Jabberwocky: Using Context Clues to Find Meaning in 'Nonsensical Speech'

Paul M. Heider

pmheider@buffalo.edu

CSE727: Contextual Vocabulary Acquisition

May 10, 2007

Abstract

No matter what your preferred method of counting is, the average adult knows a very large number of words. Moreover, most of these words were not—and could not have been—explicitly taught to the individual. Contextual Vocabulary Acquisition (CVA) is an explanation via computational model of how these underspecified terms slip into our daily vocabulary and how to help people actively acquire them. It is instantiated in a knowledge representation, reasoning, and acting system called SNePS. We endow CASSIE, a SNePS-based agent, with the world knowledge required to understand a passage. As she is exposed to more of the text, we can query her regarding new inferences about and information gained from the reading. One advantage of CVA over other models is that CASSIE's need for an explicit and computable algorithm forces us to fully define all facets of our theory. I will analyze CASSIE's ability to garner information from Lewis Carroll's *Jabberwocky*, a poem famous for its ability to convey meaning despite the vast number of nonsense words. Specifically, I am interested in what CASSIE understands about a "jabberwock" from a context with as many novel as familiar words. The conclusion offers an analysis of CASSIE's results in comparison to human interpretations of the same passage. Finally, I discuss methods for increasing CASSIE's understanding of the poem and how to streamline her prior knowledge.

1 The CVA Project

With low end estimate at 45,000 words (Nagy and Anderson, 1984) and higher estimates at least an order of magnitude larger (Berwick, 1989), the average adult has accumulated a substantial vocabulary. What more, the majority of these words (according to Nagy and Anderson, 90%) are not 'taught' but must be inferred from discourse, reading, etc.

In all cases, these new words occur in some sort of environment and that environment, more often that not, contains some clues or feedback about features of the word. When a reader unconsciously uses these clues to develop and understanding of a word, we call this incidental contextual vocabulary acquisition (CVA). Much less often but still achievalbe, a reader will set out to actively collect and analyze these clues (via deliberate CVA). Ideally, institutations of education would help guide students who have difficulty with incidental CVA by training them on deliberate CVA methods.[1] According to overview by Rapaport and Kibby (2006), the science of teaching deliberate CVA is still very, very peurile. The main weakness in the state of the art strategies is that they are most vague at the point when the student needs the most guidance. For instance, the strategy described for English as a Second Language in Clarke and Nation (1980) (via Rapaport and Kibby (2006)) puts all of the focus into "guess the word". Woe be to the student who does not already have clear methods for developing textual clues into possible guesses and good heuristics choosing guesses from the lot.

In contrast with these attempts, we are trying to develop the algorithmic side of CVA through computational modeling and modeling of human protocols. It is instantiated in a LISP-based knowledge representation, reasoning, and acting system called SNePS. I will leave a more thorough explanation of the SNePS engine to the next section. One advantage of CVA over other models is that the system's need for an explicit and computable algorithm forces us to fully define all facets of our theory. Generally speaking, we endow CASSIE, a SNePS-based agent, with the world knowledge required to understand a passage. As she is exposed to more of the text, we can query her regarding new inferences about and information gained from the reading. As we are interested in vocabulary acquisition, a single word from the text is chosen as the target unknown word which we wish to learn more about. Most questions to CASSIE are directed at better understanding that word.

More specifically, any given text (below, I will be analyzing Lewis Carroll's *Jabberwocky* from *Through the Looking-Glass and What Alice Found There*) is translated into the equivalent SNePS's node-network. I have used responses to the text elicited from humans to help guide my decision as to what prior knowledge is also necessary to understand the text. These background rules and world facts are not explicitly tied in any way to the target word and no information is coded about the 'unknown' word, either. We then feed CASSIE the passage's SNePS representation one unit (usually a clause) at a time to simulate reading. Finally, we invoke a definition algorithm that tries to analyze this created semantic network from the perspective of the unknown word. What concepts are directly related (first-order) to it? What do we know about the other concepts normally associated with first-order concepts?

After a brief introduction to SNePS, I will analyze the human protocols that were gathered and the subsequent background knowledge I felt was necessary to encode. I will discuss CASSIE's ability to garner information from the passage and, finally, what developments could be made to the general CVA algorithm to improve the reading of this

[1] Imagine how large an advantage those proficient users of CVA have if they know and understand even 50% more words than their unskilled counterparts.

and other texts. I thought *Jabberwocky* would be a fitting text to analyze meaning recovery as it is most famous for its ability to convey meaning despite the vast number of nonsense words. As Alice describes the passage, “Somehow it seems to fill my head with ideas—only I don’t exactly know what they are!”.

1.1 A SNePS Background

Semantic networks come in many shapes and flavors. Some of the major differences lie in the meaning attached to each node, the paths between nodes, and how inferences are made about the nodes and paths. Some systems, for example, assign concepts and relations only to the nodes. Meaning is depicted in the network by conceptual nodes connected to each other through relational nodes. SNePS, in contrast, is an inferencing system that assigns meaning to both nodes and paths. Inferences can be made based on matching path structures, matching nodes, or some combination of the two. The paths used for this system are bi-directional but not transitive. In other words, you can move across a path in either direction but the directions are not necessarily treated equally. Finally, the system keeps separate those nodes which are believed by the system to be true (“asserted,” in the language of SNePS) and those which could possibly be true. Among other uses, these unasserted nodes can function as propositions believed by other agents or untested hypotheses, depending on the purpose of the network.

In figure 1, the proposition “colorless ideas” can be depicted by the node m1! with an object arc to word “ideas” and a property arc to the word “colorless”. The ! (bang) labels those nodes which are believed by the system (for our purposes, CASSIE). In addition to the meaning mentioned above, this contrast can also help us to differentiate between CASSIE knowing that there exists some colorless ideas and the abstract concept of colorless ideas.

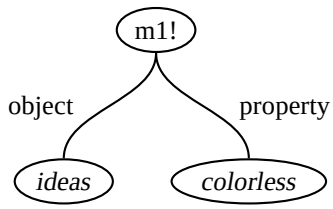


Figure 1: The Simple Proposition “colorless ideas” in SNePS

In one view of SNePS networks, each node represents a separate proposition. Sentences are traditionally modeled as a single proposition with sub-propositions. For example, the syntax for a basic declarative sentence is shown in figure 2. I have used a dashed line to represent optional arguments. The semantics of the sentence pictured are as follows: some agent “NP” is performing the action “Verb” with direct object modifier “DO” and an optional indirect object “IO”.

In the last two examples, the network has only shown relationships between concepts. Two of the standard case frames (i.e., a set of arc labels with an associated semantic interpretation) attach these abstract concepts to the words

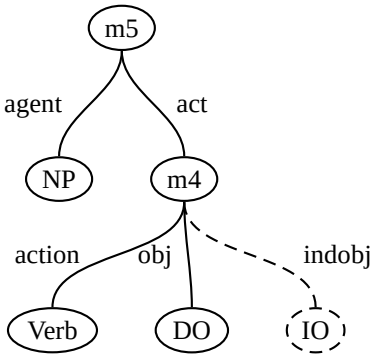
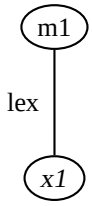
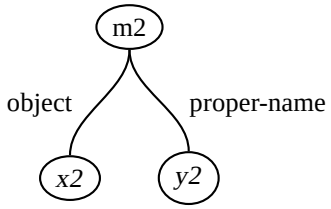


Figure 2: A Simple Declarative Sentence Template in SNePS

which refer to them: LEX and OBJECT/PROPER-NAME. The semantic interpretation for these frames are listed below the syntax in figure 3. There are six more standard frames I make use of: MEMBER/CLASS, SUBCLASS/SUPERCLASS, OBJECT/PROPERTY, PART/WHOLE, OBJECT/REL/POSSESSOR, and OBJECT1/REL/OBJECT2. They have been grouped into sets of two with the semantics and syntax likewise juxtaposed.[2] My use of the case frames is intended to mirror the semantics prescribed by the algorithm and can be further researched in the SNePS case frame dictionary (Shapiro et al., 1996). I will cover the non-standard case frames employed in the final discussion section as they did not influence CASSIE's understanding of "jabberwock".



