse and print entailments for this sense
                        TraverseCausations(cur_sense_ptr);
                }

                //move to the next sense
        cur_sense_ptr = cur_sense_ptr -> nextss;
    }
}


/*****************************************************************************
 *
 * The following set of functions will traverse search results for each sense
 *
 *****************************************************************************/


/*****************************************************************************
 *
 * TraverseAntonyms()
 *
 * This function loops through all the antonym synsets for the synset
 * pointed to by synset_ptr and creates their SNePS representations.
 *
 *****************************************************************************/

void TraverseAntonyms(SynsetPtr synset_ptr)
{
        //get point to first antonym
        SynsetPtr cur_synset_ptr = synset_ptr -> ptrlist;
```

```
        //base node name for the synset corresponding to the search string
        char base_node_name_ant1[MAX_BASE_NODE_NAME_LEN];
        //base node name for the synset corresponding to a search result
        char base_node_name_ant2[MAX_BASE_NODE_NAME_LEN];

        GetBaseNodeName(synset_ptr, base_node_name_ant1);

        //print the synset corresponding to the search string
        PrintSynset(synset_ptr);

   //loop through all antonyms
   while(cur_synset_ptr)
   {
                PrintSynset(cur_synset_ptr);

                GetBaseNodeName(cur_synset_ptr, base_node_name_ant2);
                printf("(assert antonym *%s \n %6s antonym *%s)\n\n",
                        base_node_name_ant1,
                        "",
                        base_node_name_ant2);

                //move to the next sense
      cur_synset_ptr = cur_synset_ptr -> nextss;
   }
}


/*****************************************************************************
 *
 * TraverseHypernyms()
 *
 * This function will traverse through hypernym synsets
 * and convert them to SNePS/CVA representations
 *
 *****************************************************************************/

void TraverseHypernyms(SynsetPtr synset_ptr)
{
        char prev_base_node_name[MAX_BASE_NODE_NAME_LEN];
        char cur_base_node_name[MAX_BASE_NODE_NAME_LEN];
        //save pointer passed to this function
        SynsetPtr cur_synset_ptr = synset_ptr;

        //initialize cur_base_node_name
        GetBaseNodeName(cur_synset_ptr, cur_base_node_name);

        //Print the first synset in the hierarchy
        PrintSynset(cur_synset_ptr);

        //Traverse hierarchy
        while(cur_synset_ptr -> ptrlist)
        {
                //move to the next level
                cur_synset_ptr = cur_synset_ptr -> ptrlist;

                //save prev base node name and get the new one
                strcpy(prev_base_node_name, cur_base_node_name);
                //strcpy(cur_base_node_name, GetBaseNodeName(cur_synset_ptr));
                GetBaseNodeName(cur_synset_ptr, cur_base_node_name);

                PrintSynset(cur_synset_ptr);
```

```
                printf("(assert subclass *%s \n %6s superclass *%s)\n\n",
                        prev_base_node_name,
                "",
                        cur_base_node_name);
        }
}


/*****************************************************************************
 *
 * TraverseHolonyms()
 *
 * This function loops through all the meronym synsets for the synset
 * pointed to by synset_ptr and creates their SNePS representations.
 *
 *****************************************************************************/

void TraverseHolonyms(SynsetPtr synset_ptr)
{
        //get point to first meronym
        SynsetPtr cur_synset_ptr = synset_ptr -> ptrlist;

        //base node name for the synset corresponding to the search string
        char base_node_name_whole[MAX_BASE_NODE_NAME_LEN];
        //base node name for the synset corresponding to a search result
        char base_node_name_part[MAX_BASE_NODE_NAME_LEN];

        GetBaseNodeName(synset_ptr, base_node_name_part);

        //print the synset corresponding to the search string
        PrintSynset(synset_ptr);

  //loop through search results
        //base_node_name_part remains the same because it came from the searchstr
        //on each iteration we retrieve the next base_node_name_whole
  while(cur_synset_ptr)
  {
                PrintSynset(cur_synset_ptr);

