Revealing Nadsat:
Defining Words of a Fictional Language through Contextual Vocabulary Acquisition

Mark Zorn
CSE663: Advanced Knowledge Representation
mrzorn@cse.buffalo.edu

December 12, 2006

Abstract

The Contextual Vocabulary Acquisition (CVA) project at the University at Buffalo is an ongoing project which attempts to computationally represent a passage of text with a word of unknown definition and, through the context of the sentence or passage, make a logical guess at the meaning of the word. When a human comes across such a word, they will use the other words in the surrounding sentence(s) as well their background knowledge to deduce what the particular unknown word means. While this may come as second nature to most humans, it is no easy task for a computer. In this study, a sentence from the Anthony Burgess novel A Clockwork Orange will be represented in the Semantic Network Processing System (SNePS). Throughout the novel, Burgess uses a fictional language which he has dubbed nadsat. Nadsat is a slang used by the characters within the novel and any reader, be it human or computer, would have to use large amounts of contextual information to infer the meaning of these nadsat words. This study focuses on deriving a meaning for the nadsat word *cantora*. Once a passage is represented in a semantic network, a set of background knowledge must be told to Cassie, the computerized cognitive agent of SNePS. Cassie will then, with the help of a CVA noun definition algorithm, attempt to provide possible definitions for cantora. A discussion of the results of this noun algorithm will be presented, as well as the results of verbal protocols performed on humans reading this same passage. Additional future work that will benefit the future of the CVA project will also be discussed.

1 – The CVA Project and SNePS

How does one comprehend what they are reading? A reader with sufficient education and vocabulary size has the ability to read and understand a vast array of books, papers, magazines, etc. How does a reader reach this point? From the time a person first starts learning to read as a child, their vocabulary, as well as skill set for inferring the definition of a particular word or set of words, increases. The average adult reader, upon discovering a word they do not know, will not immediately use a dictionary to learn the definition of this word. Although using a dictionary in this way would give you a correct answer every time, in many cases the reader will not need the dictionary definition of a word to understand what they are reading. But, if they do not know what a specific word means, how are they able to grasp the meaning of a sentence or passage? This question has a few answers. First, and most simply, the word may not be important to the overall meaning and can safely be ignored. However, more interestingly, and the focus of this study, an approximate definition for the word can be acquired through the use of context, using surrounding words to help determine what the unknown word means, combined with the background knowledge of the reader.

The CVA project at the University at Buffalo has defined contextual vocabulary acquisition as "the active, deliberate acquisition of word meanings from text by reasoning from contextual cues, background knowledge, and hypotheses developed from prior encounters with the word, but without external sources of help such as dictionaries or people." (CVA Project Description) As human readers, each of us most likely performs contextual vocabulary acquisition in different ways. Although these differences may be slight or vast, there is currently no one algorithm that can be used by humans to derive

meanings from unknown words. The teaching of CVA to a student, whether it be a young child just learning to read, a person learning a new language, or anyone interested in improving their reading comprehension skill, could be improved if such an algorithm is developed. Once created, this algorithm could be taught to a student with the goal of allowing the student to stop guessing what an unknown word means and be able to compute this definition.

The Semantic Network Processing System (SNePS), also developed at the University of Buffalo, is used as a tool for the CVA project. SNePS has the ability to represent natural language in a semantic network. The semantic network is made up of nodes, which represent individual constants and functional terms, and directed arcs, which represent the relationships between these nodes. (SNePS wiki) Through the use of two languages, SNePSUL and SNepSLOG, a knowledge engineer can build a semantic network to represent just about anything found in natural language.

The semantic networks created in SNePS are viewed as the mind, or knowledge base, of an agent named Cassie. As Cassie's knowledge base increases she will attempt to draw inferences and essentially learn new information by creating nodes in the semantic network. Cassie will make inferences when reading a passage with an unknown word based upon her background knowledge; what she already knows about the world. These inferences, and the semantic network in its entirety, will be used in conjunction with a word definition algorithm created for the CVA project at the University at Buffalo. With the help of this algorithm, Cassie should be able to derive a definition for an unknown word provided she has the necessary background information in the knowledge base and the passage is converted correctly into a semantic network.

2 – The Passage and Unknown Word

If there was ever a list of novels that required CVA to fully comprehend not only the overall meaning of passages, but each individual sentence, Anthony Burgress' A Clockwork Orange would be near the top of it. For this novel, Burgress invented a fictional language called nadsat which is spoken by the narrator and characters. The word nadsat itself is the "suffix of Russian numerals from 11 to 19 (-надцать). The suffix slurs the Russian words for 'on ten' — i.e., 'one-on-ten', 'two-on-ten', and so on — and thus forms an almost exact linguistic parallel to the English '-teen'." (Nadsat wiki) Nadsat serves as a slang language used by the teens in the novel and is comprised of English with some Russian, as well as influences from Cockney and the King James Bible. But just to make things even more difficult, Burgress flat out invented many of the words himself. (Nadsat wiki) Obviously this poses a problem for a reader of the novel. When reading a sentence with a nadsat word in it, a reader can not use an English dictionary, or Russian dictionary, or any dictionary for that matter, to look up these words. The definition of these words can be derived solely from CVA.

The source passage for this study on CVA in SNePS is taken from A Clockwork Orange.

"Then we came to a very neat like cantora with typewriters and flowers on the desks, and at the like chief desk the top millicent was sitting…"

The unknown word we have selected from this passage is cantora. Millicent is another nadsat word which will we assume to mean policeman/police officer.

Normally, upon choosing a passage to represent and an unknown word whose meaning must be derived, the next step would be to look up the word in a dictionary. However, we are not that fortunate in this case. This word will not appear in any dictionary in its present form. Using CVA ourselves on this passage, we may guess that cantora means office, or perhaps more specifically, a police office or station. After a little research, it is discovered that cantora comes from the Russian word kontora, meaning office. (Nadsat Dictionary) But how did we use CVA to derive this definition? This question will be key as we attempt to represent the passage and ask Cassie what she believes to be the definition of cantora.

2 – CVA by Humans - Verbal Protocols

An early step in the CVA project is to have a number of people read the passage. Unless any of these people have read A Clockwork Orange before, chances were very high that none of them had ever heard or seen the word cantora before reading the passage. As such, these test subjects should give unbiased results when asked about the CVA they performed on the passage. Each person was asked to read the passage, explain what they though the word cantora meant, and describe how they derived this definition, including what information they use from the passage, as well as what prior background knowledge was used.

We will look at responses from three of the test subjects, Susan, Mike, and Katey. When asked what they thought the word cantora meant, they responded as follows (note that the subjects where told that a millicent was a policeman/police officer):

Susan: "An Office"

Mike: "A police station or the sort. From the sound of the word and because of the fact that flowers are present, perhaps it's a little one in a little town."

Katey: "…an open floored police station, (or more generally an office floor/cubicle farm) probably pre-1980's (at least) or in a less technologically savvy country."

From these three responses it is immediately evident that there is enough information in the passage for a person to infer a definition for cantora. Although each test subject gave a different response, each person was able to capture the overall meaning of the word. These differing responses are typical of CVA when performed by humans. Each of the three test subjects had a different method for determining the

meaning of cantora. Likewise, each individual has a different set of background knowledge which they applied while reading the passage.

When asked what particular information and background knowledge was used to infer these definitions, they responded as follows:

Susan: "Typewriters and desks seem like things that would be found in an office and I couldn't really think of another word that seemed to fit."