[[*m*1]] is the concept expressed by uttering [[*x*1]]



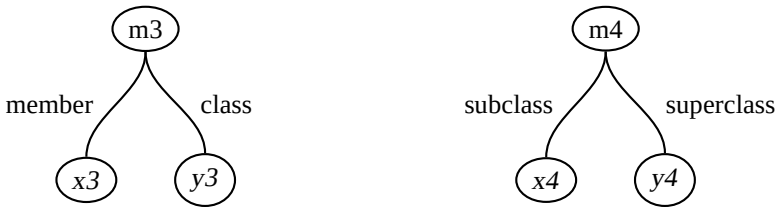
[[*m*2]] is the proposition
[[*x*2]] is called by the proper name [[*y*2]]

Figure 3: Referent Case Frame Syntax and Semantics

2 Human Protocols

The original text of *Jabberwocky* consists of seven stanzas. For the purposes of this project, I have eliminated the first and last stanza as they do not contain any references to our target word "jabberwock". The remaining stanzas were labeled one through five (with a base zero if you wish to include the dropped stanza). Before interviewing a subject,

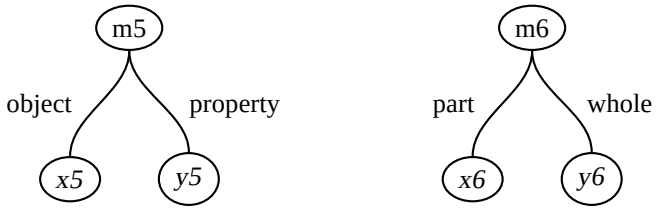
[2]This manner of grouping is not the standard system but one I find mnemonically useful.



[[*m*3]] is the proposition
[[*x*3]] is a member of the class [[*y*3]]

[[*m*4]] is the proposition
[[*x*4]] is a subclass of [[*y*4]]

Figure 4: Ontological Case Frame Syntax and Semantics



[[*m*5]] is the proposition
[[*x*5]] has the property [[*y*6]]

[[*m*6]] is the proposition
[[*x*6]] is a part of [[*y*6]]

Figure 5: Elemental Case Frame Syntax and Semantics

I asked if she had any prior knowledge of the poem or word. At two of the participants had of the poem but did not know what it was about. My protocols come from 2 native speakers of English and 3 non-native speakers (although one of the former has a graduate degree in literary criticism). The passage was displayed one stanza at a time with the highlighting shown below (and in the appendix) to emphasize the any reference to the target word. After the consultant had an opportunity to read the newest part of the passage, I interrogated him as to his understanding of the term. In the case of a null response, I asked if a particular word in the passage influenced his opinion (e.g., “Where you surprised the jabberwock had a head?”). Any other response had to be supported with evidence from the text or general world knowledge (e.g., “It has a big head because [the head] makes [the hero] galumph [and] galumph sounds like ‘awkward’, ‘humpy’, ‘gallop”). Lastly, I thought it was important to include native and non-native speakers because much of the text includes artificial words. Some of these words have onomatopoeic qualities and I thought different language bases would help normalize any such influence.

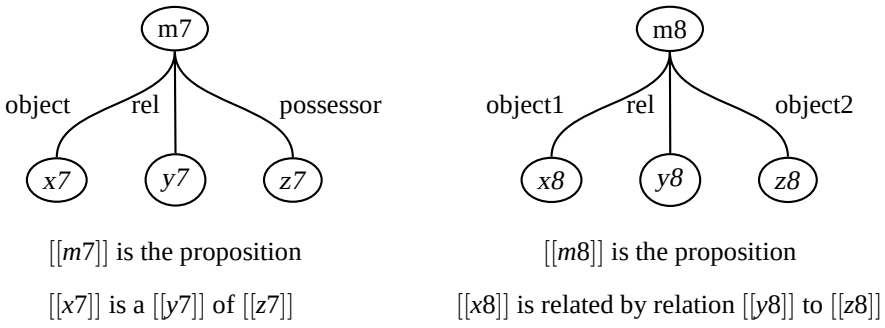


Figure 6: Relational Case Frame Syntax and Semantics

(1) "Beware the **Jabberwock**, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"

2.1 Stanza One

The responses to the first stanza ranged from broadest "not human" (CO) to the already specific "a dangerous monster or giant bird". The primary clues cited initially were the presence of 'jaws' and 'claws'. The general consensus was that both of these are common features of animals, monsters, and birds but not of humans. Next, the warning to 'beware' implies the Jabberwock is somehow dangerous. One particular informant went so far as to say that 'bite' and 'catch' must mean abnormal (and thus, most likely dangerous) version of the more standard action. Many people cited the parallel structure with the 'Jubjub bird' as evidence for the Jabberwock's avian status.

From this section, I chose to develop those features in CASSIE's prior knowledge which would emphasize the features 'monster' and 'dangerous'. As you can see from the algorithm's output after reading the first stanza, anatomical features are apparent, one superclass of the Jabberwock can be derived, and CASSIE can infer at least one property. Discussion in the next section explains how my particular background knowledge helped the reader to derive this early definition.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Ask Cassie what "JabberwockY" means:
^(
--> defineNoun "JabberwockY")
 Definition of JabberwockY:
 Class Inclusions: monster,
```

```
 Possible Structure: jaws, jaws claws,
 Possible Actions: bite,
 Possible Properties: dangerous,
nil
* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

2.2 Stanza Two

(2) He took his vorpal sword in hand:
Long time the manxome **foe** he sought –
So rested he by the Tumtum tree,
And stood awhile in thought.

The second stanza revealed more relational features. Everyone noted that ‘sought’ had strong implications but disagreed as to exactly what they were. It could mean the Jabberwock is valuable or that it is a good hider. The beast must also be mortal because, as CO put it, “swords are for slaying”. The two most interesting reactions to the word ‘enemy’ were from RA and VA. The former thought only a truly worthy creature could be a enemy. The jabberwock was thus a fabulous creature. The latter believed ‘enemy’ reinforced her notion of dangerous and harmful from the first stanza.

I thought the relationship of enemy was most important to address. The other features were very difficult to develop and did not seem to provide any knowledge not imparted by another part of the poem. ‘Evil’, as you can see CASSIE inferred, may not be a very salient feature to you. Again, I will defer explanation until the advertised discussion of background knowledge.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Ask Cassie what "JabberwockY" means:
^(
--> defineNoun "JabberwockY")
 Definition of JabberwockY:
 Class Inclusions: monster,
 Possible Structure: jaws, jaws claws,
 Possible Actions: bite,
 Possible Properties: evil, dangerous, foe of human,
nil
```

```
* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

2.3 Stanza Three

(3) And, as in uffish thought he stood,
The **Jabberwock**, with eyes of flame,
Came whiffling through the tulgey wood,
And burbled as it came!

The consultants' assumptions about anatomy were brought to bear on this stanza. Most had accepted the mention of jaws and claws as significant. Perhaps because eyes were so essentially part of the picture they had already constructed, the eyes were initially only interpreted metaphorically. The author should not need to mention physical eyes and thus did not. It was also interesting to hear 'whiffle' be described as 'flying' in as many different ways as I had informants.

Because of this stanza, I include a section on anatomy in the prior knowledge and defined 'whiffling' as a manner of flight. The extra definitional aspects produced by the third stanza would probably only appear in a more thorough definition done by a human. For CASSIE the additional information regarding actions helps her to create classes of individuals with shared action spaces.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Ask Cassie what "JabberwockY" means:
^(
--> defineNoun "JabberwockY")
 Definition of JabberwockY:
 Class Inclusions: monster,
 Possible Structure: jaws, jaws claws, jaws eyes,
 Possible Actions: whiffle, fly, bite, burble,
 Possible Properties: evil, dangerous, foe of human,
nil
* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

2.4 Stanza Four

As with eyes in the passage above, none of the subjects thought 'head' was a feature worth meaning. Nonetheless, they all concurred with the notion that the Jabberwock had a head. Without a clever psychological experiment, it would be difficult to determine if they resolved the existence of the head only when queried or at the introduction of another

(4) One, two! One, two! And through and through
The vorpal blade went snicker-snack!
He left **it** dead, and with **its** head
He went galumphing back.

highly related feature. 'Galumph' received varied but relatively uninteresting interpretations. The mortality of the creature was again noted. Also, the significance of the head as a trophy could provide fodder for more development of the passage.