                GetBaseNodeName(cur_synset_ptr, base_node_name_whole);
                printf("(assert whole *%s \n %6s part *%s)\n\n",
                        base_node_name_whole,
                        "",
                        base_node_name_part);

                //move to the next sense
     cur_synset_ptr = cur_synset_ptr -> nextss;
  }
}


/*****************************************************************************
 *
 * TraverseMeronyms()
 *
 * This function loops through all the meronym synsets for the synset
 * pointed to by synset_ptr and creates their SNePS representations.
 *
 *****************************************************************************/

void TraverseMeronyms(SynsetPtr synset_ptr)
```

```
{
        //get point to first meronym
        SynsetPtr cur_synset_ptr = synset_ptr -> ptrlist;

        //base node name for the synset corresponding to the search string
        char base_node_name_whole[MAX_BASE_NODE_NAME_LEN];
        //base node name for the synset corresponding to a search result
        char base_node_name_part[MAX_BASE_NODE_NAME_LEN];

        GetBaseNodeName(synset_ptr, base_node_name_whole);

        //print the synset corresponding to the search string
        PrintSynset(synset_ptr);

   //loop through search results
        //base_node_name_whole remains the same because it came from the searchstr
        //on each iteration we retrieve the next base_node_name_part
   while(cur_synset_ptr)
   {
                PrintSynset(cur_synset_ptr);

                GetBaseNodeName(cur_synset_ptr, base_node_name_part);
                printf("(assert whole *%s \n %6s part *%s)\n\n",
                        base_node_name_whole,
                        "",
                        base_node_name_part);

                //move to the next sense
      cur_synset_ptr = cur_synset_ptr -> nextss;
   }
}


/*****************************************************************************
 *
 * TraverseEntailments()
 *
 * This function loops through all the entailment synsets for the synset
 * pointed to by synset_ptr and creates their SNePS representations.
 *
 *****************************************************************************/

void TraverseEntailments(SynsetPtr synset_ptr)
{
        //get pointer to first entailment
        SynsetPtr cur_synset_ptr = synset_ptr -> ptrlist;

        //base node name for the synset corresponding to the search string
        char base_node_name_action[MAX_BASE_NODE_NAME_LEN];
        //base node name for the synset corresponding to a search result
        char base_node_name_entails[MAX_BASE_NODE_NAME_LEN];

        GetBaseNodeName(synset_ptr, base_node_name_action);

        //print the synset corresponding to the search string
        PrintSynset(synset_ptr);

   //loop through search results
        //base_node_name_action remains the same because it came from the searchstr
        //on each iteration we retrieve the next base_node_name_entails
   while(cur_synset_ptr)
   {
```

```
                PrintSynset(cur_synset_ptr);

                GetBaseNodeName(cur_synset_ptr, base_node_name_entails);
                printf("(assert verb-concept *%s \n %6s entails *%s)\n\n",
                        base_node_name_action,
                        "",
                        base_node_name_entails);

                //move to the next sense
        cur_synset_ptr = cur_synset_ptr -> nextss;
    }
}


/*****************************************************************************
 *
 * TraverseCausations()
 *
 * This function loops through all the causation synsets for the synset
 * pointed to by synset_ptr and creates their SNePS representations.
 *
 *****************************************************************************/

void TraverseCausations(SynsetPtr synset_ptr)
{
        //get point to first meronym
        SynsetPtr cur_synset_ptr = synset_ptr -> ptrlist;

        //base node name for the synset corresponding to the search string
        char base_node_name_cause[MAX_BASE_NODE_NAME_LEN];
        //base node name for the synset corresponding to a search result
        char base_node_name_effect[MAX_BASE_NODE_NAME_LEN];

        GetBaseNodeName(synset_ptr, base_node_name_cause);

        //print the synset corresponding to the search string
        PrintSynset(synset_ptr);

    //loop through search results
        //base_node_name_cause remains the same because it came from the searchstr
        //on each iteration we retrieve the next base_node_name_effect
    while(cur_synset_ptr)
    {
                PrintSynset(cur_synset_ptr);

                GetBaseNodeName(cur_synset_ptr, base_node_name_effect);
                printf("(assert cause *%s \n %6s effect *%s)\n\n",
                        base_node_name_cause,
                        "",
                        base_node_name_effect);