Mike: "…from the rest of the sentence, desks, typewriters, millicenti all around… knowing that cantora is a noun based on context… preconceived notion of what a police station would look like."

Katey: "All the nouns are office things, typewriters, desks with personal decor, and a supervisor at a "chief desk" who is also a police officer. I used my prior knowledge of what items are typically found in offices, what kind of technology is from what era, and the definition of millicent given after the passage."

Looking at these responses, each person used different information to infer the definition of cantora. However, there are similarities found between the three responses. All three test subjects determined that the desks and typewriters mentioned in the passage caused them to believe that a cantora was an office of some sort. From these responses it seems that the idea of offices containing desks and typewriters is an important piece of background knowledge used while performing CVA on this passage. In fact, almost every response mentioned the desks and typewriters as a means for determining that a cantora must be an office. Clearly this will have to be represented in Cassie's background knowledge.

The response of Lauren, a fourth test subject, while not correct in her end result of CVA, is worth mentioning. Lauren stated that she believed cantora to be "a canteen" and came to this conclusion because "It sounded Spanish for canteen… I have no prior knowledge of the Spanish word for canteen, it just sounds like it." While it appears that Lauren was just being lazy and was perhaps not the best test subject for these verbal protocols, she does bring up an interesting point. Often in different languages, words with similar meanings will sound or look alike. For example, the English word calendar and the Spanish word calendario both refer to the same object/concept. Calendar and calendario are examples of cognates, two words that were derived from the same ancestral language and therefore maintain similarities in look and sound. In fact, in the particular case of cantora, if one were to look at the Russian word for office, kontora, you will notice that these could be considered cognates.

While the use of cognates will not work for all words in Burgess' nadsat, it is not necessarily a bad step in CVA. Perhaps if Lauren were more familiar with Russian instead of Spanish, she would have assumed that a cantora was in fact an office. The idea behind using cognates in a computational CVA algorithm would be a difficult one to implement, however, it is a method used by humans and thus worth mentioning. It is also worth mentioning that false cognates will prove to be a stumbling block in CVA. The English word soap and the Spanish word sopa appear to be cognates; however sopa is the Spanish word for soup, not soap. These false cognates provide an additional layer of complexity to what would be the already difficult task of computing based upon cognates.

4 – Representing the Passage in SNePS

Once it had been established through the verbal protocols that there was enough contextual information in the chosen passage to be able to derive at least a basic understanding of what the word cantora meant, the next step in this study was to represent the passage in a SNePS semantic network. In order to accomplish this task, the passage had to be broken down into a set of basic information. This process is by no means automated at this point in the CVA project. A human is required to review a given passage and decide how they will choose to represent it as a semantic network. Just like the process of performing CVA, no two humans will necessary represent a given passage with the exact same semantic network. Allowing a computer to parse a given passage and convert it into such a network, however, is no small task, and is a topic for other papers and projects.

The seemingly simply passage chosen from A Clockwork Orange has a lot of information embedded within it. While a human reader may not consciously realize just how much of this information is present in the passage, for the semantic network to provide a full and complete representation of the passage all of this information must be included.

Before we delve into the information present in the passage, let us first define the case-frames used in this study, as well as their semantics. Due to the use of SNePSLOG in this study, all case frames will appear similar to predicates from first order logic.

lex case frame
thing-called(x)
[[thing-called(x)]] – x is the lexical name given to an object, concept, event, etc.

object/property case frame
object-property(x,y)
[[object-property(x,y)]] = x has the property y

member/class case frame
member-class(x,y)
[[member-class(x,y)]] = x is a member of the class y

agent/act case frame
agent-act(x,y)
[[agent-act(x,y)]] = x does the action y

obj1/rel/obj2 case frame
obj1-rel-obj2(x,y,z)
[[obj1-rel-obj2(x,y,z)]] = x is related to z by the relation y

action/object case frame
action-object(x,y)
[[action-object(x,y)]] = x is done on/to y

subclass/superclass case frame
subclass-superclass(x,y)
[[subclass-superclass(x,y)]] = x is a subclass of the superclass y

Now that we have defined a set of case frames to be used by SNePS, we can begin breaking our passage down into a set of the basic information presented.

"Then we came to a very neat like cantora…"

The first part of the passage will be broken into three separate pieces of information. First we must represent that some "we" did the action of coming to a cantora using the agent/act case frame.

agent-act(thing-called(we),action-object(thing-called(come),cantora1))!

The cantora is said to be neat, and as such we must represent that the cantora has this property.

object-property(cantora1, thing-called(neat))!

Finally, there is a third piece of information in this part of the passage that may not be consciously apparent to a human reader but which they understand while reading. The cantora mentioned is a specific cantora, a single instance of the overall class of cantoras. Therefore, we must represent the fact that this particular cantora is a member of the class of all the cantoras in the world.

member-class(cantora1,thing-called(cantora))!

You will see such representations throughout the semantic network built for this passage. In this case, cantora1 is a base node in the network, and could be thought of as a variable in a programming language. With this member/class assertion, we have told Cassie that cantora1 is a specific instance of the class cantora.

"…with typewriters and flowers on the desks…"

The next section of the passage requires some degree of thought before deciding upon a representation. The passage speaks of typewriters, flowers, and desks in the plural, but does not specify exactly how many of these objects there are in the cantora. There are two potential approaches to representing this portion of the passage. The first is a rule based approach. We would establish that for all x, where x is a desk in the cantora, there is a typewriter and flowers on top of x. This rule will represent the basic information required; there are desks with typewriters and flowers on them. However, this is not our only representation option. A second method, and the one chosen for this study, is the use of a mental model.

The overall idea behind mental models is more complex than we will necessarily have to discuss in this study. However, a few basic features of a mental model should be

noted. A mental model will only include what a person or agent believes to be true, not necessarily which is true in the real world. Because of this, each person or agent could have a different mental model about a given thing. The way in which the mental model is used in our representation of the passage could be far different if performed by another person. Mental models have a similar structure to what they represent, however they are simpler than the actual thing or concept. A mental model can be used to simplify a more complex concept and allows a person to reason about this simplified model and make predictions about the effect of a particular action on the model. (Mental Models for Producers)

As we've stated, the cantora in this passage has some number of typewriters and flowers on top of desks, but we do not know exactly how many of each. There could be eight desks, twelve typewriters, and ten flowers, or there could be twenty five desks, twenty five typewriters, and one hundred flowers. They question we have to ask ourselves is, does it really matter exactly how many of each there are? We understand that since the plural form of each noun is used, there must be at least two desks, each with at least one typewriter and one flower on it. For the representation of this passage, taking the minimum number of these objects, in this case two, should be sufficient. We can create representations of two separate desks in the cantora, each with its own typewriter and flower on it. In doing this, we have satisfied the requirements of the passage, there are desks, typewriters, and flowers, but in creating this mental model, we will then have base nodes for each of these individual objects so that we may talk about each one independent from the rest.

The representation of this portion of the passage is as follows:

The base nodes desk1 and desk2 are specific instances of the class desk.

member-class(desk1, thing-called(desk))!

member-class(desk2, thing-called(desk))!

In building the semantic network representation of the passage, the word "with (…*cantora with typewriters*…) was assumed to mean that the cantora contained these objects. As such, the cantora contains both desk1 and desk2.

obj1-rel-obj2(cantora1,thing-called(contains),desk1)!

obj1-rel-obj2(cantora1,thing-called(contains),desk2)!

