Acquiring the Meaning of "hummock" Using Sentence Context and SNePSLOG

Scott Settembre
December 02, 2006
CSE663: Advanced Topics in Knowledge Representation and Reasoning

Abstract

We represent a passage from a novel in the knowledge representation language "SNePSLOG" to allow a computer to manipulate this information and draw inferences about the meaning of a specific word. To accomplish such a task, choices needed to be made about what information needs to be represented, how it can be represented within the SNePSLOG language, and also as to the level of detail that the representation needs to encapsulate. Using techniques from the Contextual Vocabulary Acquisition (CVA) project, in particular a strategy of verbal "think-aloud" protocols and a "noun-algorithm" which SNePSLOG can utilize, we can gain insight into what information is necessary to supply to a reasoning agent so that it can derive the meaning of the word "hummock". By both directly encoding the sentence itself and also by defining some background knowledge (drawn from subjects interviewed during the verbal protocols) we show that we can infer valuable information about the meaning of the unknown word. We decided to adopt a strategy of representing a "memory image" and then wrote a rule to draw a parallel between the scenes described and then finally we inferred similarities between the unknown word and an intensional object that was represented. The potential of techniques like these in establishing a general method which can derive the meanings of words with a minimal set of background knowledge could revolutionize both the way we teach our children as well as in the design of self-learning programs.

1. The CVA Project

The Contextual Vocabulary Acquisition (CVA) project encompasses two areas of research (Rapaport and Kibby 2005). The first area is concerned with developing a methodology which can be utilized by students of all grade levels to improve the process of learning new words. By examining how experienced readers generate initial meanings for unknown words and then continuously refine that meaning as more information surrounding the word is read, the CVA project hopes to develop a theory of how a cognitive agent refines its comprehension of unknown words.

The second area of research, which is the primary focus of this paper, is the use of a computational framework to implement and prove the methodology to generate meanings of new words. To this end, a series of algorithms have already been implemented in LISP for use in the "SNePS" system. There currently exists a noun-algorithm and a verb-algorithm, which implements respectively both noun and verb context acquisition strategies, developed by Rapaport and Ehrlich (2000). Our primary focus will be on an unknown noun in this paper, and so we will be using and commenting on the existing noun-algorithm.

SNePS is a semantic network system developed by first by Dr. Stuart Shapiro and further refined by the SNeRG research group at the University of Buffalo (1999-2006). It is both a knowledge representation system and a reasoning system. There are two language-based interfaces to the SNePS system, one is called SNePSUL and was used

quite extensively for most of the CVA programming projects heretofore, and the other is called SNePSLOG (Shapiro 2004:71-75), which we will be using for our project.

2. The Paragraph and Unknown Word

We chose to select a passage from "A Box of Matches" by Nicholson Baker and implement the noun-algorithm on the word "hummock". The passage reads as follows:

> *"Now the sun had risen enough to clear the dense **hummock** of forest across the creek, and thus sunlight was striking and warming the leaves on this particular tree for the first time since they'd frozen."*
>
> (Baker, Nicholson (2003), *A Box of Matches* (New York: Random House): 36.)

The American Heritage Dictionary has three meanings for the word "hummock":

1. A low mound or ridge of earth; a knoll.
2. A tract of forested land that rises above an adjacent marsh in the southern United States.
3. A ridge or hill of ice in an ice field.

3. Verbal "Think-Aloud" Protocols

One of the decisions encountered when attempting to represent something is the choice of what knowledge to code and what knowledge to ignore. All knowledge about a word may be relevant and helpful, but what background knowledge is explicitly necessary to form a good meaning for an unknown word.

A technique commonly used in the CVA project is called the "think-aloud" protocols. By observing how a person goes about forming an initial meaning for an unknown word and then how they may continue to refine this meaning, we can extract what information might be essential to forming an adequate definition. We chose to interview 3 subjects, each of varying verbal abilities, to see what strategies they use as well as perhaps what strategies to avoid.

3.1. Subject 1

The first subject is a 29 year old female with a BS in Mathematics and Computer Science. She read the sentence and declared quite specifically that she felt the word meant "cluster." She said that the meaning was simple because "it fit" into the sentence perfectly, as they had heard the phrase "dense forest" before and that "a forest is trees" and therefore "'dense cluster of trees' fit in the place of 'dense hummock of forest' easily."

We then asked if she considered any other part of the sentence and to read it again before we gave them the dictionary definition. She read the sentence again and decided that “cluster of forest” didn’t make as much sense as it did the first time they read it. Reading carefully again and again, she decided that “bunch” would probably work better than “cluster” since she envisioned a “series of pine trees that were very tall and very close together.”

We asked if she felt that the “height” of the forest was important to the meaning of the word. She pointed out that it probably does because the sentence describes the sun “clearing” or getting higher than the trees so that the sunlight could reach the leaves of a smaller tree.

3.2. Subject 2

The second subject is a 59 year old female with a MS in teaching who has taught elementary school for over 15 years. She initially felt that the word meant “impassable.” When we asked why, she said “because it was dense.” We asked her to take another look, since the word “dense” was already there, and that they should also pay attention to the latter part of the sentence.

By implying that she should read more into the sentence, she assumed she was incorrect and came up with a second attempt, but this time using context from the latter half of the sentence. The second guess was “wall”, as in “dense wall of forest”, with the

reasoning that “the sunlight had to get over the wall of trees in order to hit this [other] tree.”

3.3. Subject 3

The third subject is a 59 year old male with a MS in teaching who has taught high school and ran a resource room for reading comprehension for over 30 years. In addition, we personally know that he has an extensive prior vocabulary, so we felt that he would probably know the word. The subject, after reading the sentence, gave two options for what he felt the word meant. The first option was “the reverse of a glade, like an island of trees, instead of a clearing of trees.” The second option was “tree ridge like the trees that line the palisades in New Jersey.”

In the first option, he determined the “reverse of a glade” because in farmland in New York, sometimes there is a small island of trees that stick out in the middle of old farmland. These trees are let to grow and become tall and wide. This would be enough to block out the sunlight for some smaller tree nearby.

For the second option, he felt that a series of trees that line a palisade could also be able to block another tree that was “across a river”. In the sentence, although he did not state it, this image may have been constructed in his mind from the “across the creek” phrase. We neglected to ask, but it seemed like he was constructing two different

possible images in his mind or at least “matching” two different possible images of scenes he had seen before, based on the description.

Though he “matched” an image he had of the Palisades in New Jersey, there may be another factor that he used that considered the “function” of the scene. That is, he constructed and then tried to match another scene from his memory where the “function” of the blocking of sunlight had occurred. As he would drive to work in the early morning, the line of trees on the palisades would block the sun from reaching his eyes, and when the sun would clear this line of trees, the sun would be in his eyes. This idea of using memory to match the functionality of a scene and then draw parallels between each of the scenes implies a set of inferences of rules that may be able to be used to our advantage.

3.4. Representation plan

We were impressed by the third subject’s use of prior images to arrive at his initial definitions. We then decided that perhaps the use of imagery would be a novel way to help the noun-algorithm arrive at similar conclusions. Our plan was to represent images, in this case memories from Subject 3, in the most direct way we could and represent the sentence in a similar fashion. We would then explore what background knowledge would be needed to allow parallels or similarities to be draw between the memories and sentence, and then implement that as well.

4. Case Frame Selection and Definition

Since our representation language was SNePSLOG, and SNePSLOG had not been used prior to this point with the noun-algorithm, we were also given the task to create "case frames" that would be both compatible with the noun-algorithm's existing case frames (Napieralski 2003) and also be readable within SNePSLOG.

A "case frame" is a syntactically agreed upon way to organize information so that two different agents or algorithms can semantically interpret the meaning of the arranged information in a similar way. In this case, we are the knowledge engineers who will represent the sentence in a way so that the noun-algorithm can interpret it the way we intend. We need to adhere to agreed upon node/arc arrangements so that the noun-algorithm can perform its inferences.

Most of the case frames were drawn from an informal document by Rapaport (CCS663 Listserv). We further refined the case frame definitions to encompass a consistent naming and readability to them, with the exception of three which are common to the literature of knowledge representation (Isa, AKO, and postfix "–wrt" as in Act-wrt), and enhanced the documentation to include examples for future students of this work. This documentation is in the code situated by the defined frame itself, complete with a textual diagram and examples of the use in a SNePSLOG code fragment.