                //move to the next sense
        cur_synset_ptr = cur_synset_ptr -> nextss;
    }
}


/*****************************************************************************
 *
 * The following are utility functions
 *
 *****************************************************************************/
```

```
/*****************************************************************************
 *
 * GetBaseNodeName()
 *
 * This function generates base node name for the given synset;
 * the newly generated base node name is returned via the second argument
 *
 *****************************************************************************/

char * GetBaseNodeName(SynsetPtr synset_ptr, char * result)
{
        char base_node_name[MAX_BASE_NODE_NAME_LEN]; //new base node name to return
        int word_num; //loop counter

        //concatenate all words in synset
        strcpy(base_node_name, synset_ptr -> words[0]);
        for(word_num = 1; word_num < synset_ptr -> wcount; word_num++)
        {
                sprintf(base_node_name, "%s-%s",
                        base_node_name, synset_ptr -> words[word_num]);

        }

        //replace possible quote char with '-'
        ReplaceChar(base_node_name, '\'', '-');

        //attach part of speech and this synset's offset in data file
        sprintf(base_node_name, "%s-%c-%d",
                base_node_name, *(synset_ptr -> pos), synset_ptr -> hereiam);

        return strcpy(result, base_node_name);
}

/*****************************************************************************
 *
 * PrintSynset()
 *
 * This function will print the synset pointed to by the argument passed
 * to this function. The synset will be printed as a SNePSUL expression:
 * the concept which this synset represents is pointed to by the "concept"
 * arc; each member of this synset is pointed to by the "word" arc.
 *
 *****************************************************************************/

void PrintSynset(SynsetPtr synset_ptr)
{
        int word_num = 0; //loop counter
        char base_node_name[MAX_BASE_NODE_NAME_LEN]; //base node unique identifer

        //generate base node name
        GetBaseNodeName(synset_ptr, base_node_name);

        //print the synset pointed to by synset_ptr in SNePSUL
        printf("(assert concept #%s \n %6s word \"%s\")\n",
                        base_node_name,
                                        "",
                                        synset_ptr -> words[0]);

        for(word_num = 1; word_num < synset_ptr -> wcount; word_num++)
        {
```

```
                        printf("(assert concept *%s \n %6s word \"%s \")\n",
                                        base_node_name,
                                                        " ",
                                        synset_ptr -> words[word_num]);
                }
                printf("\n");
}


/*****************************************************************************
 *
 * PrintSynsetStruct()
 *
 * This function is useful for debugging purposes
 * it prints the content of the synset structure (ss)
 * defined in $WNHOME/src/include/wntypes.h
 *
 *****************************************************************************/

void PrintSynsetStruct(SynsetPtr synset_ptr)
/* this function is useful for debugging */
{
                int loop_count;

                printf("\n***** SYNSET STRUCT DUMP *****\n");
                printf("current file position: %d\n", synset_ptr -> hereiam);
                printf("type of ADJ synset: %d\n", synset_ptr -> sstype);
                printf("file number that synset comes from: %d\n", synset_ptr -> fnum);
                printf("part of speech: %c\n", *(synset_ptr -> pos));
                printf("number of words in synset: %d\n", synset_ptr -> wcount);
                printf("words: ");
                for(loop_count = 0; loop_count < synset_ptr -> wcount; loop_count++)
                {
                        printf("%s, ", synset_ptr -> words[loop_count]);
                }
                printf("\n");
                printf("unique id in lexicographer file: %d\n", *(synset_ptr -> lexid));
                printf("unique id in lexicographer file: %d\n", *((synset_ptr -> lexid) + 1));
   printf("sense number in wordnet: %d\n", *(synset_ptr -> wnsns));
   printf("which word in synset we're looking for: %d\n", synset_ptr -> whichword);
   printf("number of pointers: %d\n", synset_ptr -> ptrcount);
                printf("pointer type: %d\n", *(synset_ptr -> ptrtyp));
                printf("pointer offsets: %l\n", *(synset_ptr -> ptroff));
                printf("pointer part of speech: %d\n", *(synset_ptr -> ppos));
                printf("pointer to fields: %d\n", *(synset_ptr -> pto));
                printf("pointer from fields: %d\n", *(synset_ptr -> pfrm));
                //printf("number of verb frames: %d\n", synset_ptr -> fcount);
                //printf("frame numbers: %d\n", *(synset_ptr -> frmid));
                //printf("frame to field: %d\n", *(synset_ptr -> frmto));
                printf("synset gloss: %s \n", synset_ptr -> defn);
   printf("unique synset key: %d\n", synset_ptr -> key);
                printf("***** END SYNSET STRUCT *****\n\n");
}