The base nodes typewriter1 and typewriter2 are specific instances of the class typewriter.

member-class(typewriter1, thing-called(typewriter))!

member-class(typewriter2, thing-called(typewriter))!

Each typewriter is related to a desk by the "on" relation. Typewriter1 is on top of desk1 and typewriter2 is on top of desk2.

obj1-rel-obj2(typewriter1,thing-called(on),desk1)!

obj1-rel-obj2(typewriter2,thing-called(on),desk2)!

Base nodes flowers1 and flowers2 are specific instances of the class flowers.

member-class(flowers1, thing-called(flowers))!

member-class(flowers2, thing-called(flowers))!

The flowers are related to a desk by the "on" relation. Flowers1 is on top of desk1 and flowers2 is on top of desk2.

obj1-rel-obj2(flowers1,thing-called(on),desk1)!

obj1-rel-obj2(flowers2,thing-called(on),desk2)!

"…at the like chief desk the top millicent was sitting…"

The remainder of the passage is pretty straight forward in its representation. The chief desk is a specific instance of the desk class and has the property of being the chief desk. We also make the assumption that since we were talking about other desks that were contained within the cantora, that this chief desk is also inside the cantora, although this is not explicitly said.

member-class(chiefdesk1,thing-called(desk))!

object-property(chiefdesk1,thing-called(chiefdesk))!

obj1-rel-obj2(cantora1,thing-called(contains),chiefdesk1)!

We finish the presentation of the passage by representing the top millicent. Similar to the chief desk, the top millicent is a specific instance of the millicent class who has the property of being the top millicent. We must then represent that the top millicent is sitting at the chief desk.

member-class(topmillicent1,thing-called(millicent))!

object-property(topmillicent1,thing-called(topmillicent))!

agent-act(topmillicent1,action-object(thing-called(sitting),chiefdesk1))!

5 – Background Information

Now that the passage has been represented in a semantic network, the next step in the computerized CVA process is to provide our agent, Cassie, with the appropriate background knowledge she will need in order to derive a definition for the word cantora. Selecting what background information to represent is no small task. An average human reading this particular passage will have a wealth of background information regarding the various objects and actions occurring. They will have information about the specific objects in the passage, the desks, typewriters, and flowers. Likewise, information about police officers, what their jobs are, where they might work, etc. And finally, a human reader will have information regarding the actions and relations found in this passage, what it means for an object to be on top of another, as well as what it means for someone to be sitting at a desk.

How much or how little of this information should be represented? The average reader may know what a typewriter is, how it is used, and a base understanding of how it functions. However, a person who is an expert on typewriters will know significantly more information about typewriters, how each individual part inside it functions, the different kinds of typewriters, and perhaps even the history and the evolution of the typewriter. Choosing what to represent depends on our ultimate goal. For our passage, the history and specifics of the typewriter will be extraneous information, perhaps nice to have, but certainly not required.

Since we are dealing with only one passage in this study, and representing only the information required to derive the meaning of cantora, our set of background knowledge will be very small. In fact, when compared to the background knowledge of a

human reader, Cassie's will be miniscule. In this way, the background knowledge chosen is cherry-picked; we choose only to represent what is required for Cassie to derive a definition for our unknown word. The intention is that this information will allow Cassie to define cantora, however it may leave something to be desired when comparing it to how a human performs CVA. The knowledge a human has about a particular set of objects/concepts/actions is not cherry-picked in the way that Cassie's knowledge is. A human will have more information and rules that they will use to attempt to infer the meaning of the unknown word, which may result in contradictions or ambiguities that Cassie, in this CVA study, will not have to interpret. Obviously we can only tell Cassie so much information, so although her knowledge base will be small, the validity of the study will not be lost as she will still have to use this knowledge in an attempt to deduce the meaning of an unknown word, in our case, cantora.

Thinking back to our verbal protocols, the one piece of information that almost all of our human test subjects used to decide that a cantora was a form of an office was the fact that if something contains desks, and there are typewriters on these desks, then chances are we are talking about an office. This goes back to the idea of mental models. When people think of an office, they picture a desk, or group of desks, with typewriters, or more modernly computers, on the desks. While this mental model may not always be true, in general, it seems to be an accurate representation and, as such, is a good piece of background information to represent.

Informally, the rule we wish to define says that if a place has a desk with a typewriter on it, then that place is an office. More formally, we wish to say that for all x, y, z, and w, if x contains y, y is a desk, z is a typewriter, z is on y, and x is a member of

some class w, then w is a subclass of the superclass office. We take this rule and add it to Cassie's set of background knowledge as follows:

```
all(x,y,z,w)(
        {
                obj1-rel-obj2(x, thing-called(contains), y),
                member-class(y, thing-called(desk)),
                member-class(z, thing-called(typewriter)),
                obj1-rel-obj2(z, thing-called(on), y),
                member-class(x, w)
        }
        &=>
        subclass-superclass(w, thing-called(office))
        )!
```

Looking at the rule which Cassie now has in your knowledge base, we see that this is really the only piece of background knowledge we need. Remembering back to our representation of the passage, Cassie knows that cantora1 contains desk1, that typewriter1 is on top of desk1, and that cantora1 is a member of the class cantora. With these pieces of information, in combination with the rule just established, Cassie will be able to determine that the class cantora is a subclass of the class office. That is to say, a cantora *is* an office. It seems like this piece of background knowledge is all Cassie needs to accomplish the task of performing CVA on the passage and we have already established that it is a legitimate rule to believe through our verbal protocols.

Since this single rule is all we really need to derive a definition for cantora, we could stop here and not add any additional background information to the knowledge base. However, a few additional piece of information will not cause any harm and can be added to increase the overall understanding of the sentence. As in the case of the history of the typewriter, all of the following additional background knowledge is seemingly extraneous and as such, will only be mentioned briefly as they can be safely removed with no ill effect.

A few rules for the transitivity of "containing" have been added to Cassie's knowledge base. For all x, y, and z, if x contains y and z is sitting at y, then x contains z. Thus, in our passage, since the top millicent is sitting at the chief desk, and the cantora contains the chief desk, then the cantora also contains the top millicent.

```
all(x,y,z)(
        {
        obj1-rel-obj2(x,thing-called(contains),y),
        agent-act(z,action-object(thing-called(sitting),y)),
        }
        &=>
        obj1-rel-obj2(x,thing-called(contains),z)
        )!
```

Conversely, for all x, y, and z, if x contains z and z is sitting at y, then x contains y.

```
all(x,y,z)(
        {
        obj1-rel-obj2(x,thing-called(contains),z),
        agent-act(z,action-object(thing-called(sitting),y)),
        }
        &=>
        obj1-rel-obj2(x,thing-called(contains),y)
)!
```

Similar to the above rules, for all x, y, and z, if x contains y and z is on top of y, then x contains z. In our passage, the cantora contains all the desks, and therefore everything on top of the desks.

```
all(x,y,z)(
        {
        obj1-rel-obj2(x,thing-called(contains),y),
        obj1-rel-obj2(z,thing-called(on),y)
        }
        &=>
        obj1-rel-obj2(x,thing-called(contains),z)
)!
```

The addition of rules and facts to the knowledge base could continue ad infinitum. Obviously we have to choose a stopping point. A more complete knowledge base may containing much more information in this set of background knowledge, but for our study it seems as though we have met the required minimum amount of knowledge that must be represented.