The most consistent elements of the Jabberwock's existence have already been covered by other inferences. In fact, the official definition as produce by the algorithm does not change after the third stanza. This stanza did inspire me to add the 'state' arc for reasons I will explain in the discussion section.

2.5 Stanza Five

(5) "And, has thou slain the **Jabberwock**?
Come to my arms, my beamish boy!
O frabjous day! Callooh! Callay!"
He chortled in his joy.

The happiness of the old man to hear of the death led RA and KI to believe the Jabberwock must be bad. CO was only more convinced the Jabberwock must be great, valiant, or powerful to warrant such notoriety in death.

The consultants had exhausted their basic level inferences about the Jabberwock and mostly opted to maintain their earlier beliefs.

3 Current Background Knowledge and Further Work

3.1 Inheritance and Linnaean Knowledge

While perhaps not apropos to every text, my poem benefited greatly from a very general ontology of creatures and their related features. One informant claimed the Jabberwock was naturally a bird because it had claws while another preferred to infer the claws from the fact that the Jabberwock was a bird. We could see a very fruitful correlation between class membership and natural properties. Even the negative membership was inferred from these properties (viz., 'jabberwock' is *not* human because it has jaws and claws).

I think more research could be done into developing a simple ontology of common nouns. There has already been

some work in giving CASSIE access to the notion of basic level terms. The SNePS ability to search along arbitrarily complicated paths has also given CASSIE an broad means to infer interesting implications of property inheritance. I was still forced to explicitly code some rules of feature acquisition. For instance, many speakers felt the Jabberwock was dangerous because it had dangerous clauses. This type of part/whole property inheritance has not been fully explored. The primary difficulty in developing this areas is creating a complete but not contrary or fully circular system. I imagine such a system would be composed of modules that could be individually applied to situations and texts without invoking the entire apparatus.

3.2 Morality

An informant who specializes in literary criticism introduced some very intriguing features derived from expectations of the story and the medium. For instance, the focus or main character of a story is assumed to be moral center and, thus, represents our temporary concept of good. Her enemies are inherently evil and her goals are desirable ends.

On a more general note, I think CASSIE could be invested with similar plot expectations. My background included only two facts: bad is the enemy of good and the main character is always good. Nonetheless, CASSIE's sentiments toward the two primary characters matched those of the readers.

3.3 Semantic Knowledge

Several consultants provided feedback that focused on one particular connotation of a word or another. I have already mentioned the unanimity of 'whiffle' being a subclass of 'fly'. Two more potent examples are 'catch' and 'bite'. Because the author chose to mention these acts, readers felt they must be significant and not the prototypical form of either verb. Even though both were viewed in the same light, I intentionally divided their significance between the two so as to more unique paths to derive a definition rather than a single, all-encompassing path. Specifically, both verbs were thought to imply some sort of dangerous animal. Instead of associated catching and biting with danger, I chose to encode only 'catch' as a dangerous action. I then focused on the anatomical aspect of jaws (i.e., if A has jaws, A must be an animal because all animals have jaws). I do not think the background knowledge related to these features can really be generalized or necessarily put to use outside of this passage. The more important take-away message would be to stay consistent to a particular word sense within a passage. That is, if jaws are implied to be dangerous at the start of a text, any later mentions of jaws are also of the dangerous variety.

3.4 Passage Specific Knowledge

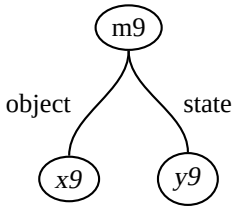
Finally, 'monster', 'beast', and 'creature' were used by all those surveyed. It was an important element of everyone's understanding of the Jabberwock. As part of the general Linnaean taxonomy above, I included special knowledge

about what constitutes a monster. For the purposes of this passage, a monster was any dangerous living thing and, most certainly, different from a human. Much like the additional semantics I discussed above, this prior knowledge is unlikely to transfer well to other texts.

4 Deepening the Passage Comprehension

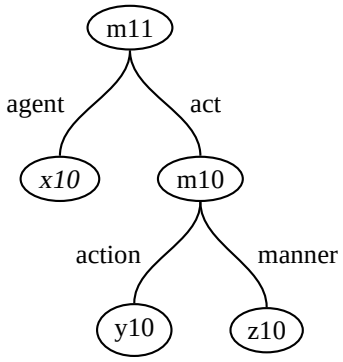
In an effort to improve CASSIE's overall understanding of *Jabberwocky* without regard to defining the unknown word, I have introduced two non-standard case frames. As neither of the two case frames I introduced were significantly used by the algorithm, OBJECT/STATE and AGENT/ACT/ACTION/MANNER both represent extensions that would deepen CASSIE's understanding of the poem. First, I thought it was important to distinguish permanent properties of an object from temporary properties. This distinction is especially important for given passage so as to prevent CASSIE from inferring that 'dead' is a general property of Jabberwock's. There are footprints in the algorithm of a

Development of 'manner' would no doubt be in parallel with an adverbial algorithm. Because of the particular protocols I was working worth, I could have also chosen to encode the relationship between 'whiffle' and 'fly' with a SUBCLASS/SUPERCLASS case frame instead.



[[*m*9]] is the proposition

[[*x*9]] has a non-permanent property [[*y*9]]



[[*m*10]] is the proposition

[[*x*10]] performs the action [[*y*10]] in a [[*z*10]] manner

Figure 7: Non-standard Case Frame Syntax and Semantics

A rather simple step for the future would be to properly apply motion and location case frames to the passage. I know they are currently not used by the algorithm, but they would provide for a more accurate reading of the passage.

Unfortunately, I was only able to use the algorithm to define the class in which the Jabberwock is a member. I would like to see the algorithm develop in a direction that allows for the 'definition' of single instances. Instead asking "What is a Jabberwock", I imagine this algorithm as asking "Who is (a) Jabberwock".

Finally, my encoding of the passage could be aligned with the current algorithm better. There is evidence of my

slight deviation in many “possible” features rather than definite or strict features. Some alignment requires recoding the prior knowledge to be more causal. Other improvements could be made by changing my description of the semantic space. In general, I used fewer ANT/CQ arcs than would be ideal.

A Lewis Carroll's *Jabberwocky*

'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe:
All mimsy were the borogoves,
And the mome raths outgrabe.

"Beware the **Jabberwock**, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"

He took his vorpal sword in hand:
Long time the manxome **foe** he sought –
So rested he by the Tumtum tree,
And stood awhile in thought.

And, as in uffish thought he stood,
The **Jabberwock**, with eyes of flame,
Came whiffling through the tulgey wood,
And burbled as it came!

One, two! One, two! And through and through
The vorpal blade went snicker-snack!
He left **it** dead, and with **its** head
He went galumphing back.

"And, has thou slain the **Jabberwock**?
Come to my arms, my beamish boy!
O frabjous day! Callooh! Callay!"
He chortled in his joy.

'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe:
All mimsy were the borogoves,
And the mome raths outgrabe.