5. Representing the Sentence

We began by deciding to represent the sentence as closely as possible to what the sentence is actually stating without rephrasing it. As you will see, while doing this, we found a lot of potential background information that could be represented, but does not have much to do with the meaning of “hummock”. Some of this information we moved to perhaps be implemented in the background information section of the code, leaving the important and more directly related information.

We broke the sentence up into three sections and will go through each for clarity. Section one represented the phrase "{Now} the sun had risen {enough} to clear the dense hummock of forest across the creek,”. The words enclosed in brackets are not directly represented in the code; instead their meaning is implied or moved to the background information section of the code.

First we represent “the sun had risen”:

```
;; the sun had risen
Isa(Concept-called("hummock"),hummock1).
Isa(Concept-called("sun"),sun1).
Does(The-action(Concept-called("risen")),sun1).
```

You will notice that we enclose many of the words in quotations and in a proposition “Concept-called”. For future students of this work, it is necessary to create an arc named “lex” between the intensional representation (a unique node representing the actual object in the world) and the symbol from the English lexicon that represents that object. [See the case-frame for “Concept-called” in the code] Though it is needed for the noun-algorithm to communicate its results, it is also a good way to separate the difference between a word and what the word represents.

In this case, we create an intensional representation “hummock1” which will represent the hummock of the sentence, as well as creating “sun1” which will represent the sun of the sentence. We then assert that sun1 does an action called “risen”.

Next we represent “enough to clear the dense hummock”:

```
;; {enough} to clear [the dense hummock]
Does(Act-wrt(hummock1,Concept-called("clear")),sun1).
```

The previous intensional representation of the “sun”, sun1, is now used to relate directly to the unknown word “hummock”, or hummock1. We use the case-frame Does to relate an action “clear” that sun1 does with respect to (wrt) hummock1.

```
;; the dense hummock
Is-property(Concept-called("dense"),hummock1).
```

We create another relationship between the unknown word “hummock”, or hummock1, and a property called “dense”. We, of course, are using our knowledge of the word dense in this case, to understand that it is a property. As we have seen, this is not always obvious, as can be seen from the standpoint of subject 2 in the verbal protocols. We also saw that subject 1 used this word to help her arrive at a meaning, albeit incorrect, but yet the importance of adjectives like “dense” may have potential to help understand a word that appear in similar constructed phrases instead of similar memories.

Astute readers who peruse this paper will note that we have neglected to represent “enough”. Now admittedly, the concept of “enough” can entail a great deal of representation. We shy away from representing this because we did not feel it was necessary to describe what “enough” means when it has little to do with the word hummock, or at least we use our discretion as a knowledge engineer to limit the resolution of the information needed to accurately get the job done. In any case, we move the word enough into the amorphous “background knowledge” section for future enhancement.

Next we represent “the dense hummock of forest across the creek”:

```
;; hummock of forest
Isa(Concept-called("forest"),forest1).
Is-a-part-of(forest1,hummock1).
```

```
;; [the dense hummock of forest] across the creek
Isa(Concept-called("creek"),creek1).
Has-relation(Concept-called("across"),creek1,hummock1).
```

A straightforward reading of the code shows similar intensional representations for forest1 and creek1. A new case-frame being used is “Has-relation” and the code merely states that there is a relationship of “across” between the creek1 and hummock1. The interesting issue here is the case frame “Is-a-part-of” and will relate later directly to the use of imagery in drawing inferences, which we will discuss later in the paper.

Section two of representing the sentence is concerned with the phrase “{and thus} sunlight was striking and warming the leaves on this {particular} tree”.

```
;; sunlight was striking and warming the leaves
Isa(Concept-called("sunlight"),sunlight1).
Is-a-part-of(sun1,sunlight1).
Does(Act-wrt(leaves1,Concept-called("striking")),sunlight1)
Does(Act-wrt(leaves1,Concept-called("warming")),sunlight1).

;; the leaves on this {particular} tree
Isa(Concept-called("tree"),tree1).
Isa(Concept-called("leaves"),leaves1).
Is-a-part-of(tree1,leaves1).
```

Here we further literally represent the sentence by intensionally representing sunlight, leaves, and tree using sunlight1, leaves1, and tree1. Note that we are not proposing here the idea of collections as objects of type leaf are leaves; instead we treat the idea of leaves as one part of a tree. This level of detail is another act of discretion on the knowledge engineer's part in a sense limiting what we can reason about should we extend or use this representation at a later time.

We also are representing the act of "striking" and "warming" and make note of it here because of the use of these acts in our background knowledge representation of memories which will then be used to infer things about hummock.

Section three of the literal representation of the sentence is concerned with "{for the first time since} they'd (they had) frozen."

```
;; they'd (they had) frozen
Is-property(Concept-called("frozen"),leaves1).
```

We represent this here for completeness, but does not have much to do with inferences that will be drawn. This section is declared separately because we had initially felt that the phrase "for the first time since" would be important to the understanding of hummock. This phrase and other words or phrases like it found in brackets are discussed a little more in the comments of the background section of the code, however, we will not be discussing it as it has no direct relationship to the unknown word. We do not dismiss

its importance though, especially if this work was to be extended, since an implementation of temporal acts and relationships would be very useful.

6. Initial Run

Here we show the results of applying the noun-algorithm to the representation we detailed in Section 5. It is clear that there is not enough information explicitly described in the passage to produce an adequate definition like that from the third subject interviewed in the verbal “talk aloud” protocols.

```
; Ask Cassie what "hummock" means:
^(snepsul::defineNoun "hummock")
 Definition of hummock:
 Actions performed on a hummock: sun clear,
 Possible Properties: dense, across creek, nil
```

We can see, however, that there is a potential to use imagery and similarity to further enhance the definition. For example, we can see that the action of “sun clear” performed on “hummock” is similar to the way Subject 3 matched a similar scene in his memory. He described the sun clearing the Palisades in NJ which contained a row of

trees, and since the palisades is a hill-like object perhaps we will get a similar match if a parallel can be inferred between the two scenes.

7. Background Knowledge

As the results of the initial run were encouraging, we decided to pursue representing a memory that the reasoning agent could have been introduced to. We decided to represent the image of the sun clearing a palisade of trees and hitting a subject's eyes. We will do this by encoding the sentence "The sun cleared the palisade of tree and the sunlight struck the subject's eyes."

```
;; Subject 3's: Memory image
;; The sun cleared the NJ Palisades, sunlight struck
;; subject3's eye
Isa(Concept-called("hill-like object"),hillobject1).
Isa(Concept-called("NJ Palisades"),palisade1).
Isa(hillobject1,palisade1).
Is-a-part-of(subject3,eyes3).
Is-a-part-of(sun1,sunlight1).
Does(Act-wrt(palisade1,Concept-called("clear")),sun1).
Does(Act-wrt(eyes3,Concept-called("striking")),sunlight1).
```

We introduce here a class called "hill-like object" which will represent an object in the world that has a hill shape. It would have been ideal to be able to use the case-frame AKO in the statement `Isa(hillobject1,palisade1)`, but unfortunately

we could not get our code to work with the noun-algorithm using the subclass/superclass arcs, so instead we decided to use Isa to also represent membership in the superclass directly. This may be a failure on our part, but we did try to directly assert this relationship so that it would fall under the algorithm step of “list direct class inclusions of N & ownership of Ns” (Rapaport and Elrich 2000:19) and we had no success most likely due to our own misinterpretation of what the algorithm is doing.

Now to examine the scene matching rule:

```
;; Rules to match this similar scene:
;; If the sun1 clears an object1, and sunlight1 strikes
;; object2, then it is possible that a similar scene like
;; this may have the object in the object1 position may be
;; of the same class as object1.
all(x,y,z)(
 {
   Does(Act-wrt(x,Concept-called("clear")),sun1),
   Does(Act-wrt(y,Concept-called("striking")),sunlight1),
   Isa(z,x)
 }
 &=>
    all(h,j)(
    {
      Does(Act-wrt(h,Concept-called("clear")),sun1),
      Does(Act-wrt(j,Concept-called("striking")),sunlight1)
    }
    &=>
      Isa(z,h)
    )
)!
```

For the scene matching rule, we decided to match quite literally the idea of "clear" and "striking" in a scene. Although this could cover many a case, from flashlights to car headlights, there may be a more generalized rule that could be used for varied arrangements of actions. Here, however, we focus on the "sun" and "sunlight" specifically, in a scene matching rule that would be applicable to any instance where the sun would clear an object and then the sunlight could hit another object. Should this specialized series of actions occur, then there may be a common superclass shared between the objects being "cleared" in both scenes.