5 – Results – Cassie's Inference

The passage has been represented in a semantic network and the necessary background knowledge has been added to the knowledge base. The final step is to ask Cassie what she believes the word cantora to mean:

```
^(snepsul::defineNoun "cantora")
Definition of cantora: nil
```

It appears as though Cassie can not infer the meaning of cantora from the representation of the passage and the background knowledge that we gave her. What went wrong? To a human it seems logical, if cantora is a subclass of office, then obviously a cantora is an office. Did our background knowledge work as intended? Does Cassie know that cantora is a subclass of office? Upon reviewing the semantic network created at the end of running the demo file, we find the following:

```
wff36!:  subclass-superclass(thing-called(cantora),thing-called(office))
```

This statement, wff36, was created by Cassie as she read the passage and applied the rule we placed in her background knowledge. The problem must lie elsewhere.

It turns out that the problem seems to arise from the difference in SNePSUL and SNePSLOG. The representation of the passage and background knowledge was writing in SNePSLOG while the noun definition algorithm that Cassie uses to attempt to derive the meaning of an unknown word was written in SNePSUL. Upon further investigation we find that if we attempt to use the SNePSUL find command on our representation in SNePSLOG, the command does not appear to work properly. If we ask, in SNePSUL as the noun definition algorithm does, whether cantora is a subclass of anything at all we are essentially told no, or, 'I do not know':

```
%(snepsul::find (compose superclass- ! subclass lex) cantora)
 CPU time : 0.00
```

Recall that wff36 clearly states that cantora is a subclass of office. It is evident that for one reason or another, the SNePSUL find, which the noun definition algorithm uses, does not work properly with SNePSLOG code. If we ask this same question to Cassie using SNePSLOG, she will give us a much different response:

```
subclass-superclass(thing-called(cantora),?x)?
wff36!:  subclass-superclass(thing-called(cantora),thing-called(office))
CPU time : 0.01
```

When the SNePSLOG command is used, Cassie is able to find wff36 in her network and tells us that cantora is indeed a subclass of office. It is for this reason that the actual call to the noun definition algorithm does not return office as we would expect, but instead returns nil. Further investigation into exactly why this occurs will be necessary if the current definition algorithm will be used in conjunction with SNePSLOG.

6 – Future Work – Short Term & Long Term

The most obvious shortcoming of this CVA study using a passage from A Clockwork Orange is that when Cassie attempts to derive a meaning for the word cantora, using the noun definition algorithm, she does not return an answer. It appears the she does not know what cantora means. However, we have proved that if we ask her to find a superclass of cantora, she will return office as an appropriate superclass. In this way, she does in fact know that a cantora is an office.

The next step in working with this passage would be to convert the entirety of the SNePSLOG code to SNePSUL, the code in which the noun definition algorithm is written, and then ask again what the definition of cantora is. Two things could happen upon doing this, the algorithm could still not work and return nothing, or the algorithm could reason that a cantora is an office. If the algorithm returns office when asked what cantora means, we will have established that there may be a problem in running the SNePSUL algorithm code with a SNePSLOG representation. If the algorithm does not return office, or anything for that matter, then the representation of the passage and background knowledge may be at fault. Although it appears to be an accurate representation, there may be some hidden problem that would have to be discovered.

In the long term, there are three areas of research and development that need to be addressed. Firstly, the problem of running SNePSUL commands, such as the find command, with SNePSLOG should be researched. With our SNePSLOG representations of the passage and the required background knowledge, we can ask the same question using both SNePSUL and SNePSLOG and get drastically different results. With the small research done for this study in the matter, it appears that in most cases using the

SNePSUL commands, like those found in the noun definition algorithm, do not work properly with code written in SNePSLOG. However, using the equivalent commands in SNePSLOG will produce the results that one would expect. Obviously there is some issue here and it may take a great deal of research to establish why this occurs.

Secondly, the definition algorithms being used by Cassie when she performs CVA were written some time ago and, as previously mentioned, are in SNePSUL. Although it would be a great deal of work, converting these algorithms into SNePSLOG may be a worthwhile undertaking. Converting these algorithms could provide insight into what is and is not working with the present algorithms. If, when converted to SNePSLOG, the algorithms work as intended on the representation of our passage, then we will be able to establish that the problem was simply that the algorithms were written in SNePSUL. However, if the newly formed SNePSLOG algorithms still did not work with our representation, we could assume that the knowledge base we built is not quite as we expect it to be. Either way, conversion of these algorithms to SNePSLOG would bring about a greater understanding of how the algorithms use the semantic network built for a given passage and the background knowledge associated with it.

The third and final area of research that could be addressed for the CVA project is by far the most complicated. Currently for each passage we want to represent in a SNePS semantic network, a knowledge engineer has to read the passage and determine how they will choose to build a representation. Creating an automated natural language parser that could read a passage and build a network representation, while a lofty goal, would take the CVA project one step closer to full automation and computerization.

References Cited

cse.buffalo.edu “CVA Project Description” December 12 2006

http://www.cse.buffalo.edu/~rapaport/CVA/cvadescription.html

Kelake.org “Mental Models for Producers and other more ordinary folk” December 12 2006

http://www.kelake.org/articles/mentalmodel/index.html

Soomka.com “Nadsat Dictionary” December 12 2006

http://soomka.com/nadsat.html

Wikipedia.org “Nadsat” December 12 2006

http://en.wikipedia.org/wiki/Nadsat

Wikipedia.org “SNePS” December 12 2006

http://en.wikipedia.org/wiki/SNePS

Appendix A – SNePS Code - cantora.demo

```
;=========================================================================
; FILENAME: cantora.demo
; DATE:           12/12/06
; PROGRAMMER:     Mark Zorn

;sentence

;"Then we came to a very neat like cantora with typewriters and flowers
; on the desks, and at the like chief desk the top millicent was
;sitting."

;From "A Clockwork Orange" by Anthony Burgess

;; this template version:
;; http://www.cse.buffalo.edu/~rapaport/CVA/snepslog-template-
;;2006114.demo

; Lines beginning with a semi-colon are comments.
; Lines beginning with "^" are Lisp commands.
; Lines beginning with "%" are SNePSUL commands.
; All other lines are SNePSLOG commands.
;
; To use this file: run SNePSLOG; at the SNePSLOG prompt (:), type:
;
;     demo "cantora.demo" av
;
; Make sure all necessary files are in the current working directory
; or else use full path names.
;
=========================================================================

; 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

; Load the appropriate definition algorithm:
^(cl:load "/home/unmdue/mrzorn/CSE663/CVA/defun_noun")

; Clear the SNePS network:
clearkb

; OPTIONAL:
; UNCOMMENT THE FOLLOWING CODE TO TURN FULL FORWARD INFERENCING ON:
;
^(cl:load "/projects/rapaport/CVA/STN2/ff")
```

```
; define frames here:
; ====================

;define lex case frame
define-frame thing-called(nil lex)

;define object property case frame
;[[object-property(x,y)]] = x has the property y
define-frame object-property(nil object property)

;define member class case frame
;[[member-class(x,y)]] = x is a member of the class y
define-frame member-class(nil member class)

;define agent act case frame
;[[agent-act(x,y)]] = x does the action y
define-frame agent-act(nil agent act)

;define object1 rel object2 case frame
;[[obj1-rel-obj2(x,y,z)]] = x is related to z by the relation y
define-frame obj1-rel-obj2(nil object1 rel object2)

;define action object case frame
;[[action-object(x,y)]] = x is done on/to y
define-frame action-object(nil action object)

;define subclass superclass case frame
;[[subclass-superclass(x,y)]] = x is a subclass of the superclass y
define-frame subclass-superclass(nil subclass superclass)

;define unused arc labels for the defineNoun function to work
%(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)

; define paths here:
; ====================
; (put annotated SNePSLOG code for your paths here;
; be sure to include both syntax and semantics;
; consult "/projects/rapaport/CVA/mkb3.CVA/paths/snepslog-paths"
; for the proper syntax and some suggested paths;
; be sure to define frames above for any paths that you need here!)

define-path class (compose class (kstar (compose subclass- !
superclass)))
```