B Complete Demo Transcript

```
Script started on Thu May 10 03:32:07 2007
pollux {~/courses/CSE727/word} > acl
International Allegro CL Enterprise Edition
8.0 [Solaris] (Apr 10, 2007 13:09)
Copyright (C) 1985-2005, Franz Inc., Oakland, CA, USA.  All Rights Reserved.

This development copy of Allegro CL is licensed to:
   [4549] University at Buffalo

;; Optimization settings: safety 1, space 1, speed 1, debug 2.
;; For a complete description of all compiler switches given the
;; current optimization settings evaluate (explain-compiler-settings).
;;---
;; Current reader case mode: :case-sensitive-lower
cl-user(1): (load "/projects/snwiz/bin/sneps")
; Loading /projects/snwiz/bin/sneps.lisp
Loading system SNePS...10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
SNePS-2.6 [PL:2 2007/03/19 18:32:05] loaded.
Type '(sneps)' or '(snepslog)' to get started.
t
cl-user(2): (sneps)


   Welcome to SNePS-2.6 [PL:2 2007/03/19 18:32:05]

Copyright (C) 1984--2004 by Research Foundation of
State University of New York. SNePS comes with ABSOLUTELY NO WARRANTY!
Type '(copyright)' for detailed copyright information.
Type '(demo)' for a list of example applications.

   5/10/2007 3:32:33

* (demo "/home/lingrad/pmheider/courses/CSE727/word/jabberwock.demo")
; Fast loading from bundle code/streama.fasl.

File /home/lingrad/pmheider/courses/CSE727/word/jabberwock.demo is now the
 source of input.


 CPU time : 0.03

* ; ============================================================================
; FILENAME:     jabberwock.demo
; DATE:         2007-05-10
; PROGRAMMER:   Paul M. Heider

;; this template version:       snepsul-template.demo-20061005.txt

; Lines beginning with a semi-colon are comments.
; Lines beginning with "^" are Lisp commands.
; All other lines are SNePSUL commands.
;
```

```
; To use this file: run SNePS; at the SNePS prompt (*), type:
;
;  (demo "/home/lingrad/pmheider/courses/CSE727/word/jabberwock.demo" :av)
;  (demo "/home/lingrad/pmheider/courses/CSE727/word/jabberwock.demo")
;
; Make sure all necessary files are in the current working directory
; or else use full path names.
; ============================================================================

; Turn off inference tracing.
; This is optional; if tracing is desired, then delete this.
^(
--> setq snip:*infertrace* nil)
nil



 CPU time : 0.00

*
; Load the appropriate definition algorithm:
^(
--> load "/projects/rapaport/CVA/STN2/defun_noun.cl")
; Loading /projects/rapaport/CVA/STN2/defun_noun.cl
t



 CPU time : 0.24

* ;
; Clear the SNePS network:
(resetnet)

Net reset - Relations and paths are still defined


 CPU time : 0.00

*
; load all pre-defined relations:
(intext "/projects/rapaport/CVA/STN2/demos/rels")
Loading file /projects/rapaport/CVA/STN2/demos/rels.
; Fast loading /util/acl80/code/streamc.001
;;; Installing foreign patch, version 1
;   Fast loading from bundle code/efft-euc-base.fasl.
;   Fast loading from bundle code/efft-utf8-base.fasl.
;   Fast loading from bundle code/efft-void.fasl.
;   Fast loading from bundle code/efft-latin1-base.fasl.


 CPU time : 0.43