Ideally, we should add additional rule clauses that signify that this is happening in the wilderness perhaps, enforcing similarity between context as well. As you can see from the code in Appendix B, we attempted to implement just such an exception.

```
;Isa(Concept-called("building-like
object"),buildingobject1).
;Isa(Concept-called("Empire State Building"),building1).
;Isa(buildingobject1,building1).
;Is-a-part-of(subject3,eyes3).
;Is-a-part-of(sun1,sunlight1).
;Does(Act-wrt(building1,Concept-called("clear")),sun1).
;Does(Act-wrt(eyes3,Concept-called("striking")),sunlight1).
```

Though this code is commented out for the final run, when run in test runs the noun-algorithm does put the "building-like object" in the class-inclusion portion of the definition. This shows the shortcoming in the scene matching rule, however, additional

assertions can be added to mitigate these negative conclusions. For example, one additional statement can show that a building-like object cannot be part of a forest, as in the following statement:

```
;; a building-like object cannot be "of forest"
;~Is-a-part-of(forest1,buildingobject1).
```

When this is run, a contradiction occurs, thus showing that we need to further refine the scene matching rule to include the common sense fact that properties or relationships of objects that are hypothesized to be similar, need to be consistent. The lack of consistency causes the contradiction, since “hummock” is known to be part of the forest, and so how can it also be a building-like object when building-like objects cannot be part of the forest. We attempted to implement a conditional to correct this, in a sense implementing a default, but unfortunately ran into trouble trying to represent the clause if a “part-of” relationship is NOT asserted to NOT be true (as in the code statement above), then it can be assumed to be true.

As we see, from our strategy of drawing parallels between scenes, we seem not to need many of the more common sense rules that other words may use. By using similarities between scenes, rules that we had planned to implement, like transitivity with `Is-a-part-of` or `superclass-subclass` define paths seemed not to be necessary to get an adequate result.

8. Final Run

Here is the resultant output from the noun-algorithm in the final run which can be seen in Appendix A. We can see that the definition now includes something quite like the actual definition of “hummock”. A “hill-like object” is inferred by using the scene matching rule.

```
; Ask Cassie what "hummock" means:
^(snepsul::defineNoun "hummock")
 Definition of hummock:
 Possible Class Inclusions: hill-like object, NJ Palisades,
 Actions performed on a hummock: sun clear,
 Possible Properties: dense, across creek, nil


 CPU time : 0.05

:

End of /home/csgrad/ss424/hummock.demo demonstration.


 CPU time : 0.99

: Isa(palisades1,hummock1)?


 CPU time : 0.01
```

The final result was a bit puzzling to us, since we felt that the NJ Palisades should not be included in the result. This may be a side effect of us using `Isa` the way we did, however just to be sure, we also added the statement `Isa(palisades1,hummock1)?` to see if there was a relationship of member-class there, but there was not, and so our confusion continued. However, all in all, we are pleased with the results.

9. Conclusion

Our understanding of the goals of the second area of research (the AI team) in the CVA project is to have a computational agent be able to read new words in context, generate definitions for these unknown words, then be able to both further refine the definitions with the introduction of new contextual material.

The unstated "ultimate" goal might be for such a cognitive agent to be able to learn additional new words automatically when they appear in context with previously generated definitions. This automated contextual vocabulary acquisition (ACVA) agent would need to be initially seeded with enough information about the physical world and common sense rules our human minds seem to have, or have access to a knowledge base which it can reference on demand, or perhaps even a communicate with a human counterpart that could answer questions that such an agent may pose, similar to the OpenCyc.Org project. (OpenCyc.Org 2006).

Perhaps one way a knowledge engineer may want to address this, as we have attempted to demonstrate, is to represent scenes of the world and let new word definitions be drawn and refined from these scenes. This may not be so unlike Minsky's frames (Minsky 1974) in that he was able to infer the steps of a newly forming script, by matching already seen steps from an existing script. Buying a hamburger at McDonalds may follow similar steps as say buying a slice of pizza at Luigi's.

We think such a technique, as we used here, could be versatile enough and applicable enough to be a component in the CVA project word algorithms. Imagine a large database of descriptions of images, simple descriptions, yet they contain unstated principles of relationships between intensional objects and common sense. We would not need to explicitly define these common sense rules, instead by just representing the scenes some good and valued inferences could be drawn. When an ACVA agent reads a sentence and comes across an unknown word, it can draw upon similarities in scenes and draw inferences. Although these inferences may be initially incorrect, they would be able to be refined as more context, or for that matter more memories, is added to the knowledge base.

The immediate criticism that should be leveled here is that just because a situation seems the same, does not automatically mean that objects in them are of similar types or are in contained in a superclass together. An easy example of this can also be found in the memory that we implement here. Obviously the leaves on a tree have nothing to do

with a person's eye, yet if we use a similar strategy that we did for hummock, we may draw that erroneous conclusion. There would need to be some consistency in the way we compare two scenes and what limitations must be placed on inferences that can be drawn for objects given widely differing properties like leaves and eyes. A whole common sense series of rules may need to be developed to ensure that correct inferences will be drawn. Evidently more needs to be discussed here.

We do not claim to have a perfect strategy; however, I think we see that there is a definite difference between representing common sense rules and relationships, and explicitly representing scenes of the world and allowing inference to do comparisons. In the case of this noun and this sentence, this technique may be appropriate, but when dealing with more complex ideas that may not be entirely based on physical interactions between objects, this similarity strategy may fall short of being an adequate solution. It would be interesting to see if such a technique could fit in to the CVA algorithms in any significant way.

Appendix A: Final “hummock.demo” run

A copy of this final run, complete with comments, can be found on the projects page of my school web site at:

http://www.cse.buffalo.edu/~ss424/cse663_projects.html

This run has been edited for readability. Case frame definitions, extraneous comments, and some section headers have been removed in this appendix. Please see Appendix B for the actual code, or the web page link above.

```
   Welcome to SNePSLOG (A logic interface to SNePS)

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.

: demo "hummock.demo"

File /home/csgrad/ss424/hummock.demo is now the source of input.


 CPU time : 0.00

:
;;; ============================================================================
;;;
;;; FILENAME:      hummock.demo
;;; DATE: 10/18/2006
;;; PROGRAMMER:    Scott Settembre
;;; ============================================================================

… TRIMMED FOR CLARITY …
… REMOVED CASE FRAME DEFINITIONS, EXTRANEOUS COMMENTS & SECTION HEADERS
… PLEASE SEE FINAL RUN ON THE WEB AT:
… http://www.cse.buffalo.edu/~ss424/cse663_projects.html…
… OR SEE APPENDIX B FOR AN ENTIRE LISTING OF THE CODE …

;;; ============================================================================
;;;
;;; Represent the Sentence and Word here:
;;;
;;; "Now the sun had risen enough to clear the dense hummock of forest across the creek,
;;;  and thus sunlight was striking and warming the leaves on this particular tree
;;;  for the first time since they'd frozen."
;;;
;;; Notes:
;;;       Words surrounded by "{" and "}" are not being represented, but have been
discussed
;;;       in the background knowledge section for perhaps future respresentation.
;;;
;;;       Words surrounded by "(" and ")" are just an expansion of the previous word or
;;;       phrase and may or may not be represented.
;;;
;;;       Words surrounded by "[" and "]" are represented either before or after the
;;;       piece(s) of knowledge that are being represented.
;;;
;;; ============================================================================
```

```
;; ------------------------------------------------------------------------------------
;; "{Now} the sun had risen {enough} to clear the dense hummock of forest across the
creek,.."
;; ------------------------------------------------------------------------------------

;; the sun had risen
Isa(Concept-called("hummock"),hummock1).

  wff2!:  Isa(Concept-called(hummock),hummock1)

 CPU time : 0.00

: Isa(Concept-called("sun"),sun1).

  wff4!:  Isa(Concept-called(sun),sun1)

 CPU time : 0.00

: Does(The-action(Concept-called("risen")),sun1).

  wff7!:  Does(The-action(Concept-called(risen)),sun1)

 CPU time : 0.00

:
;; {enough} to clear [the dense hummock]
Does(Act-wrt(hummock1,Concept-called("clear")),sun1).

  wff10!:  Does(Act-wrt(hummock1,Concept-called(clear)),sun1)

 CPU time : 0.00

:
;; the dense hummock
Is-property(Concept-called("dense"),hummock1).

  wff12!:  Is-property(Concept-called(dense),hummock1)

 CPU time : 0.00

:
;; hummock of forest
Isa(Concept-called("forest"),forest1).

  wff14!:  Isa(Concept-called(forest),forest1)

 CPU time : 0.01

: Is-a-part-of(forest1,hummock1).

  wff15!:  Is-a-part-of(forest1,hummock1)

 CPU time : 0.00

:
;; [the dense hummock of forest] across the creek
Isa(Concept-called("creek"),creek1).

  wff17!:  Isa(Concept-called(creek),creek1)

 CPU time : 0.00

: Has-relation(Concept-called("across"),creek1,hummock1).

  wff19!:  Has-relation(Concept-called(across),creek1,hummock1)

 CPU time : 0.00

:
;; ------------------------------------------------------------------------------------
;;  {and thus} sunlight was striking and warming the leaves on this {particular} tree
```