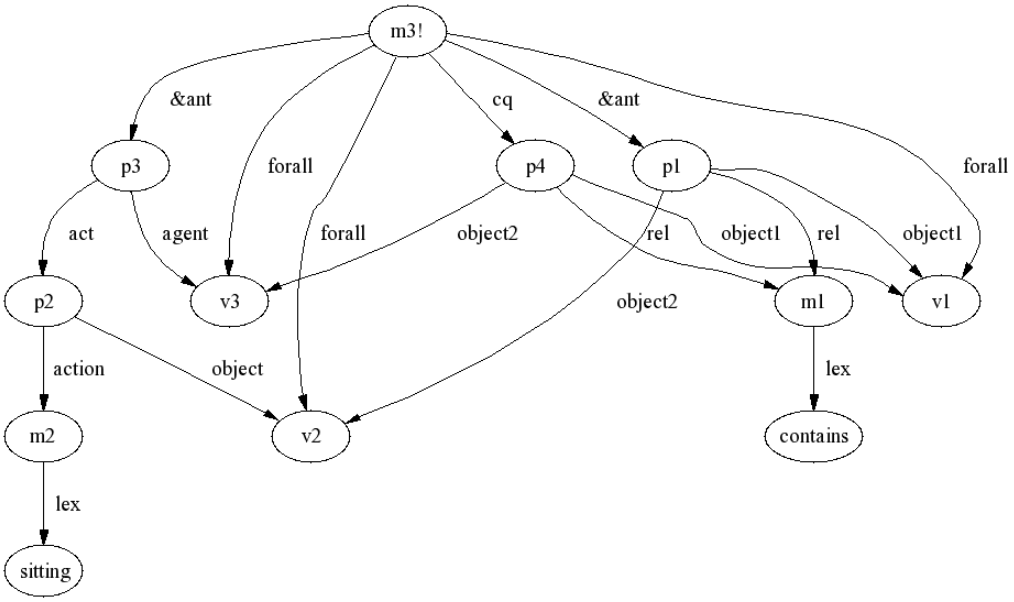
```
; BACKGROUND KNOWLEDGE:
; ====================

;if x contains y and z is sitting at y then x contains z
all(x,y,z)(
      {
      obj1-rel-obj2(x,thing-called(contains),y),
      agent-act(z,action-object(thing-called(sitting),y)),
      }
      &=>
      obj1-rel-obj2(x,thing-called(contains),z)
)!

;if x contains z and z is sitting at y then x contains y
all(x,y,z)(
      {
      obj1-rel-obj2(x,thing-called(contains),z),
      agent-act(z,action-object(thing-called(sitting),y)),
      }
      &=>
      obj1-rel-obj2(x,thing-called(contains),y)
)!

;if x contains y and z is ontop of y then x contains z
all(x,y,z)(
      {
      obj1-rel-obj2(x,thing-called(contains),y),
      obj1-rel-obj2(z,thing-called(on),y)
      }
      &=>
      obj1-rel-obj2(x,thing-called(contains),z)
)!

;millicents are people
subclass-superclass(thing-called(millicent),thing-called(people))!

;if x is a person and y is a desk and x is sitting at y then x is
working.
all(x,y,z)(
      {
      member-class(x,thing-called(person)),
      agent-act(x,action-object(thing-called(sitting),y)),
      member-class(y,thing-called(desk))
      }
      &=>
      agent-act(x,action-object(thing-called(work),y))
)!
```

```
;if x is working at y and y is a desk and x is a millicent
;and z contains y, then z is a policestation
all(x,y,z)(
      {
      agent-act(x,action-object(thing-called(work),y)),
      member-class(y,thing-called(desk)),
      member-class(x,thing-called(millicent)),
      obj1-rel-obj2(z,thing-called(contains),y)
      }
      &=>
      member-class(z,thing-called(policestation))
)!

;police stations are offices
subclass-superclass(thing-called(policestation),thing-called(office))!

;if x contains a desk and there is a typewriter on the desk
;then the class that x is a member of, is a subclass of office

all(x,y,z,w)(
      {
      obj1-rel-obj2(x,thing-called(contains),y),
      member-class(y, thing-called(desk)),
      member-class(z, thing-called(typewriter)),
      obj1-rel-obj2(z,thing-called(on),y),
      member-class(x,w)
      }
      &=>
      subclass-superclass(w, thing-called(office))
)!


; Cassie READS THE PASSAGE:
; =========================

;we came to a cantora
agent-act(thing-called(we),action-object(thing-called(come),
                              cantora1))!

;cantora1 is a cantora
member-class(cantora1,thing-called(cantora))!

;the cantora is neat
object-property(cantora1,thing-called(neat))!

;desk1 is a member of class desks
member-class(desk1, thing-called(desk))!

;desk2 is a member of class desks
member-class(desk2, thing-called(desk))!

;cantora contains desk 1
obj1-rel-obj2(cantora1,thing-called(contains),desk1)!

;cantora contains desk 2
obj1-rel-obj2(cantora1,thing-called(contains),desk2)!
```

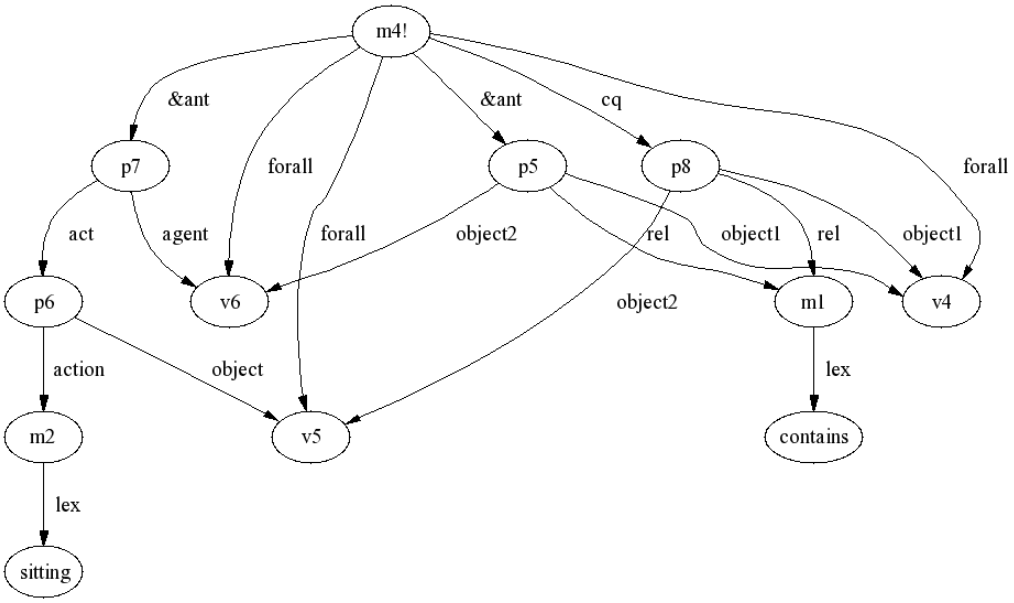
```
;typewrite1 is a member of class typewriter
member-class(typewriter1, thing-called(typewriter))!

;typewrite2 is a member of class typewriter
member-class(typewriter2, thing-called(typewriter))!

;typewriter1 is on desk1
obj1-rel-obj2(typewriter1,thing-called(on),desk1)!

;typewriter2 is on desk2
obj1-rel-obj2(typewriter2,thing-called(on),desk2)!

;flowers1 is a member of class flowers
member-class(flowers1, thing-called(flowers))!

;flowers2 is a member of class flowers
member-class(flowers2, thing-called(flowers))!

;flowers1 is on desk1
obj1-rel-obj2(flowers1,thing-called(on),desk1)!

;flowers2 is on desk2
obj1-rel-obj2(flowers2,thing-called(on),desk2)!

;the specific chief desk
member-class(chiefdesk1,thing-called(desk))!

;chiefdesk1 has the property of being the chief desk
object-property(chiefdesk1,thing-called(chiefdesk))!

;the cantora contains the chief desk
obj1-rel-obj2(cantora1,thing-called(contains),chiefdesk1)!

;top millicent
member-class(topmillicent1,thing-called(millicent))!

;topmillicent1 has the property of being the top millicent
object-property(topmillicent1,thing-called(topmillicent))!

;the top millicent was sitting at the chief desk
agent-act(topmillicent1,action-object(
            thing-called(sitting),chiefdesk1))!

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

;;Ask if Cassie knows if cantora is a subclass of anything.
;;snepsul find
%(snepsul::find (compose superclass- ! subclass lex) cantora)

;;Ask if Cassie knows if cantora is a subclass of anything.
;;snepslog find
subclass-superclass(thing-called(cantora),?x)?
```