*
```

```
; load all pre-defined path definitions:
(intext "/projects/rapaport/CVA/mkb3.CVA/paths/paths")
Loading file /projects/rapaport/CVA/mkb3.CVA/paths/paths.
before implied by the path (compose before
                            (kstar (compose after- ! before)))
before- implied by the path (compose (kstar (compose before- ! after))
                             before-)
after implied by the path (compose after
                           (kstar (compose before- ! after)))
after- implied by the path (compose (kstar (compose after- ! before))
                            after-)
sub1 implied by the path (compose object1- superclass- ! subclass
                          superclass- ! subclass)
sub1- implied by the path (compose subclass- ! superclass subclass- !
                           superclass object1)
super1 implied by the path (compose superclass subclass- ! superclass
                            object1- ! object2)
super1- implied by the path (compose object2- ! object1 superclass- !
                             subclass superclass-)
superclass implied by the path (or superclass super1)
superclass- implied by the path (or superclass- super1-)


 CPU time : 0.06

*
; object/state: a subclass of the object/property
; for (assert object A state B), object A is said to be in the state B where
; state is some non-permanent property currently assigned to A
(define state)

(state)

 CPU time : 0.00

*
; agent/act/action/manner:  a subclass of the agent/act/action
; for (assert agent A act (build action B manner C)), C correlates to a
; standard definition of an adverb:  A performs B in the manner of C.
;(define manner)

; BACKGROUND KNOWLEDGE:
; =====================

;; animacy is a property with two possible states:  dead or alive
;; any individual of a class who has the proper animate can either
;; be in the state of dead or in the state of alive (and is alive by default)

;;;; ANIMALS ;;;;

;; birds are animals
(describe
 (assert subclass (build lex "bird") = bird
         superclass (build lex "animal") = animal))
```

```
(m3! (subclass (m1 (lex bird))) (superclass (m2 (lex animal))))

(m3!)

 CPU time : 0.00

*
;; animals are animate
(describe
 (assert object *animal
   property (build lex "animate") = animate))

(m5! (object (m2 (lex animal))) (property (m4 (lex animate))))

(m5!)

 CPU time : 0.00

*
;; Animals have jaws
(describe
 (assert whole *animal part (build lex "jaws")))

(m7! (part (m6 (lex jaws))) (whole (m2 (lex animal))))

(m7!)

 CPU time : 0.00

*
;; Animals have claws
(describe
 (assert whole *animal part (build lex "claws")))

(m9! (part (m8 (lex claws))) (whole (m2 (lex animal))))

(m9!)

 CPU time : 0.01

*
;; Animals have eyes
(describe
 (assert whole *animal part (build lex "eyes")))

(m11! (part (m10 (lex eyes))) (whole (m2 (lex animal))))

(m11!)

 CPU time : 0.00

*
;; Animals have a head
```

```
(describe
 (assert whole *animal part (build lex "head")))

(m13! (part (m12 (lex head))) (whole (m2 (lex animal))))

(m13!)

 CPU time : 0.00

*
;; dangerous animals are monsters
;; All P that is a member of the class animal and has the property dangerous
;; is a monster
(describe
 (assert forall $p
   &ant ((build member *p class *animal)
            (build object *p property (build lex "dangerous")))
   cq (build member *p class (build lex "monster"))))

(m16! (forall v1)
 (&ant (p2 (object v1) (property (m14 (lex dangerous))))
  (p1 (class (m2 (lex animal))) (member v1)))
 (cq (p3 (class (m15 (lex monster))) (member v1))))

(m16!)

 CPU time : 0.00

*
;; All P that is a member of some subclass of animal and that has the
;; property dangerous is a monster
(describe
 (assert forall ($p $q)
   &ant ((build member *p class *q)
            (build subclass *q superclass (build lex "animal"))
            (build object *p property (build lex "dangerous")))
   cq (build member *p class (build lex "monster"))))

(m17! (forall v3 v2)
 (&ant (p6 (object v2) (property (m14 (lex dangerous))))
  (p5 (subclass v3) (superclass (m2 (lex animal))))
  (p4 (class v3) (member v2)))
 (cq (p7 (class (m15 (lex monster))) (member v2))))

(m17!)

 CPU time : 0.01

*
; All Q, that have a member which is a monster, is a class of monsters
; (that is, all members of Q are monsters)

;; All Q, that have a member which is a monster, is a subclass of monsters
(describe
```

```
  (assert forall ($p $q)
    &ant ((build member *p class (build lex "monster"))
             (build member *p class *q))
    cq (build subclass *q superclass (build lex "monster"))))

(m18! (forall v5 v4)
 (&ant (p9 (class v5) (member v4))
  (p8 (class (m15 (lex monster))) (member v4)))
 (cq (p10 (subclass v5) (superclass (m15)))))

(m18!)

 CPU time : 0.00

*
;;;; HUMANS ;;;;

;; humans are animate
(describe
 (assert object (build lex "human") = human
         property *animate))

(m20! (object (m19 (lex human))) (property (m4 (lex animate))))

(m20!)

 CPU time : 0.00

*
;; For all P and Q, if they share the "son of" relation, they are both human
(describe
 (assert forall ($p $q)
   ant (build object1 *p rel (build lex "son of") object2 *q)
   cq ((build member *p class *human)
       (build member *q class *human))))

(m22! (forall v7 v6)
 (ant (p11 (object1 v6) (object2 v7) (rel (m21 (lex son of)))))
 (cq (p13 (class (m19 (lex human))) (member v7))
  (p12 (class (m19)) (member v6))))

(m22!)

 CPU time : 0.01

*

;;;; CLASS INFERENCING ON PATHS ;;;;

;;;; If P should beware some class Q which has a member R,
;;;; then P should also beware R
;;;; (NB the inverse could also be true but was not coded as it did not
;;;; directly apply)
(describe
```

```
 (assert forall ($p $q $r)
   &ant ((build agent *p
               act (build action (build lex "beware") object *q))
         (build member *r class *q))
   cq (build agent *p
             act (build action (build lex "beware")
                        object *r))) = beware-the-X)

(m24! (forall v10 v9 v8)
 (&ant (p16 (class v9) (member v10))
  (p15 (act (p14 (action (m23 (lex beware))) (object v9))) (agent v8)))
 (cq (p18 (act (p17 (action (m23)) (object v10))) (agent v8))))

(m24!)

 CPU time : 0.00

*
;; If Q is an instance of jaws and P has Q and P is a member of R,
;; then R is subclass of animal
(describe
 (assert forall ($p $q $r)
   &ant (build member *q class (build lex "jaws"))
   &ant (build whole *p part *q)
   &ant (build member *p class *r)
   cq (build subclass *r superclass *animal))
  = types-of-animals-have-jaws)

(m25! (forall v13 v12 v11)
 (&ant (p21 (class v13) (member v11)) (p20 (part v12) (whole v11))
  (p19 (class (m6 (lex jaws))) (member v12)))
 (cq (p22 (subclass v13) (superclass (m2 (lex animal))))))

(m25!)

 CPU time : 0.00

*
;; For all P that are members of the class Animal, they have Q, a member
;; of jaws
(describe
  (assert forall ($p $q)
    ant (build member *p class (build lex "animal"))
    cq ((build whole *p part *q)
          (build member *q class (build lex "jaws"))))
  = all-animals-have-jaws)

(m26! (forall v15 v14)
 (ant (p23 (class (m2 (lex animal))) (member v14)))
 (cq (p25 (class (m6 (lex jaws))) (member v15))
  (p24 (part v15) (whole v14))))

(m26!)
```

```
 CPU time : 0.00

*
;; For all P that are members of the class Q which is a subclass of
;; Animal, they have R, a member of jaws
(describe
  (assert forall ($p $q $r)
    &ant ((build member *p class *q)
             (build subclass *q superclass (build lex "animal")))
    cq ((build whole *p part *r)
          (build member *r
                   class (build lex "jaws"))))
   = all-subclasses-of-animals-have-jaws)

(m27! (forall v18 v17 v16)
 (&ant (p27 (subclass v17) (superclass (m2 (lex animal))))
  (p26 (class v17) (member v16)))
 (cq (p29 (class (m6 (lex jaws))) (member v18))
  (p28 (part v18) (whole v16))))

(m27!)

 CPU time : 0.00

*
;; things that catch are dangerous
;;;; Any agent P which can catch is also dangerous
(describe
 (assert forall $p
   ant (build agent *p
              act (build action (build lex "catch")))
   cq (build object *p