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

;; sunlight was striking and warming the leaves
Isa(Concept-called("sunlight"),sunlight1).

  wff21!:  Isa(Concept-called(sunlight),sunlight1)

 CPU time : 0.00

: Is-a-part-of(sun1,sunlight1).

  wff22!:  Is-a-part-of(sun1,sunlight1)

 CPU time : 0.00

: Does(Act-wrt(leaves1,Concept-called("striking")),sunlight1).

  wff25!:  Does(Act-wrt(leaves1,Concept-called(striking)),sunlight1)

 CPU time : 0.00

: Does(Act-wrt(leaves1,Concept-called("warming")),sunlight1).

  wff28!:  Does(Act-wrt(leaves1,Concept-called(warming)),sunlight1)

 CPU time : 0.00

:

;; the leaves on this {particular} tree
Isa(Concept-called("tree"),tree1).

  wff30!:  Isa(Concept-called(tree),tree1)

 CPU time : 0.00

: Isa(Concept-called("leaves"),leaves1).

  wff32!:  Isa(Concept-called(leaves),leaves1)

 CPU time : 0.00

: Is-a-part-of(tree1,leaves1).

  wff33!:  Is-a-part-of(tree1,leaves1)

 CPU time : 0.00

:

;; -------------------------------------------------------------------------------
;;  {for the first time since} they'd (they had) frozen."
;; -------------------------------------------------------------------------------

;; they'd (they had) frozen
Is-property(Concept-called("frozen"),leaves1).

  wff35!:  Is-property(Concept-called(frozen),leaves1)

 CPU time : 0.00

:




;;; ============================================================================
;;;
;;; Represent Background Knowledge here:
;;;
;;;       AdjacentWorldObjects can block other AdjacentWorldObjects
```

```
;;;
;;; ============================================================================================

;; ------------------------------------------------------------------------------------------
;; Background Knowledge Rule Section:
;;
;; This section is dedicated to implement rules and propositions that will allow further
;; inferences to be drawn about world knowledge not explicitly stated in the sentence.
;; The main question concerning background knowledge in this problem, however, is how
much
;; and what depth of representation is necessary to extract the real meaning of the word.
;;
;; There are some concepts in the sentence that I will not address more than describing
;; what they are and leave it for future development if needed. These words or phrases
;; may be idioms, however, the majority of them are concepts that do not deal with the
;; meaning of the word expressly. I have bracketed them with "[" and "]" in the sentence
;; represented below this section.
;;
;; ------------------------------------------------------------------------------------------

;; ------------------------------------------------------------------------------------------
;; Background Knowledge Rule Section: being similiar to a hill
;;
;;       In phase 1 of this project, I found that one of the subjects (subject 3) who
;;       described a hummock in terms of a scene that he was aware of from his memories.
;;       This imagery is what I wish to capture in this background representation.
;;
;;       "The sun cleared the pallisade and sunlight struck the subject's eye"
;;
;; ------------------------------------------------------------------------------------------


;; Subject 3's: Memory image
;; The sun cleared the NJ Palisades, sunlight struck subject3's eye
Isa(Concept-called("hill-like object"),hillobject1).

  wff37!:  Isa(Concept-called(hill-like object),hillobject1)

 CPU time : 0.00

: Isa(Concept-called("NJ Palisades"),palisade1).

  wff39!:  Isa(Concept-called(NJ Palisades),palisade1)

 CPU time : 0.00

: Isa(hillobject1,palisade1).

  wff40!:  Isa(hillobject1,palisade1)

 CPU time : 0.00

: Is-a-part-of(subject3,eyes3).

  wff41!:  Is-a-part-of(subject3,eyes3)

 CPU time : 0.00

: Is-a-part-of(sun1,sunlight1).

  wff22!:  Is-a-part-of(sun1,sunlight1)

 CPU time : 0.00

: Does(Act-wrt(palisade1,Concept-called("clear")),sun1).

  wff43!:  Does(Act-wrt(palisade1,Concept-called(clear)),sun1)

 CPU time : 0.00

: Does(Act-wrt(eyes3,Concept-called("striking")),sunlight1).
```

```
  wff45!:  Does(Act-wrt(eyes3,Concept-called(striking)),sunlight1)

 CPU time : 0.00

:

;; Rules to match this similar scene:
;; If the sun1 clears an object1, and sunlight1 strikes object2, then
;; it is possible that a similar scene like this may have
;; the object in the object1 position may be of the same class as object1
;; however, the object2 must also preserve its other relationships
all(x,y,z)(
 {
         Does(Act-wrt(x,Concept-called("clear")),sun1),
         Does(Act-wrt(y,Concept-called("striking")),sunlight1),
         Isa(z,x)
 }
 &=>
         all(h,j)(
          {
                  Does(Act-wrt(h,Concept-called("clear")),sun1),
                  Does(Act-wrt(j,Concept-called("striking")),sunlight1)
          }
          &=>
                  Isa(z,h)
         )
 )!

  wff52!:  Isa(Concept-called(NJ Palisades),hummock1)
  wff51!:  Isa(hillobject1,hummock1)
  wff50!:  Isa(Concept-called(hummock),palisade1)
  wff49!:  all(j,h)({Does(Act-wrt(j,Concept-called(striking)),sunlight1),Does(Act-
wrt(h,Concept-called(clear)),sun1)} &=> {Isa(hillobject1,h)})
  wff48!:  all(j,h)({Does(Act-wrt(j,Concept-called(striking)),sunlight1),Does(Act-
wrt(h,Concept-called(clear)),sun1)} &=> {Isa(Concept-called(NJ Palisades),h)})
  wff47!:  all(j,h)({Does(Act-wrt(j,Concept-called(striking)),sunlight1),Does(Act-
wrt(h,Concept-called(clear)),sun1)} &=> {Isa(Concept-called(hummock),h)})
  wff46!:  all(z,y,x)({Isa(z,x),Does(Act-wrt(y,Concept-
called(striking)),sunlight1),Does(Act-wrt(x,Concept-called(clear)),sun1)} &=>
{all(j,h)({Does(Act-wrt(j,Concept-called(striking)),sunlight1),Does(Act-wrt(h,Concept-
called(clear)),sun1)} &=> {Isa(z,h)})})
  wff45!:  Does(Act-wrt(eyes3,Concept-called(striking)),sunlight1)
  wff43!:  Does(Act-wrt(palisade1,Concept-called(clear)),sun1)
  wff40!:  Isa(hillobject1,palisade1)
  wff39!:  Isa(Concept-called(NJ Palisades),palisade1)
  wff37!:  Isa(Concept-called(hill-like object),hillobject1)
  wff32!:  Isa(Concept-called(leaves),leaves1)
  wff30!:  Isa(Concept-called(tree),tree1)
  wff25!:  Does(Act-wrt(leaves1,Concept-called(striking)),sunlight1)
  wff21!:  Isa(Concept-called(sunlight),sunlight1)
  wff17!:  Isa(Concept-called(creek),creek1)
  wff14!:  Isa(Concept-called(forest),forest1)
  wff10!:  Does(Act-wrt(hummock1,Concept-called(clear)),sun1)
  wff4!:  Isa(Concept-called(sun),sun1)
  wff2!:  Isa(Concept-called(hummock),hummock1)

 CPU time : 0.29

:



;;; ============================================================================
;;;
;;; Run the algorithm to see if the meaning of "hummock" can be derived
;;;
;;; "Now the sun had risen enough to clear the dense "hummock" of forest across the
creek,
;;;  and thus sunlight was striking and warming the leaves on this particular tree
;;;  for the first time since they'd frozen."
```

```
;;;
;;;
;;; ============================================================================


; Ask Cassie what "hummock" means:
^(snepsul::defineNoun "hummock")
 Definition of hummock:
 Possible Class Inclusions: hill-like object, NJ Palisades,
 Actions performed on a hummock: sun clear,
 Possible Properties: dense, across creek, nil


 CPU time : 0.05

:

End of /home/csgrad/ss424/hummock.demo demonstration.