Appendix B – SNePS Semantic Network Diagrams

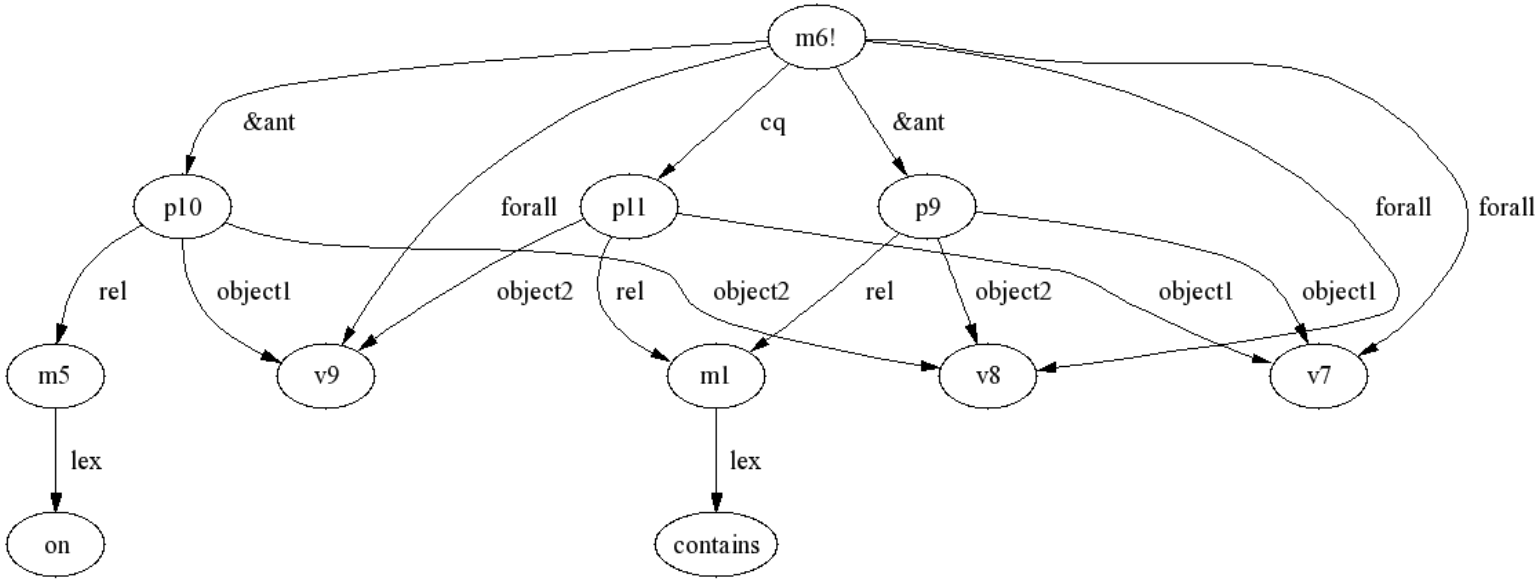
Background Knowledge Representation



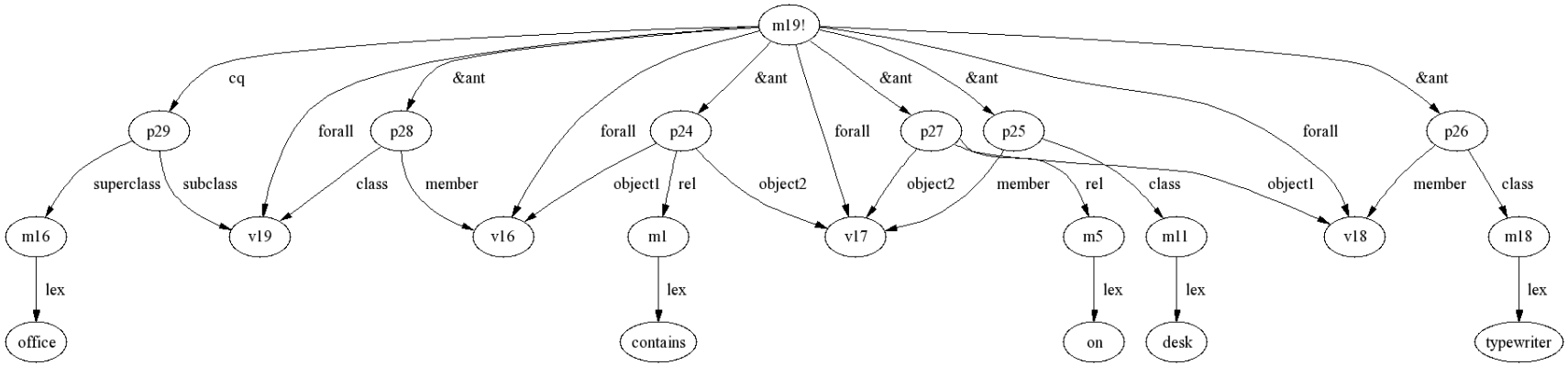
For all x, y, and z, if x contains y and z is sitting at y, then x contains z.



For all x, y, and z, if x contains z and z is sitting at y, then x contains y.