/*****************************************************************************
 *
 * PrintDefinition()
 *
 * Print the string passed to this function, inserting an EOL character
 * every time line_length characters have been printed. This function is
 * intended to improve formatting of the synset definition output
```

```
 *
 ****************************************************************************/

void PrintDefinition(char * definition)
{
	int i; //loop index
	int const line_length = 45; //number of characters in one line

	for(i = 0; i < strlen(definition); i++)
	{
		if(i%line_length == 0)
		{
			//print EOL every time we've printed line_length characters
			printf("\n; ");
		}

		printf("%c", definition[i]);
	}
	printf("\n");
}


/****************************************************************************
 *
 * PrintSearchHeader()
 *
 ****************************************************************************/

void PrintSearchHeader(char * searchtype, int pos, char * searchstr)
{
	char pos_str[MAX_POS_NAME_LEN]; //pos name (e.g. "noun" or "adjective")
	GetPOSName(pos, pos_str); //get pos name

	printf("\n; ------------------------------------------------\n");
	printf("; ------------------------------------------------\n");
	printf(";  %s search results for %s \"%s\"\n",
		searchtype, pos_str, searchstr);
	printf("; ------------------------------------------------\n");
	printf("; ------------------------------------------------\n\n");
}


/****************************************************************************
 *
 * PrintSenseHeader()
 *
 ****************************************************************************/

void PrintSenseHeader(int sensenum, char * definition)
{
		//print sense number and the textual gloss for this sense
		printf("; ---------- sense %d ----------",
			sensenum);
		PrintDefinition(definition);
		printf("; ------------------------------------------------\n\n");
}


/****************************************************************************
 *
 * ReadFileIntoString()
 *
```

```
 * This function slurps a text file into a string. It operates by reading
 * the file character by charactre into the string passed to this function
 * as a parameter until the end of file character is reached. The function
 * returns the number of characters read.
 *
 ******************************************************************************/

int ReadFileIntoString(char * file_name, char * result_str)
{
	FILE * input_file_ptr = NULL;
	int char_count = 0;

	//open intput file
	input_file_ptr = fopen(file_name, "r");
	if(!input_file_ptr) {printf("couldn't open input file!\n"); exit(1);}

	//slurp the entire file into result_str
	while((result_str[char_count++] = fgetc(input_file_ptr)) != EOF);

	fclose(input_file_ptr); //close input file

	//return the number of characters in input file
	return char_count;
}


/******************************************************************************
 *
 * Tokenize()
 *
 * The first argument of this function is a text string. The second argument
 * is a table (an array of strings).
 *
 * This function will filter out all non-alpha characters from the string
 * passed to this function as an argument. The string will be split into
 * tokens (words, i.e. - strings separated by empty space). The result
 * will be placed into the table passed to this function as the second
 * argument.
 *
 * This function returns the number of words in the resulting table.
 *
 ******************************************************************************/

int Tokenize(char * passage, char tokenized_passage[][MAX_WORD_LEN])
{
	//allowable ascii character ranges ('A'-'Z' or 'a' - 'z')
	int const ascii_range_low1 = 65; //capital 'A'
	int const ascii_range_high1 = 90; //capital 'Z'
	int const ascii_range_low2 = 97; //small 'a'
	int const ascii_range_high2 = 122; //small 'z'

	int loop_index = 0; //loop index
	int char_count = 0; //used to count chars in a word
	int word_count = 0; //used to count the number of words in passage