             property (build lex "dangerous"))) = catchers-are-dangerous)

(m30! (forall v19)
 (ant (p30 (act (m29 (action (m28 (lex catch))))) (agent v19)))
 (cq (p31 (object v19) (property (m14 (lex dangerous))))))

(m30!)

 CPU time : 0.00

*
;; If some part of you is dangerous, you are dangerous
;;;; For all P with part Q such that Q is dangerous,
;;;; P is, by association, dangerous
(describe
 (assert forall ($p $q)
   &ant ((build whole *p part *q)
         (build object *q property (build lex "dangerous")))
   cq (build object *p
             property (build lex "dangerous")))
   = you-are-what-youre-made-of)
```

```
(m31! (forall v21 v20)
 (&ant (p33 (object v21) (property (m14 (lex dangerous))))
  (p32 (part v21) (whole v20)))
 (cq (p34 (object v20) (property (m14)))))

(m31!)

 CPU time : 0.00

*
;; given NPa Va NPb and NPa Va NPc
;; where NPb is of unknown class and NPc is of known class
;; NPb is a type of NPc

;; whiffling is a type of flying
;;;; If P whiffles, P can be said to fly with some whiffly aspect
(describe
 (assert forall $p
   ant (build agent *p act (build action (build lex "whiffle")))
   cq (build agent *p act (build action (build lex "fly")
                                manner (build lex "whifflingly")))))

(m37! (forall v22)
 (ant (p35 (act (m33 (action (m32 (lex whiffle))))) (agent v22)))
 (cq
  (p36
   (act (m36 (action (m34 (lex fly)))
         (manner (m35 (lex whifflingly)))))
   (agent v22))))

(m37!)

 CPU time : 0.00

*
;; bad is the enemy of good
;;;; If P is the foe of Q and Q is good, then P is evil
(describe
 (assert forall ($p $q)
   &ant ((build object1 *p rel (build lex "foe of") object2 *q)
         (build object *q property (build lex "good")))
   cq (build object *p property (build lex "evil"))) = good-fights-evil)

(m41! (forall v24 v23)
 (&ant (p38 (object v24) (property (m39 (lex good))))
  (p37 (object1 v23) (object2 v24) (rel (m38 (lex foe of)))))
 (cq (p39 (object v23) (property (m40 (lex evil))))))

(m41!)

 CPU time : 0.01

*
;; The ego should always be a hero
```

```
;;;; Any P with the property "main character" is good
(describe
 (assert forall $p
   ant (build object *p property (build lex "main character"))
   cq (build object *p property (build lex "good"))) = my-side-is-right)

(m43! (forall v25)
 (ant (p40 (object v25) (property (m42 (lex main character)))))
 (cq (p41 (object v25) (property (m39 (lex good))))))

(m43!)

 CPU time : 0.00

*
;; Given P is a member of Q and Q is a subclass of R, Q has the
;; property S if that P and R also already have it

;; Given P is a member of Q and Q is a subclass of R, Q has the
;; part S if P has T, an instance of it, and R has it abstractly

;; Given P is a member of Q and P is a member of R and
;; that R has at least one subclass S, Q is also
;; a subclass of R

; CASSIE READS THE PASSAGE:
; =========================

;;;;;;;;;;;;;;;;;;;;;;;; STANZA 1 ;;;;;;;;;;;;;;;;;;;;;;;;
; ‘‘Beware the jabberwock, my son!
; The jaws that bite, the claws that catch!
; Beware the Jubjub bird, and shun
; The frumious Bandersnatch!’’
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; One should beware the jabberwock.
;;;; Something is called "jabberwock."
(describe
 (add object #TheJabberwock
      proper-name "TheJabberwock"))

(m44! (object b1) (proper-name TheJabberwock))

(m44!)

 CPU time : 0.00

*
;;;; It is a type of jabberwocky
(describe
 (add member *TheJabberwock
      class (build lex "JabberwockY") = JabberwockY))

(m46! (class (m45 (lex JabberwockY))) (member b1))
```

```
(m13! (part (m12 (lex head))) (whole (m2 (lex animal))))
(m11! (part (m10 (lex eyes))) (whole (m2)))
(m9! (part (m8 (lex claws))) (whole (m2)))
(m7! (part (m6 (lex jaws))) (whole (m2)))
(m3! (subclass (m1 (lex bird))) (superclass (m2)))

(m46! m13! m11! m9! m7! m3!)

 CPU time : 0.40

*
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Ask Cassie what "JabberwockY" means:
^(
--> defineNoun "JabberwockY")
 Definition of JabberwockY:
 Named Individuals: TheJabberwock,
nil



 CPU time : 0.09

* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;; The addressee is named George Spelvin
(describe
 (add object #addressee
      proper-name "George Spelvin"))

(m67! (object b2) (proper-name George Spelvin))

(m67!)

 CPU time : 0.00

*
;;;; The speaker is named Old Man.
(describe
 (add object #speaker
      proper-name "Old Man"))

(m68! (object b3) (proper-name Old Man))

(m68!)

 CPU time : 0.01

*
;;;; Addressee is Speaker’s son
(describe
 (add object1 *addressee
      rel (build lex "son of")
      object2 *speaker))
```

```
(m71! (class (m19 (lex human))) (member b2))
(m70! (class (m19)) (member b3))
(m69! (object1 b2) (object2 b3) (rel (m21 (lex son of))))

(m71! m70! m69!)

 CPU time : 0.03

*
;;;; Addressee should beware the jabberwock.
(describe
 (add agent *addressee
      act (build action (build lex "beware") = beware
                 object *TheJabberwock)) = beware-the-jw)

(m73! (act (m72 (action (m23 (lex beware))) (object b1))) (agent b2))

(m73!)

 CPU time : 0.02

*
;; Jaws that bite are what the jabberwock has.
;;;; Something is a type of jaws
(describe
 (add member #jaws
      class (build lex "jaws") = jaws-lex) = somethings-are-jaws)

(m74! (class (m6 (lex jaws))) (member b4))

(m74!)

 CPU time : 0.19

*
;;;; The jabberwock has these.
(describe
 (add whole *TheJabberwock
      part *jaws) = jw-has-jaws)

(m84! (part b4) (whole b1))
(p45! (part v18) (whole b1))
(m49! (subclass (m45 (lex JabberwockY))) (superclass (m2 (lex animal))))
(p29! (class (m6 (lex jaws))) (member v18))

(m84! p45! m49! p29!)

 CPU time : 0.06

*
;;;; All jaws are used for biting.
(describe
 (add forall $p
```

```
      ant (build member *p class (build lex "jaws"))
      cq (build agent *p
                act (build action (build lex "bite")))) = jaws-bite)

(m88! (act (m86 (action (m85 (lex bite))))) (agent b4))
(p93! (act (m86)) (agent v18))
(m87! (forall v28) (ant (p91 (class (m6 (lex jaws))) (member v28)))
 (cq (p92 (act (m86)) (agent v28))))
(m74! (class (m6)) (member b4))
(p29! (class (m6)) (member v18))

(m88! p93! m87! m74! p29!)

 CPU time : 0.10

*
;; Claws that catch are what the jabberwock has.
;;;; Something is of the type claws
(describe
 (add member #claws
      class (build lex "claws") = claws-lex) = somethings-are-claws)

(m101! (act (m86 (action (m85 (lex bite))))) (agent b5))
(m99! (class (m6 (lex jaws))) (member b5))
(m90! (class (m8 (lex claws))) (member b5))

(m101! m99! m90!)

 CPU time : 0.30

*
;;;; The jabberwock has these.
(describe
 (add whole *TheJabberwock
      part *claws) = jw-has-claws)

(m103! (part b5) (whole b1))
(m49! (subclass (m45 (lex JabberwockY))) (superclass (m2 (lex animal))))

(m103! m49!)

 CPU time : 0.03

*
;;;; Claws are used for catching.
(describe
 (add forall $p
      ant (build member *p class (build lex "claws"))
      cq (build agent *p
                act (build action (build lex "catch")))) = claws-catch)

(m106! (subclass (m45 (lex JabberwockY)))
 (superclass (m15 (lex monster))))
(m105! (subclass (m15)) (superclass (m2 (lex animal))))
```

```
(m104! (forall v29) (ant (p105 (class (m8 (lex claws))) (member v29)))
 (cq (p106 (act (m29 (action (m28 (lex catch))))) (agent v29))))
(m102! (act (m29)) (agent b5))
(m93! (object b5) (property (m14 (lex dangerous))))
(m90! (class (m8)) (member b5))
(p45! (part v18) (whole b1))
(m51! (class (m15)) (member b1))
(m47! (object b1) (property (m14)))
(p29! (class (m6 (lex jaws))) (member v18))

(m106! m105! m104! m102! m93! m90! p45! m51! m47! p29!)

 CPU time : 0.21

*
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Ask Cassie what "JabberwockY" means:
^(
--> defineNoun "JabberwockY")
 Definition of JabberwockY:
 Class Inclusions: monster,
 Possible Structure: jaws, jaws claws,
 Possible Actions: bite,
 Possible Properties: dangerous,
nil



 CPU time : 0.17

* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; One should beware the Jubjub bird.
;;;; There is a type of bird called Jubjub.
(describe
 (add subclass (build lex "Jubjub bird") = Jubjub
      superclass *bird))