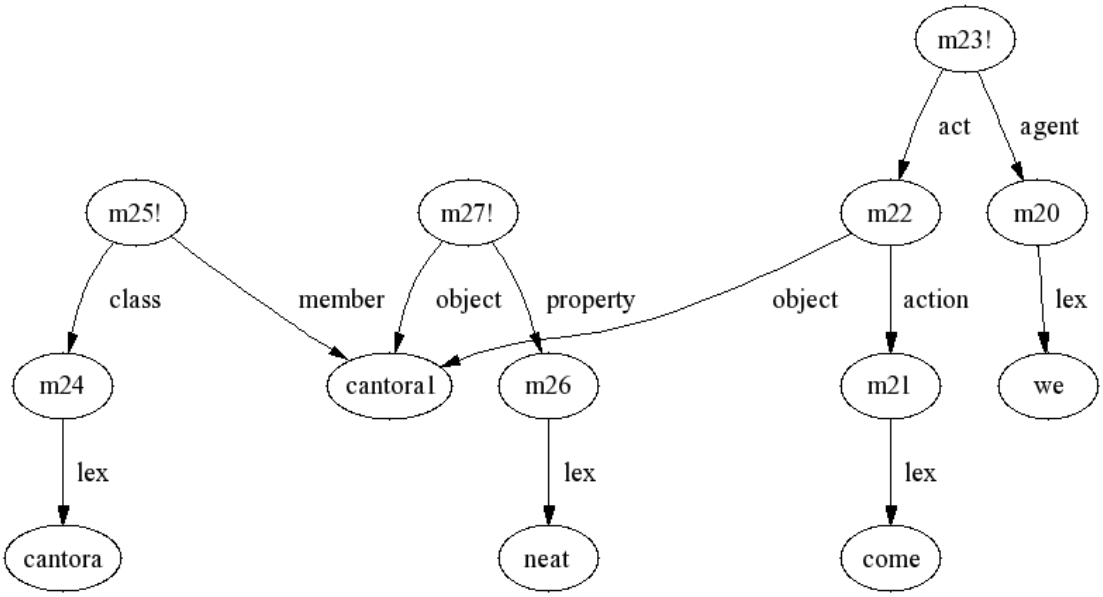


For all x, y, and z, if x contains y and z is on top of y, then x contains z.

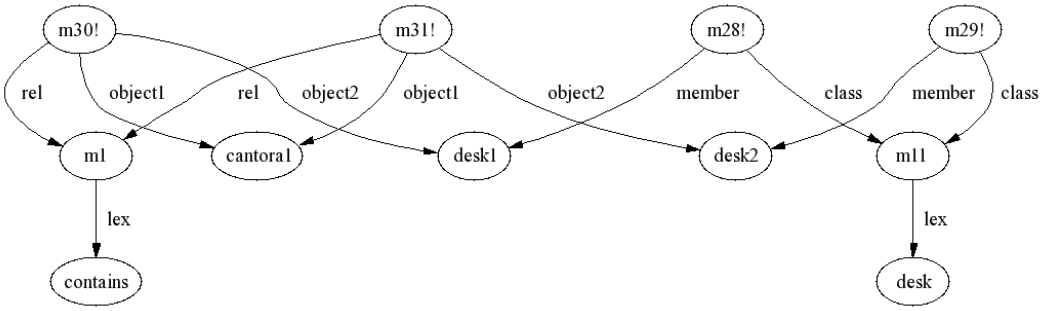


For all x, y, z, and w,
if x contains y,
y is a desk,
z is a typewriter,
z is on y,
and x is a member of some class w,
then w is a subclass of the superclass office.

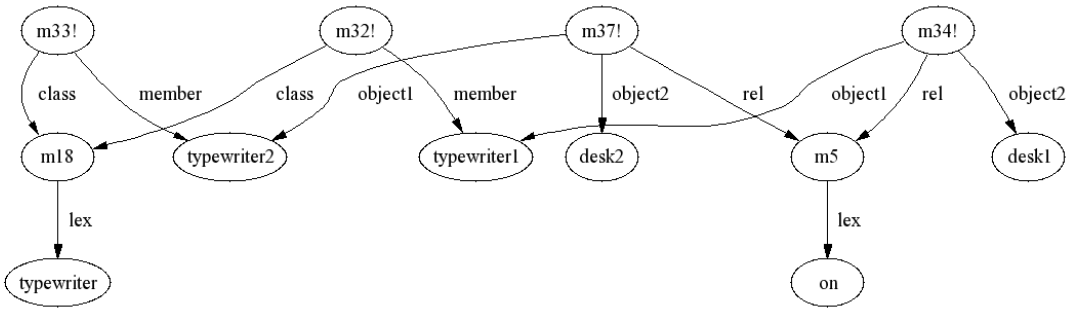
Passage Representation



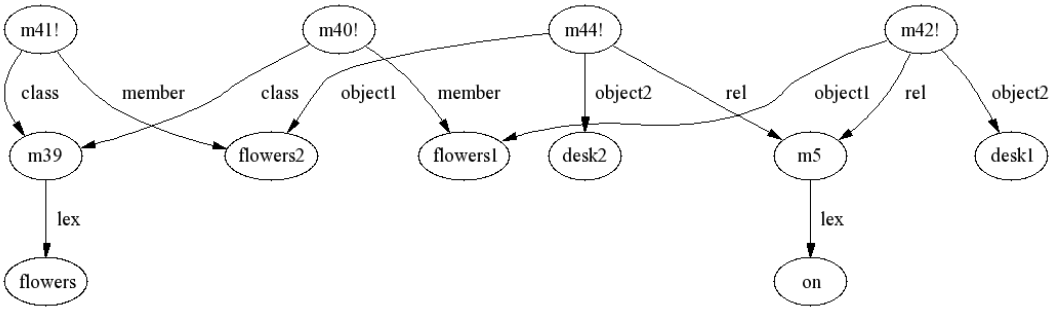
We came to a neat cantora.



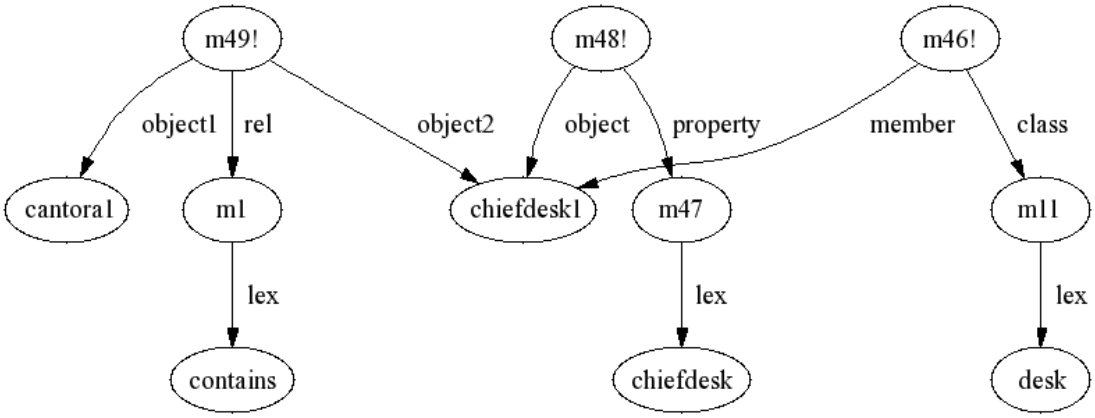
The cantora contains two desks.