	//loop through every character in the passage
	//excluding the last one (EOF)
	for(loop_index = 0; loop_index < strlen(passage); loop_index++)
	{
		//check if this is the end of the word
		if((passage[loop_index] == ' ')
			||
```

```
                (passage[loop_index] == '\n'))
            {
                //insert a null terminating char at the end of current word
                tokenized_passage[word_count][char_count] = '\0';

                //make sure this word is not an empty string
                //(so that an empty space after an empty space
                // does not count as a word)
                if(strlen(tokenized_passage[word_count]) != 0)
                {
                    word_count++; //go to the next position in the table
                }

                char_count = 0; //begin filling a new word
                continue; //begin the loop over
            }

            //check if this character falls within the allowable range
            if(((passage[loop_index] >= ascii_range_low1) &&
               (passage[loop_index] <= ascii_range_high1))
                ||
              ((passage[loop_index] >= ascii_range_low2) &&
               (passage[loop_index] <= ascii_range_high2))
               ||
                (passage[loop_index] == '_') //'_' is allowed
                ||
                (passage[loop_index] == '-')) //'-' is allowed
            {
                //save the character and increment the character count
                tokenized_passage[word_count][char_count++] = passage[loop_index];
            }
        }

        //return the number of words in passage
        return word_count;
}


/*****************************************************************************
 *
 * GetPOSName()
 *
 * This function is used to convert int POS code into string
 *
 *****************************************************************************/

void GetPOSName(int pos, char * result)
{
        switch(pos)
        {
            case NOUN:
                strcpy(result, "noun");
                break;

            case VERB:
                strcpy(result, "verb");
                break;

            case ADJ:
                strcpy(result, "adjective");
                break;
```

```
                case ADV:
                        strcpy(result, "adverb");
                        break;

                default:
                        strcpy(result, "unknown POS");
                        break;
        }
}


/*****************************************************************************
 *
 * ReplaceChar()
 *
 * This function is used to replace one character within a string with another
 *
 *****************************************************************************/

void ReplaceChar(char * str, char char_to_replace, char replacement_char)
{
        int index = 0;

        for(index = 0; index < strlen(str); index++)
        {
                if(str[index] == char_to_replace)
                {
                        str[index] = replacement_char;
                }
        }
}
```

References

1. Becker, Chris (2004), “Contextual Vocabulary Acquistion; Contextual Information in Verb Contexts: from analysis to algorithm”. Available online at http://www.cse.buffalo.edu/~rapaport/CVA/becker-verbs.pdf

2. Beckwith, Richard, Miller, George, A, & Tengi, Randee. “Design and implementation of the WordNet lexical database and searching software”. Included in wordNet software distribution. See http://www.cogsci.princeton.edu/~wn/.

3. Ehrlich, Karen, & Rapaport, William J. (1995), "A Computational Theory of Vocabulary Expansion: Project Proposal", Technical Report 95-15 (Buffalo: SUNY Buffalo Department of Computer Science) and Technical Report 95-08 (Buffalo: SUNY Buffalo Center for Cognitive Science).

4. Fellbaum, Christiane, (1990) “English Verbs as a Semantic Net”, International Journal of Lexocography 3(4): 270-301.

5. Miller, George, A (1990) “Nouns in WordNet: a lexical inheritance system”, International Journal of Lexicography, 3(4), 245 – 264.

6. Rapaport, William J., & Ehrlich, Karen (2000), "A Computational Theory of Vocabulary Acquisition", in Lucja M. Iwanska & Stuart C. Shapiro (eds.), Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language (Menlo Park, CA/Cambridge, MA: AAAI Press/MIT Press): 347-375.

7. Rapaport, William J., & Kibby, Michael W. (2002), "ROLE: Contextual Vocabulary Acquisition: From Algorithm to Curriculum".

8. Shapiro, Stuart C., & Rapaport, William J. (1987), "SNePS Considered as a Fully Intensional Propositional Semantic Network", in Nick Cercone & Gordon McCalla (eds.), The Knowledge Frontier: Essays in the Representation of Knowledge (New York: Springer-Verlag): 262-315.