(m117! (subclass (m116 (lex Jubjub bird))) (superclass (m1 (lex bird))))

(m117!)

 CPU time : 0.01

*
;;;; The addressee should beware all Jubjub-birds.
(describe
 (add agent *addressee
      act (build action (find lex "beware")
                 object *Jubjub)))

(m119!
 (act (m118 (action (m23 (lex beware)))
       (object (m116 (lex Jubjub bird)))))
```

```
 (agent b2))

(m119!)

 CPU time : 0.03

*
;; One should shun the frumious bandersnatch.
;;;; Something is a bandersnatch.
;;;; It is frumious.
;;;; The addressee should shun the bandersnatch.
;;;;;; Skipped because nothing interesting comes of the bandersnatch ;;;;;;

;;;;;;;;;;;;;;;;;;;;;; STANZA 2 ;;;;;;;;;;;;;;;;;;;;;;
; He took his vorpal sword in hand:
; Long time the manxome foe he sought --
; So rested he by the Tumtum tree,
; And stood awhile in thought.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The addressee is our main character
(describe
 (add object *addressee
         property (build lex "main character")) = ego-is-main-character)

(m121! (object b2) (property (m39 (lex good))))
(m120! (object b2) (property (m42 (lex main character))))

(m121! m120!)

 CPU time : 0.10

*
;;;; A vorpal sword is an instance of a sword
(describe
 (add member (build lex "vorpal sword") = vorpal-sword
      class (build lex "sword") = sword))

(m137! (act (m86 (action (m85 (lex bite)))))
 (agent (m122 (lex vorpal sword))))
(m133! (class (m6 (lex jaws))) (member (m122)))
(m124! (class (m123 (lex sword))) (member (m122)))

(m137! m133! m124!)

 CPU time : 0.45

*
;;;;;; "I am afraid I can't explain 'vorpal blade' for you..."
;;;;;;  Dodgson's Explanation to Maud Standen
;;;;;;  --Letter, December 1877
;;;;;;  http://www76.pair.com/keithlim/jabberwocky/poem/maudstanden.html
;;;; It has the property of being vorpal
(describe
```

```
 (add object *vorpal-sword
      property (build lex "vorpal")))

(m141! (object (m122 (lex vorpal sword)))
 (property (m140 (lex vorpal))))

(m141!)

 CPU time : 0.02

*
;;;; It is the addressee's.
(describe
(add object *vorpal-sword
        rel *sword
        possessor *addressee))

(m142! (object (m122 (lex vorpal sword))) (possessor b2)
 (rel (m123 (lex sword))))

(m142!)

 CPU time : 0.02

*
;; The addressee sought the jabberwock.
(describe
 (add agent *addressee
      act (build action (build lex "seek")
                 object *TheJabberwock)))

(m145! (act (m144 (action (m143 (lex seek))) (object b1))) (agent b2))

(m145!)

 CPU time : 0.01

*
;; The jabberwock was a manxome foe.
;;;; The jabberwock is the foe of the addressee.
(describe
(add object1 *TheJabberwock
        rel (build lex "foe of")
        object2 *addressee) = jw-is-foe-of-ego)

(m147! (object b1) (property (m40 (lex evil))))
(m146! (object1 b1) (object2 b2) (rel (m38 (lex foe of))))

(m147! m146!)

 CPU time : 0.09

*
;;;;;;;;;;;;;;;;;;;;;;; STANZA 3 ;;;;;;;;;;;;;;;;;;;;;;;
```

```
; And, as in uffish thought he stood,
; The jabberwock, with eyes of flame,
; Came whiffling through the tulgey wood,
; And burbled as it came!
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The jabberwock had eyes of flame.
;;;; Something is a type of eyes.
(describe
(add member #eyes
        class (build lex "eyes") = eyes-lex) = somethings-are-eyes)

(m161! (act (m86 (action (m85 (lex bite))))) (agent b6))
(m157! (class (m6 (lex jaws))) (member b6))
(m148! (class (m10 (lex eyes))) (member b6))

(m161! m157! m148!)

 CPU time : 0.52

*
;;;; The jabberwock has these.
(describe
(add whole *TheJabberwock
        part *eyes) = jw-has-eyes)

(m164! (part b6) (whole b1))
(m105! (subclass (m15 (lex monster))) (superclass (m2 (lex animal))))
(m49! (subclass (m45 (lex JabberwockY))) (superclass (m2)))

(m164! m105! m49!)

 CPU time : 0.04

*
;;;; They are made of flame.
(describe
(add whole *eyes
        part (build lex "flame") = flame))

(m166! (part (m165 (lex flame))) (whole b6))

(m166!)

 CPU time : 0.03

*
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Ask Cassie what "JabberwockY" means:
^(
--> defineNoun "JabberwockY")
 Definition of JabberwockY:
 Class Inclusions: monster,
 Possible Structure: jaws, jaws claws, jaws eyes,
```

```
 Possible Actions: bite,
 Possible Properties: evil, dangerous, foe of human,
nil



 CPU time : 0.16

* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The jabberwock came whiffling.
;;;; The jabberwock whiffled
(describe
(add agent *TheJabberwock
        act (build action (build lex "whiffle"))))

(m65!
 (act (m36 (action (m34 (lex fly))) (manner (m35 (lex whifflingly)))))
 (agent b1))
(m63! (act (m33 (action (m32 (lex whiffle))))) (agent b1))

(m65! m63!)

 CPU time : 0.05

*
;; The jabberwock burbled as it came.
;;;; The jabberwock performed the action burble
(describe
(add agent *TheJabberwock
        act (build action (build lex "burble"))))

(m171! (act (m170 (action (m169 (lex burble))))) (agent b1))

(m171!)

 CPU time : 0.04

*
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Ask Cassie what "JabberwockY" means:
^(
--> defineNoun "JabberwockY")
 Definition of JabberwockY:
 Class Inclusions: monster,
 Possible Structure: jaws, jaws claws, jaws eyes,
 Possible Actions: whiffle, fly, bite, burble,
 Possible Properties: evil, dangerous, foe of human,
nil



 CPU time : 0.15
```

```
* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;; STANZA 4 ;;;;;;;;;;;;;;;;;;;;;;;;
; One, two! One, two! And through and through
; The vorpal blade went snicker-snack!
; He left it dead, and with its head
; He went galumphing back.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; The addressee killed the jabberwock with his sword.
;;;; The addressee killed the jabberwock
(describe
(add agent *addressee
        act (build action (build lex "kill")
                          object *TheJabberwock)) = Morte-de-Jabberwock)

(m174! (act (m173 (action (m172 (lex kill))) (object b1))) (agent b2))

(m174!)

 CPU time : 0.02

*
;;;; The jabberwock is dead.
(describe
(add object *TheJabberwock
        state *dead))

(m175! (object b1))

(m175!)

 CPU time : 0.03

*
;; The hero went galumphing home with the jabberwock’s head.
;;;; Something is a head
(describe
(add member #head
        class (build lex "head")) = something-is-a-head)

(m189! (act (m86 (action (m85 (lex bite))))) (agent b7))
(m185! (class (m6 (lex jaws))) (member b7))
(m176! (class (m12 (lex head))) (member b7))

(m189! m185! m176!)

 CPU time : 0.60

*
;;;; The jabberwock has a head.
(describe
(add whole *TheJabberwock
        part *head))
```

```
(m192! (part b7) (whole b1))
(m105! (subclass (m15 (lex monster))) (superclass (m2 (lex animal))))
(m49! (subclass (m45 (lex JabberwockY))) (superclass (m2)))

(m192! m105! m49!)

 CPU time : 0.06

*
;;;; The addressee took its head.
(describe
(add agent *addressee
        act (build action (build lex "take")
                        object *head)))

(m195! (act (m194 (action (m193 (lex take))) (object b7))) (agent b2))

(m195!)

 CPU time : 0.02

*
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Ask Cassie what "JabberwockY" means:
^(
--> defineNoun "JabberwockY")
 Definition of JabberwockY:
 Class Inclusions: monster,
 Possible Structure: jaws, jaws claws, jaws eyes, jaws head,
 Possible Actions: whiffle, fly, bite, burble,
 Possible Properties: evil, dangerous, foe of human,
nil



 CPU time : 0.16

* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;; The addressee returned home with the head
;; The addressee returned home
(describe
(add agent *addressee
        act (build
                action (build lex "return")
                object #home)))

(m198! (act (m197 (action (m196 (lex return))) (object b8))) (agent b2))

(m198!)

 CPU time : 0.02
```

```
*
;;;;;;;;;;;;;;;;;;;;;;; STANZA 5 ;;;;;;;;;;;;;;;;;;;;;;;
; ‘‘And, has thou slain the jabberwock?
; Come to my arms, my beamish boy!
; O frabjous day! Callooh! Callay!’’
; He chortled in his joy.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Killing the jabberwock made the old man happy.
;;;; The death of the jabberwock pleased the old man
(describe