 CPU time : 0.99

: Isa(palisades1,hummock1)?


 CPU time : 0.01

:
```

Appendix B: “hummock.demo” code listing

A copy of this code can be found on the projects page of my school web site at:

http://www.cse.buffalo.edu/~ss424/cse663_projects.html

This listing is printed in its entirety.

```
;;; ============================================================================
;;;
;;; FILENAME:      hummock.demo
;;; DATE: 10/18/2006
;;; PROGRAMMER:    Scott Settembre
;
; this template version:
; http://www.cse.buffalo.edu/~rapaport/CVA/snepslog-template-20061102-2.demo
;
; 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 [or see step-by-step below]; at the SNePS prompt (*), type:
;
;        demo "hummock.demo"
;
; Make sure all necessary files are in the current working directory
; or else use full path names.
;
;;; To run step-by-step instructions:
;;;      In a unix console do:
;;;                xemacs &
;;;      In xemacs in *scratch* do:
;;;                (load "cselisp.el") [then press Ctrl-J]
;;;      In xemacs do:
;;;                [Press ESC, then x, then type] run-acl [then press RETURN through all
prompts]
;;;      At lisp prompt do:
;;;                :ld /projects/shapiro/Sneps/sneps262 [then press RETURN]
;;;      After it loads, type:
;;;                (snepslog) [then press RETURN]
;;;      In snepslog, type:
;;;                demo "hummock.demo" [then press RETURN]
;;;      or to run the demo line by line, pressing key to continue to next line, type:
;;;                demo "hummock.demo" av [then press RETURN]
;;;
;;; ============================================================================


;;; ============================================================================
;;;
;;; Setup SNePSLOG and load appropriate packages
;;;
;;; ============================================================================

; Set SNePSLOG mode = 3
set-mode-3

; Turn off inference tracing; this is optional.
; If tracing is desired, enter "trace" instead of "untrace":
untrace inference

; Clear the SNePS network:
clearkb

; Define the arcs explicitly up front to avoid problem when loading the defun_noun
algorithm
```

```
%(define 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)

; Load the appropriate definition algorithm:
^(cl:load "/projects/rapaport/CVA/STN2/defun_noun")
;^(cl:load "defun_noun_snepslog")


; OPTIONAL:
; UNCOMMENT THE FOLLOWING CODE TO TURN FULL FORWARD INFERENCING ON:
;
^(cl:load "/projects/rapaport/CVA/STN2/ff")
;
; Returns:
;Warning: broadcast-one-report, :operator was defined in
;         /projects/snwiz/Install/Sneps-2.6.1/snip/fns/nrn-reports.lisp and is now being
defined in ff.cl
;t


;;; ============================================================================
;;;
;;; Following are steps needed to use the CVA noun algorithm.
;;;
;;; 1. Define the "frames" that are needed to represent the sentence as well
;;;       as the background knowledge. Although the frame names themselves can
;;;       be changed, the arcs that they describe between nodes and the names
;;;       of those arcs are required for proper operation of the noun algorithm.
;;;
;;;       Note: These frames need to be defined BEFORE the "paths" are defined, due
;;;       to the dependency on the arc names.
;;;
;;; 2. Define the "paths" that are needed to represent the information. "Paths" can
;;;       be considered as 'subconscious' reasoning or inference. In other words, an
;;;       inference defined as a path cannot be reasoned about, whereas, you can make a
;;;       rule (in background knowledge) and then include reasoning about the rule itself
;;;       in other background knowledge.  Any path that would be defined can be made into
a
;;;       inference rule, however, if you are not going to be reasoning about that
inference
;;;       then it "should" be defined as a path.
;;;
;;; 3. Define the "background knowledge" needed. This section should include all
;;;       information (rules or assertions) that will be useful in understanding the
;;;       sentence. This is not the representation of the sentence, just the rules
;;;       or assertions not explicitly referred to in the sentence needed to be able
;;;       to adequately represent the sentence pieces, ideas, relationships, concepts...
;;;
;;;       Note: The "background knowledge" section is defined prior to the sentence, but
;;;       it does not need to be completed prior to representing the sentence.  Additional
;;;       represention may be needed to tune to the inference in the noun algorithm.
;;;
;;; 4. Define the "sentence" itself. To represent the sentence, attempt to form the
;;;       assertions (and perhaps assertions should be all that is needed in this section)
;;;       based exactly on what the sentence is saying. Additional underlying meaning may
;;;       best be put in the "background knowledge" section.
;;;
;;; ============================================================================


;;; ============================================================================
;;;
;;; Define frames here:
;;;
;;; Each frame consists of a name and a sequence of arcs. Defined below are the frame
;;; names that I chose to increase readability. For each frame, I will additionally
;;; provide a description and an example.
;;;
```

```
;;; ============================================================================================

;;; ================================
;;; Frame: concept-called
;;; ================================


define-frame Concept-called (nil lex).


;SNePS SYNTAX      SEMANTICS
;
;  lex
;m----->w [[m]] is the concept expressed by uttering [[w]],
;                  where "w" is a word in the lexicon

;; Example 1:
;;       Concept-called (sun).
;;       [[represents "This intensional object (node) has a "lex" relationship with the
word 'sun'."]]
;;       [[or in other words "The 'concept of the sun' is called 'sun'."]]
;; produces a node (b1) with an arc labeled "lex" to a node ("sun"). This frame is used
;; to better facilitate the separation between how an object is represented intensionally
;; and what we call an object.  For example, the node (b1) is the intensional
representation
;; for the unique concept of sun, whereas, the word "sun" is the lexographical
representation
;; of the intensional representation of the concept of a sun. Mostly it is used to assist
the
;; noun and verb algorithms to communicate to the user.

;;; ================================
;;; Frame: Named
;;; ================================

define-frame Named (nil proper-name object).

;         m
;        / \
;object /   \proper-name
;      /     \
;  i <-       -> j
;
;SEMANTICS
;
;[[m]] is the proposition that entity [[i]] is called by the proper name
;[[j]], where "j" is a name in the lexicon.

;; Example 1:
;;       Named("Rover",specific-dog-1)
;;       [[represents "This [particular] dog is named 'Rover'."]]
;; produces a node (m1) with an arc named "object" pointing to a node (specific-dog-1)
and
;; an arc named "proper-name" to a to a node ("Rover"). This is required for the proper
;; operation of the noun algorithm, like the "lex" arc, the "proper-name" arc, allows the
;; algorithm to better communicate the definition of the word to the user.


;;; ================================
;;; Frame: Isa
;;; ================================

define-frame Isa (nil class member).

;SNePS SYNTAX
;
;         m
;        / \
;member /   \class
;      /     \
;  i <-       -> j
```

```
;
;SEMANTICS
;
;[[m]] is the proposition that individual entity [[i]] is a (member of the class [[j]]

;; Example 1:
;;       Isa(canine,specific-dog-1).
;;       [[represents "This [particular] dog is a canine."]]
;; produces a node (m1) with an arc labeled "member" to a node (specific-dog1)
;; and with an arc labeled "class" to  a node (canine)
;;
;; Example 1.a.:
;;       Isa(dog,specific-dog-1)
;;       [[represents "This [particular] dog is a dog."]]
;;
;; Example 2:
;;       Isa(Concept-called(canine),specific-dog-1).
;;       [[represents "This [particular] dog is a canine."]]
;; produces a node (m1) with an arc labeled "member" to a node (specific-dog-1)
;; and with an arc labeled "class" to  a node (b1).  Node (b2) additionally has
;; an arc labeled "lex" to a node ("canine").  This assertion look is common in the
sentence
;; and background knowledge representations because the noun algorithm requires the use
of
;; "lex" arcs to understand the relationship between the concepts that are being reasoned
;; about and the lexicon that represents them.
;;
;; BAD Example 1: *USE AKO INSTEAD*
;;       Isa(mammal,dog).
;;       [[represents "A dog is a mammal." (Or "dogs are mammals")]]
;; produces a node (m1) with an arc labeled "member" to a node (dog) and an arc labeled
;; "class" to a node (mammal).  This should NOT BE USED and instead the frame AKO or
;; "also known as" should be used to represent subclass and superclass relationships.
;; Isa should be used for representing inclusion in a class or group.
;; AKO should be used to represent that one class/group is a subclass of another.
;; See more at AKO definition below.


;;; ================================
;;; Frame: AKO
;;; ================================

define-frame AKO (nil superclass subclass).

;SNePS SYNTAX
;
;           m
;          / \
;subclass /   \superclass
;        /     \
;     i<-       ->j
;
;SEMANTICS
;
;[[m]] is the proposition that entity [[i]] is a subclass of entity [[j]].

;; Example 1:
;;       AKO(mammal,dog).
;;       [[represents "Dogs is a subclass of mammals."]]
;; produces a node (m1) with an arc labeled "subclass" to a node (dog) and an arc labeled
;; "superclass" to a node (mammal).  This is used to represent subclass and superclass
;; relationships. AKO should be used to represent that one class/group is a subclass of
another.
;;
;; Exmaple 2:
;;       AKO(Concept-called(mammal),Concept-called(dog)).
;;       [[represents "Dogs is a subclass of mammals."]]
;; produces a node (m1) with an arc labeled "subclass" to a node (b1) and an arc labeled
;; "superclass" to a node (b2). Additionally node (b1) has an arc labeled "lex" to a node
(dog)
```

```
;; and node (b2) has an arc labeled "lex" to a node (mammal). This sort of assertion will
be
;; will be used extensively to represent information. The difference between example 1
and
;; example 2 is used to help the noun algorithm communicate to the user.

;;; ================================
;;; Frame: Is-property
;;; ================================

define-frame Is-property (nil property object).

;SNePS SYNTAX
;
;         m
;        / \
;object /   \property
;      /     \
;   i<-       ->j
;
;SEMANTICS
;
;[[m]] is the proposition that entity [[i]] has the property [[j]].

;; Example 1:
;;       Is-property(bright, sun)
;;       [[represents "the sun is bright".]]
;; produces a node (m1) with an arc labeled "object" to a node (sun) and an arc labeled
;; "property" to a node (bright). This is used to represent a property of an object. This
is not
;; be used the same way the frame "Isa" or "AKO" is used.  Is-property should be used
when attempting
;; to represent a "property" or a "feature" of a member or a class.  I named it "Is-
property" instead
;; "Is" so that there would be no confusion between "Is" and "Isa"
;;
;; Example 2:
;;       Is-property(Concept-called(bright),Concept-called(sun))
;;       [[represents "the sun is bright".]]
;; produces a node (m1) with an arc labeled "object" to node (b1) which has an arc
labeled "lex" to a
;; node (sun), and node (m1) has another arc labeled "property" to a node (b2) which has
an arc labeled
;; "lex" to a node (bright). This is a typical use of the frame when using the noun/verb
algorithms.

;;; ================================
;;; Frame: Has-relation
;;; ================================

define-frame Has-relation (nil rel object2 object1).

;SNePS SYNTAX
;
;            m
;          / |  \
;         /  |   \
;object1 /   |rel \object2
;       /    |     \
;    i<-     ->k    ->j
;
;SEMANTICS
;
;[[m]] is the proposition that [[i]] is related by relation [[k]] to [[j]].

;; Example 1:
;;       Has-relation(behind,hill1,tree1)
;;       [[represents "That tree is behind that hill"]]
;;
;; Example 2:
;;       Has-relation(younger,adult,child)
```

```
;;       [[represents "A child is younger than an adult"]]
;;
;; Example 3:
;;       Has-relation(under,trunk,root)
;;       [[represents "The roots are under the trunks"]]
;;
;; produces a node (m1) which has a arc labeled "rel" (meaning the relation) to a node
(root)
;; and (m1) also has an arc labeled "object1" to a node (root) and (m1) also has an arc
labeled
;; "object2" to a node (trunk).
;;
;; Notice these examples have different "types" of nodes. Example 1 has specific objects
related to
;; each other, example 2 relates two classes to each other, and in example 3 relates
members of a
;; class to other members of a class. In other words, any two intensional objects "have a
relation"
;; between them, which is why I called the frame "Has-relation".

;;; ================================
;;; Frame: Possesses
;;; ================================

define-frame Possesses (nil object possessor rel).

;           m
;         / |  \
;        /  |   \
;object /   |rel \possessor
;      /    |     \
;   i<-     ->k    ->j
;
;SEMANTICS
;
;[[m]] is the proposition that [[i]] is a [[k]] of [[j]]; or:
;[[m]] is the proposition that [[i]] is [[j]]'s [[k]]   ; or:
;[[m]] is the proposition that [[i]] is possessed (in some unspecified way)
;      by [[j]] and that the relationship of [[i]] to [[j]] is [[k]]

;; Example 1:
;;       Possesses(computer1,Scott1,computer)
;;       [[represents "This computer is Scott's computer" or "This is Scott's computer"]]
;; Example 2:
;;       Possesses(computer1,Named(Scott,Scott1),Concept-called(computer))
;;       [[represents "This computer is Scott's computer" or "This is Scott's computer"]]
;;
;; produces for example 2, a node (m3) with an arc "object" to a node (m1), an arc "rel"
;; to a node (computer1), and an arc "possessor" to node (m2).  Node (m1) has an arc
"object" to
;; a node (Scott1) and an arc "proper-name" to an node (Scott).  Node (m2) has an arc
"lex" to a
;; node (computer).  Additional examples:
;;
;; Example 3:
;;       Possesses(Concept-called(paw),Concept-called(dog),Concept-called(paw))
;;       [[represents "A dog has a paw]]
;;
;; Example 4:
;;       Possesses(b2,Named(Earth,b1),Concept-called(orbit))
;;       [[represents "This (b2) is Earth's orbit"]]



;;; ================================
;;; Frame: Does
;;;         [Just "the action-object" part of the SNePS syntax tree]
;;; ================================

define-frame Does (nil act agent).
;Michael pets Bella.
```

```
;i.e. Does(Act-wrt(Concept-called(pet),bella1),mike1).

;SNePS SYNTAX
;
;         m
;        / \
;  agent/   \act
;      /     \
;   i<-       ->n
;              / \
;       action/   \object
;            /     \
;         j<-       ->k
;
;SEMANTICS
;
;[[m]] is the proposition that agent [[i]] performs action [[j]]; or:
;[[m]] is the proposition that agent [[i]] performs the act consisting of action [[j]].
;
;NB:  This is used for intransitive verbs, i.e., verbs that do not take
;an object.  The "action" arc is needed for consistency with the above
;case-frame as well as consistency with SNeRE.
;
;NB:  Note that node "n" has no separate semantics.  However, we could
;     provide one:  [[n]] is the act of performing action [[j]].



;;; ================================
;;; Frame: Act-wrt
;;;          [Just "the agent - act" part of the SNePS syntax tree]
;;; ================================

define-frame Act-wrt (nil object action).

;SNePS SYNTAX
;
;         m                                        m
;        / \                                      / \
;  agent/   \act               agent/   \act
;      /     \                                /     \
;   i<-       ->n   +OR+        i<-       ->n
;              / \                           /
;       action/   \object           action/
;            /     \                            /
;         j<-       ->k                       j<-
;
;SEMANTICS
;
;[[m]] is the proposition that agent [[i]] performs action [[j]] with respect to object
[[k]].
;
;NB:  This is used for transitive verbs, i.e., verbs that take an object.
;
;NB:  Note that node "n" has no separate semantics.  However, we could
;     provide one:  [[n]] is the act of performing action [[j]] wrt [[k]]


;;; ================================
;;; Frame: The-action
;;;          [Just "the action" part of the SNePS tree]
;;; ================================

define-frame The-action (nil action).
;Michael sings.
;i.e. Does(The-action(Concept-called(sing)),mike1).