(add agent *Morte-de-Jabberwock
        act (build action (build lex "please")
                        object *old-man)))

(m201! (act (m200 (action (m199 (lex please)))))
 (agent
  (m174! (act (m173 (action (m172 (lex kill))) (object b1)))
   (agent b2))))

(m201!)

 CPU time : 0.02

*
(describe *nodes)

(m201! (act (m200 (action (m199 (lex please)))))
 (agent
  (m174! (act (m173 (action (m172 (lex kill))) (object b1)))
   (agent b2))))
(m198! (act (m197 (action (m196 (lex return))) (object b8))) (agent b2))
(m195! (act (m194 (action (m193 (lex take))) (object b7))) (agent b2))
(m192! (part b7) (whole b1))
(m189! (act (m86 (action (m85 (lex bite))))) (agent b7))
(m185! (class (m6 (lex jaws))) (member b7))
(m176! (class (m12 (lex head))) (member b7))
(m175! (object b1))
(m171! (act (m170 (action (m169 (lex burble))))) (agent b1))
(m166! (part (m165 (lex flame))) (whole b6))
(m164! (part b6) (whole b1))
(m161! (act (m86)) (agent b6))
(m157! (class (m6)) (member b6))
(m148! (class (m10 (lex eyes))) (member b6))
(m147! (object b1) (property (m40 (lex evil))))
(m146! (object1 b1) (object2 b2) (rel (m38 (lex foe of))))
(m145! (act (m144 (action (m143 (lex seek))) (object b1))) (agent b2))
(m142! (object (m122 (lex vorpal sword))) (possessor b2)
 (rel (m123 (lex sword))))
(m141! (object (m122)) (property (m140 (lex vorpal))))
(m137! (act (m86)) (agent (m122)))
(m133! (class (m6)) (member (m122)))
(m124! (class (m123)) (member (m122)))
(m121! (object b2) (property (m39 (lex good))))
```

```
(m120! (object b2) (property (m42 (lex main character))))
(m119!
 (act (m118 (action (m23 (lex beware)))
       (object (m116 (lex Jubjub bird)))))
 (agent b2))
(m117! (subclass (m116)) (superclass (m1 (lex bird))))
(m106! (subclass (m45 (lex JabberwockY)))
 (superclass (m15 (lex monster))))
(m105! (subclass (m15)) (superclass (m2 (lex animal))))
(m104! (forall v29) (ant (p105 (class (m8 (lex claws))) (member v29)))
 (cq (p106 (act (m29 (action (m28 (lex catch))))) (agent v29))))
(m103! (part b5) (whole b1))
(m102! (act (m29)) (agent b5))
(m101! (act (m86)) (agent b5))
(m99! (class (m6)) (member b5))
(m93! (object b5) (property (m14 (lex dangerous))))
(m90! (class (m8)) (member b5))
(m89! (act (m86)) (agent b1))
(m88! (act (m86)) (agent b4))
(p93! (act (m86)) (agent v18))
(m87! (forall v28) (ant (p91 (class (m6)) (member v28)))
 (cq (p92 (act (m86)) (agent v28))))
(m84! (part b4) (whole b1))
(m74! (class (m6)) (member b4))
(m73! (act (m72 (action (m23)) (object b1))) (agent b2))
(m71! (class (m19 (lex human))) (member b2))
(m70! (class (m19)) (member b3))
(m69! (object1 b2) (object2 b3) (rel (m21 (lex son of))))
(m68! (object b3) (proper-name Old Man))
(m67! (object b2) (proper-name George Spelvin))
(m65!
 (act (m36 (action (m34 (lex fly))) (manner (m35 (lex whifflingly)))))
 (agent b1))
(m63! (act (m33 (action (m32 (lex whiffle))))) (agent b1))
(m59! (class (m6)) (member b1))
(p45! (part v18) (whole b1))
(m51! (class (m15)) (member b1))
(m49! (subclass (m45)) (superclass (m2)))
(m47! (object b1) (property (m14)))
(m46! (class (m45)) (member b1))
(m44! (object b1) (proper-name TheJabberwock))
(m43! (forall v25) (ant (p40 (object v25) (property (m42))))
 (cq (p41 (object v25) (property (m39)))))
(m41! (forall v24 v23)
 (&ant (p38 (object v24) (property (m39)))
  (p37 (object1 v23) (object2 v24) (rel (m38))))
 (cq (p39 (object v23) (property (m40)))))
(m37! (forall v22) (ant (p35 (act (m33)) (agent v22)))
 (cq (p36 (act (m36)) (agent v22))))
(m31! (forall v21 v20)
 (&ant (p33 (object v21) (property (m14)))
  (p32 (part v21) (whole v20)))
 (cq (p34 (object v20) (property (m14)))))
(m30! (forall v19) (ant (p30 (act (m29)) (agent v19)))
```

```
 (cq (p31 (object v19) (property (m14)))))
(m27! (forall v18 v17 v16)
 (&ant (p27 (subclass v17) (superclass (m2)))
  (p26 (class v17) (member v16)))
 (cq (p29! (class (m6)) (member v18)) (p28 (part v18) (whole v16))))
(m26! (forall v15 v14) (ant (p23 (class (m2)) (member v14)))
 (cq (p25 (class (m6)) (member v15)) (p24 (part v15) (whole v14))))
(m25! (forall v13 v12 v11)
 (&ant (p21 (class v13) (member v11)) (p20 (part v12) (whole v11))
  (p19 (class (m6)) (member v12)))
 (cq (p22 (subclass v13) (superclass (m2)))))
(m24! (forall v10 v9 v8)
 (&ant (p16 (class v9) (member v10))
  (p15 (act (p14 (action (m23)) (object v9))) (agent v8)))
 (cq (p18 (act (p17 (action (m23)) (object v10))) (agent v8))))
(m22! (forall v7 v6) (ant (p11 (object1 v6) (object2 v7) (rel (m21))))
 (cq (p13 (class (m19)) (member v7)) (p12 (class (m19)) (member v6))))
(m20! (object (m19)) (property (m4 (lex animate))))
(m18! (forall v5 v4)
 (&ant (p9 (class v5) (member v4)) (p8 (class (m15)) (member v4)))
 (cq (p10 (subclass v5) (superclass (m15)))))
(m17! (forall v3 v2)
 (&ant (p6 (object v2) (property (m14)))
  (p5 (subclass v3) (superclass (m2))) (p4 (class v3) (member v2)))
 (cq (p7 (class (m15)) (member v2))))
(m16! (forall v1)
 (&ant (p2 (object v1) (property (m14))) (p1 (class (m2)) (member v1)))
 (cq (p3 (class (m15)) (member v1))))
(m13! (part (m12)) (whole (m2)))
(m11! (part (m10)) (whole (m2)))
(m9! (part (m8)) (whole (m2)))
(m7! (part (m6)) (whole (m2)))
(m5! (object (m2)) (property (m4)))
(m3! (subclass (m1)) (superclass (m2)))

(m201! m200 m199 please m198! m197 b8 m196 return m195! m194 m193 take
 m192! m189! m185! m176! b7 m175! m174! m173 m172 kill m171! m170 m169
 burble m166! m165 flame m164! m161! m157! m148! b6 m147! m146! m145!
 m144 m143 seek m142! m141! m140 vorpal m137! m133! m124! m123 sword
 m122 vorpal sword m121! m120! m119! m118 m117! m116 Jubjub bird m106!
 m105! m104! p106 p105 v29 m103! m102! m101! m99! m93! m90! b5 m89!
 m88! p93! m87! p92 m86 m85 bite p91 v28 m84! m74! b4 m73! m72 m71!
 m70! m69! m68! Old Man b3 m67! George Spelvin b2 m65! m63! m59! p45!
 m51! m49! m47! m46! m45 JabberwockY m44! TheJabberwock b1 m43! p41 p40
 m42 main character v25 m41! p39 m40 evil p38 m39 good p37 m38 foe of
 v24 v23 m37! p36 m36 m35 whifflingly m34 fly p35 m33 m32 whiffle v22
 m31! p34 p33 p32 v21 v20 m30! p31 p30 m29 m28 catch v19 m27! p29! p28
 p27 p26 v18 v17 v16 m26! p25 p24 p23 v15 v14 m25! p22 p21 p20 p19 v13
 v12 v11 m24! p18 p17 p16 p15 p14 m23 beware v10 v9 v8 m22! p13 p12 p11
 m21 son of v7 v6 m20! m19 human m18! p10 p9 p8 v5 v4 m17! p7 p6 p5 p4
 v3 v2 m16! p3 m15 monster p2 m14 dangerous p1 v1 m13! m12 head m11!
 m10 eyes m9! m8 claws m7! m6 jaws m5! m4 animate m3! m2 animal m1 bird)

 CPU time : 0.04
```

```
*
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Ask Cassie what "JabberwockY" means:
^(
--> defineNoun "JabberwockY")
 Definition of JabberwockY:
 Class Inclusions: monster,
 Possible Structure: jaws, jaws claws, jaws eyes, jaws head,
 Possible Actions: whiffle, fly, bite, burble,
 Possible Properties: evil, dangerous, foe of human,
nil



 CPU time : 0.16

* ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

End of /home/lingrad/pmheider/courses/CSE727/word/jabberwock.demo demonstration.


 CPU time : 5.31

* (lisp)
"End of SNePS"
cl-user(3): :exit
; Exiting
pollux {~/courses/CSE727/word} > exit
exit

script done on Thu May 10 03:33:06 2007
```

References

Berwick, R. C. (1989). Learning word meanings from examples. In Waltz, D. L., editor, *Semantic Structures: Advances in Natural Language Processing*, pages 89–124. Lawrence Erlbaum Associates, Hillsdale, NJ.

Carroll, L. (1872). *Through the Looking-Glass and What Alice Found There*. Macmillan.

Clarke, D. F. and Nation, I. S. P. (1980). Guessing the meanings of words from context: Strategy and techniques. *System*, 8:211–220.

Nagy, W. E. and Anderson, R. C. (1984). How many words are there in printed school english? *Reading Research Quarterly*, 19(3):304–330.

Rapaport, W. J. and Kibby, M. W. (2006). Contextual vocabulary acquisition as computational philosophy and as philosophical computation. In Press.

Shapiro, S. C., Rapaport, W. J., Cho, S.-H., Choi, J., Feit, E., Haller, S., Kankiewicz, J., and Kuman, D. (1996). *A Dictionary of SNePS Case Frames*. Draft: http://www.cse.buffalo.edu/sneps/Manuals/dictionary.pdf.