;SNePS SYNTAX
;
;         m
;        / \
```

```
;  agent/   \act
;      /     \
;   i<-       ->n
;               /
;       action/
;            /
;           ->j
;
;SEMANTICS
;
;[[m]] is the proposition that agent [[i]] performs action [[j]]; or:
;[[m]] is the proposition that agent [[i]] performs the act consisting of action [[j]].
;
;NB:  This is used for intransitive verbs, i.e., verbs that do not take
;an object.  The "action" arc is needed for consistency with the above
;case-frame as well as consistency with SNeRE.
;
;NB:  Note that node "n" has no separate semantics.  However, we could
;     provide one:  [[n]] is the act of performing action [[j]].



;;; ================================
;;; Frame: Is-located-at
;;; ================================

define-frame Is-located-at (nil location object).

;SNePS SYNTAX
;
;         m
;        / \
;object /   \location
;      /     \
;   i<-       ->j
;
;SEMANTICS
;
;[[m]] is the proposition that [[i]] is located at location [[j]].
;

;; Example 1:
;;       Is-located-at(hill1,tree1)
;; produces a node(m) with two arcs, onc called "part" to a node (tree1) and the other
;; called "location" to a node (hill1).

;;; ================================
;;; Frame: Isa-part-of
;;; ================================

define-frame Is-a-part-of (nil whole part).

;SNePS SYNTAX
;
;         m
;        / \
;  part /   \whole
;      /     \
;   i<-       ->j
;
;SEMANTICS
;
;[[m]] is the proposition that [[i]] is a part of [[j]].

;; Example 1:
;;       Is-a-part-of (tree1,leaves1).
;; produces an node (m) with two arcs, one called "part" to a node called (leaves1) and
one called
;; "whole" to a node called (tree1).
;;
;; Example 2:
```

```
;;       Is-a-part-of (Concept-called(tree),Concept-called(leaves)).
;; produces a node (m) with two arcs, one called "part" to node (b1) and the other called
;; "whole" to node (b2).  Node (b1) has an arc called "lex" to a node (tree). and Node
(b2)
;; has an arc called "lex" to a node (leaves).  Example 1 and example 2 are very
different, in
;; that Example 1 is talking about a specific tree (tree1) and a specific group of leaves
;; (leaves1), whereas, Example 2 is talking about that any tree has as part of it a group
;; of leaves.


;;; ================================
;;; Frame: Synonyms
;;; ================================

define-frame Synonyms (nil synonym synonym).

;SNePS SYNTAX
;
;          m
;         / \
;synonym /   \synonym
;       /     \
;    i<-       ->j
;
;SEMANTICS
;
;[[m]] is the proposition that [[i]] and [[j]] are synonyms.

;; Example 1:
;;       Synonyms(big,large).
;; produces a node (m) with an arc labeled "synonym" leading to the node (big) and an arc
;; labeled "synonym" leading to the node (large).
;;
;; Example 2:
;;       Synonyms(Concept-called(big),Concept-called(large)).
;; produces a node (m) with an arc labeled "synonym" leading to a node (b1) with an arc
;; labeled "lex" to the node (big), and from the node (m) another arc labeled "synonym"
leading
;; to a node (b2) with an arc labeled "lex" to the node (large).


;;; ================================
;;; Frame: Antonyms
;;; ================================

define-frame Antonyms (nil antonym antonym).

;SNePS SYNTAX
;
;          m
;         / \
;antonym /   \antonym
;       /     \
;    i<-       ->j
;
;SEMANTICS
;
;[[m]] is the proposition that [[i]] and [[j]] are antonyms.

;; Example 1:
;;       Antonyms(big,small).
;; produces a node (m) with an arc labeled "antonym" leading to the node (big) and an arc
;; labeled "antonym" leading to the node (small).
;;
;; Example 2:
;;       Antonyms(Concept-called(big),Concept-called(small)).
;; produces a node (m) with an arc labeled "antonym" leading to a node (b1) with an arc
;; labeled "lex" to the node (big), and from the node (m) another arc labeled "antonym"
leading
;; to a node (b2) with an arc labeled "lex" to the node (small).
```

```
;;; ================================
;;; Frame: Equivalent
;;; ================================

define-frame Equivalent (nil equiv equiv).

;SNePS SYNTAX
;
;          m
;         / \
;  equiv /   \equiv
;       /     \
;    i<-       ->j
;
;SEMANTICS
;
;[[m]] is the proposition that mental entities [[i]] and [[j]] correspond to the same
external object.

;; Example 1:
;;       Equivalent(b1,b2).
;; produces a node (m) with an "equiv" arc to a node (b1) and another "equiv" arc to a
node (b2).
;;
;; Example 2:
;;       Equivalent(Named(Rover,dog1),Possession(dog2,Concept-called("dog"),me1)).
;; produces a node (m) with an "equiv" arc to a node (b1) which has two arcs, one called
;; "object" to a node (dog1) and one called "proper-name" to a node (Rover). Also, node
(m)
;; has another "equiv" arc to a node (b2) which has three arcs, one arc called "object"
to
;; node (dog2), another arc called "rel" to node (b3) which has itself an arc to node
(dog) and
;; lastly an arc "possessor" to a node (me1).



;;; ================================================================================
;;;
;;; Define paths here:
;;;
;;; Each path defined represents a "subconscious" way to look at an inference rule. The
;;; path allows SNePS to follow a sequence of arcs without the use of an actual inference
;;; rule that would exist in the network as nodes.  Therefore, inference done using paths
;;; bypasses using the network itself as being the place where that rule is represented
and
;;; instead is represented directly in other working memory.
;;;
;;; Note 1: I suspect that defining paths may be faster during the inference phase, and
so if
;;; it is not indended to reason about the rule itself (the rule is simple or represents
;;; background knowledge that is not the focus of the sentence) then it would be optimal
;;; to represent that knowledge through "define-path" instead of through an inference
rule.
;;;
;;; Note 2: See "/projects/rapaport/CVA/mkb3.CVA/paths/snepslog-paths" for some examples
and
;;; additional paths not used here.
;;;
;;; Note 3: These examples give errors so I am leaving them commented out, since I am not
;;; using them in my representation.
;;;
;;; ================================================================================

;;; Sub1
;define-path sub1 (compose object1- superclass- ! subclass superclass- ! subclass).

;;; Super1
```

```
;define-path super1 (compose superclass subclass- ! superclass object1- ! object2).

;;; The superclass path can be defined by either superclass or super1
;define-path superclass (or superclass super1).



;;; ============================================================================================
;;;
;;; Represent the Sentence and Word here:
;;;
;;; "Now the sun had risen enough to clear the dense hummock of forest across the creek,
;;;  and thus sunlight was striking and warming the leaves on this particular tree
;;;  for the first time since they'd frozen."
;;;
;;; Notes:
;;;       Words surrounded by "{" and "}" are not being represented, but have been
discussed
;;;       in the background knowledge section for perhaps future respresentation.
;;;
;;;       Words surrounded by "(" and ")" are just an expansion of the previous word or
;;;       phrase and may or may not be represented.
;;;
;;;       Words surrounded by "[" and "]" are represented either before or after the
;;;       piece(s) of knowledge that are being represented.
;;;
;;; ============================================================================================

;; --------------------------------------------------------------------------------------
;; "{Now} the sun had risen {enough} to clear the dense hummock of forest across the
creek,.."
;; --------------------------------------------------------------------------------------

;; the sun had risen
Isa(Concept-called("hummock"),hummock1).
Isa(Concept-called("sun"),sun1).
Does(The-action(Concept-called("risen")),sun1).

;; {enough} to clear [the dense hummock]
Does(Act-wrt(hummock1,Concept-called("clear")),sun1).

;; the dense hummock
Is-property(Concept-called("dense"),hummock1).

;; hummock of forest
Isa(Concept-called("forest"),forest1).
Is-a-part-of(forest1,hummock1).

;; [the dense hummock of forest] across the creek
Isa(Concept-called("creek"),creek1).
Has-relation(Concept-called("across"),creek1,hummock1).

;; --------------------------------------------------------------------------------------
;;  {and thus} sunlight was striking and warming the leaves on this {particular} tree
;; --------------------------------------------------------------------------------------

;; sunlight was striking and warming the leaves
Isa(Concept-called("sunlight"),sunlight1).
Is-a-part-of(sun1,sunlight1).
Does(Act-wrt(leaves1,Concept-called("striking")),sunlight1).
Does(Act-wrt(leaves1,Concept-called("warming")),sunlight1).


;; the leaves on this {particular} tree
Isa(Concept-called("tree"),tree1).
Isa(Concept-called("leaves"),leaves1).
Is-a-part-of(tree1,leaves1).


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

```
;;  {for the first time since} they'd (they had) frozen."
;; ------------------------------------------------------------------------------------

;; they'd (they had) frozen
Is-property(Concept-called("frozen"),leaves1).




;;; ====================================================================================
;;;
;;; Represent Background Knowledge here:
;;;
;;;       AdjacentWorldObjects can block other AdjacentWorldObjects
;;;
;;; ====================================================================================

;; ------------------------------------------------------------------------------------
;; Background Knowledge Rule Section:
;;
;; This section is dedicated to implement rules and propositions that will allow further
;; inferences to be drawn about world knowledge not explicitly stated in the sentence.
;; The main question concerning background knowledge in this problem, however, is how
much
;; and what depth of representation is necessary to extract the real meaning of the word.
;;
;; There are some concepts in the sentence that I will not address more than describing
;; what they are and leave it for future development if needed. These words or phrases
;; may be idioms, however, the majority of them are concepts that do not deal with the
;; meaning of the word expressly. I have bracketed them with "[" and "]" in the sentence
;; represented below this section.
;;
;; ------------------------------------------------------------------------------------

;; ------------------------------------------------------------------------------------
;; Background Knowledge Rule Section: being similiar to a hill
;;
;;       In phase 1 of this project, I found that one of the subjects (subject 3) who
;;       described a hummock in terms of a scene that he was aware of from his memories.
;;       This imagery is what I wish to capture in this background representation.
;;
;;       "The sun cleared the pallisade and sunlight struck the subject's eye"
;;
;; ------------------------------------------------------------------------------------


;; Subject 3's: Memory image
;; The sun cleared the NJ Palisades, sunlight struck subject3's eye
Isa(Concept-called("hill-like object"),hillobject1).
Isa(Concept-called("NJ Palisades"),palisade1).
Isa(hillobject1,palisade1).
Is-a-part-of(subject3,eyes3).
Is-a-part-of(sun1,sunlight1).
Does(Act-wrt(palisade1,Concept-called("clear")),sun1).
Does(Act-wrt(eyes3,Concept-called("striking")),sunlight1).

;; ------------------------------------------------------------------------------------
-----
;; Can't get "exists" logical symbol to work as described in
;; http://www.cse.buffalo.edu/~jsantore/snepsman/node106.html#SECTION009100000000000000000
;; so I will comment out the counter example
;; I was merely going to implement a rule that made sure that there was a consistant
;; assertion about being a part of a forest. The idea was to make sure that something
;; was NOT not true, as in ~(~Is-a-part-of(m,h)) but since the ~(~ didn't seem to work
the
;; way I hoped, I attempted the use of "exists", but the SNePSLOG interpreter would not
accept
;; the sytax as described.
;
```

```
;; Possible flaw, why not a building-like object? Try to say, a building cannot be "of
forest"
;; Let's try an example.
;; Subject 3's: Memory image
;; The sun cleared the Empire State Building, sunlight struck subject's eye
;Isa(Concept-called("building-like object"),buildingobject1).
;Isa(Concept-called("Empire State Building"),building1).
;Isa(buildingobject1,building1).
;Is-a-part-of(subject3,eyes3).
;Is-a-part-of(sun1,sunlight1).
;Does(Act-wrt(building1,Concept-called("clear")),sun1).
;Does(Act-wrt(eyes3,Concept-called("striking")),sunlight1).
;
;; a building-like object cannot be "of forest"
;~Is-a-part-of(forest1,buildingobject1).
;; ------------------------------------------------------------------------------------
-----


;; Rules to match this similar scene:
;; If the sun1 clears an object1, and sunlight1 strikes object2, then
;; it is possible that a similar scene like this may have
;; the object in the object1 position may be of the same class as object1
;; however, the object2 must also preserve its other relationships
all(x,y,z)(
 {
         Does(Act-wrt(x,Concept-called("clear")),sun1),
         Does(Act-wrt(y,Concept-called("striking")),sunlight1),
         Isa(z,x)
 }
 &=>
         all(h,j)(
          {
                  Does(Act-wrt(h,Concept-called("clear")),sun1),
                  Does(Act-wrt(j,Concept-called("striking")),sunlight1)
          }
          &=>
                  Isa(z,h)
         )
 )!


;; ------------------------------------------------------------------------------------
;; Background Knowledge Rule Section: "enough"
;;       (Commenting on but not using for representation.)
;;
;;       When something does something enough, it implies that there was some
;;       action that has been occurring up until this point in time which makes
;;       it possible for another action, event, or proposition to be true.
;;
;; ------------------------------------------------------------------------------------

;; ------------------------------------------------------------------------------------
;; Background Knowledge Rule Section: "now"
;;       (Commenting on but not using for representation.)
;;
;;       Referring to the current time from the perspective of the speaker.
;;       In this sentence, if we leave out the word "Now", it is assumed that we are
;;       talking about that time. However in this sentence, it probably does not
;;       mean the current time, but the current point in time of the narrative.
;;
;; ------------------------------------------------------------------------------------

;; ------------------------------------------------------------------------------------
;; Background Knowledge Rule Section: "and thus"
;;       (Commenting on but not using for representation.)
;;
;;       Could mean "therefore" or "and so" in this sentence. "Because" the actions in
the
;;       first part of the sentence happened, this next thing is happening.  Or we could
;;       think of this as some action, event, proposition being true, then enabling
```

```
;;       the next action, event, or proposition to be true or happening.
;;
;; ------------------------------------------------------------------------------------

;; ------------------------------------------------------------------------------------
;; Background Knowledge Rule Section: "particular"
;;       (Commenting on but not using for representation.)
;;
;;       Mostly likely implies that there are other objects to choose from of the same
;;       type, but we are told to consider only a specific object of the objects to
;;       choose from.
;;
;; ------------------------------------------------------------------------------------


;; ------------------------------------------------------------------------------------
;; Background Knowledge Rule Section: "for the first time since"
;;       (Commenting on but not using for representation.)
;;
;;       If we are dealing with a timeline, this would refer to an event occuring at a
;;       point in relation to another point where another event (of the same type or a
;;       different type) occurred previously (or before the event in question).
;;
;; ------------------------------------------------------------------------------------




;;; ====================================================================================
;;;
;;; Run the algorithm to see if the meaning of "hummock" can be derived
;;;
;;; "Now the sun had risen enough to clear the dense "hummock" of forest across the
creek,
;;;  and thus sunlight was striking and warming the leaves on this particular tree
;;;  for the first time since they'd frozen."
;;;
;;;
;;; ====================================================================================


; Ask Cassie what "hummock" means:
^(snepsul::defineNoun "hummock")
```

References

1. 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): 5.

2. Rapaport, William J., & Kibby, Michael W. (2001-2005), "Contextual Vocabulary Acquisition: from algorithm to curriculum", [http://www.cse.buffalo.edu/~rapaport/cva.html]

3. Shapiro, Stuart C. and The SNePS Research Group (1999 - 2006), [http://www.cse.buffalo.edu/sneps/].

4. Shapiro, Stuart C. and The SNePS Implementation Group (2004), "SNEPS 2.6.1 User's Manual", [http://www.cse.buffalo.edu/sneps/Manuals/manual261.pdf]

5. Napieralski, Scott (2003), "Noun Algorithm Case Frames", [http://www.cse.buffalo.edu/ stn2/cva/case-frames/']

6. OpenCyc.Org (2006), [http://www.opencyc.org/].

7. Minsky, Marvin (1974) "A Framework for Representing Knowledge", Reprinted in The Psychology of Computer Vision, P. Winston (Ed.), McGraw-Hill, 1975. Shorter versions in J. Haugeland, Ed., Mind Design, MIT Press, 1981, and in Cognitive Science, Collins, Allan and Edward E. Smith (eds.) Morgan-Kaufmann, 1992 ISBN 55860-013-2].