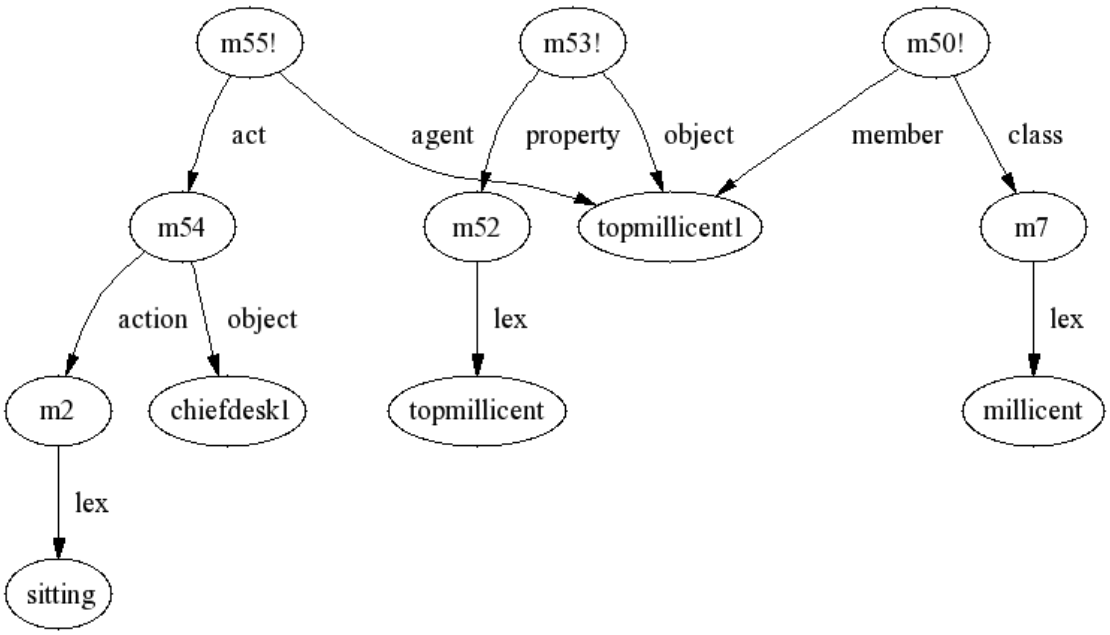
There is a typewriter on each desk.



There are flowers on each desk.

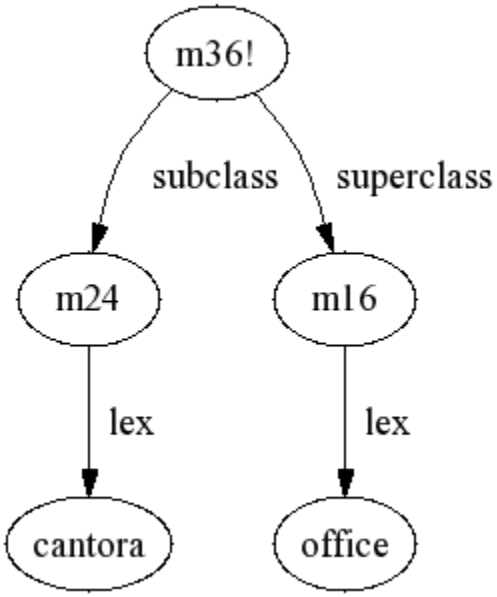


The cantora contains the chief desk.



The top millicent is sitting at the chief desk.

Inferences



Cantora is a subclass of office.

This portion of the network is inferred by Cassie based on her background knowledge and reading of the passage. This should allow her to give the definition of office to cantora.

Appendix C – Running cantora.demo

```
Script started on Tue Dec 12 19:03:43 2006
pollux {~/CSE663/CVA}  > acl International Allegro CL Enterprise
Edition
8.0 [Solaris] (Jun 30, 2006 12:35)
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): :ld /projects/shapiro/Sneps/sneps262
; Loading /projects/shapiro/Sneps/sneps262.cl
;   Loading /projects/shapiro/Sneps/Sneps262/load-sneps.lisp
;     Loading
;        /projects/snwiz/Install/Sneps-2.6.1/load-logical-
pathnames.lisp
Loading system SNePS...10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
SNePS-2.6 [PL:1a 2006/12/04 04:07:47] loaded.
Type `(sneps)' or `(snepslog)' to get started.
cl-user(2): (snepslog)


   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 "cantora.demo" av

File /home/unmdue/mrzorn/CSE663/CVA/cantora.demo is now the source of
input.

  The demo will pause between commands, at that time press
  RETURN to continue, or ? to see a list of available commands


 CPU time : 0.05

: ;
================================================================================
; FILENAME:     cantora.demo ; DATE:         12/12/06 ; PROGRAMMER:
Mark Zorn


 CPU time : 0.00
```

```
: ;sentence


 CPU time : 0.00

: ;"Then we came to a very neat like cantora with typewriters and
flowers on ;the desks, and at the like chief desk the top millicent was
sitting."


 CPU time : 0.00

: ;From "A Clockwork Orange" by Anthony Burgess


 CPU time : 0.00

: ;; this template version:      ;;
http://www.cse.buffalo.edu/~rapaport/CVA/snepslog-template-2006114.demo


 CPU time : 0.00

: ; Lines beginning with a semi-colon are comments. ; Lines beginning
with "^" are Lisp commands. ; Lines beginning with "%" are SNePSUL
commands. ; All other lines are SNePSLOG commands. ; ; To use this
file: run SNePSLOG; at the SNePSLOG prompt (:), type: ; ;       demo
"cantora.demo" av ;        ; Make sure all necessary files are in the
current working directory ; or else use full path names. ;
======================================================================


 CPU time : 0.00

: ; Set SNePSLOG mode = 3 set-mode-3

Net reset
In SNePSLOG Mode 3.
Use define-frame <pred> <list-of-arc-labels>.

achieve(x1) will be represented by {<action, achieve>, <object1, x1>}
ActPlan(x1, x2) will be represented by {<act, x1>, <plan, x2>}
believe(x1) will be represented by {<action, believe>, <object1, x1>}
disbelieve(x1) will be represented by {<action, disbelieve>, <object1,
x1>}
adopt(x1) will be represented by {<action, adopt>, <object1, x1>}
unadopt(x1) will be represented by {<action, unadopt>, <object1, x1>}
do-all(x1) will be represented by {<action, do-all>, <object1, x1>}
do-one(x1) will be represented by {<action, do-one>, <object1, x1>}
Effect(x1, x2) will be represented by {<act, x1>, <effect, x2>}
else(x1) will be represented by {<else, x1>}
GoalPlan(x1, x2) will be represented by {<goal, x1>, <plan, x2>}
if(x1, x2) will be represented by {<condition, x1>, <then, x2>}
ifdo(x1, x2) will be represented by {<if, x1>, <do, x2>}
Precondition(x1, x2) will be represented by {<act, x1>, <precondition,
x2>}
snif(x1) will be represented by {<action, snif>, <object1, x1>}
```

```
sniterate(x1) will be represented by {<action, sniterate>, <object1,
x1>}
snsequence(x1, x2) will be represented by {<action, snsequence>,
<object1, x1>, <object2, x2>}
whendo(x1, x2) will be represented by {<when, x1>, <do, x2>}
wheneverdo(x1, x2) will be represented by {<whenever, x1>, <do, x2>}
withall(x1, x2, x3, x4) will be represented by {<action, withall>,
<vars, x1>, <suchthat, x2>, <do, x3>, <else, x4>}
withsome(x1, x2, x3, x4) will be represented by {<action, withsome>,
<vars, x1>, <suchthat, x2>, <do, x3>, <else, x4>}


 CPU time : 0.01




 CPU time : 0.00

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


 CPU time : 0.00




 CPU time : 0.00

: ; Load the appropriate definition algorithm: ^(cl:load
"/home/unmdue/mrzorn/CSE663/CVA/defun_noun")
t


 CPU time : 0.24




 CPU time : 0.00

: ; Clear the SNePS network: clearkb
Knowledge Base Cleared


 CPU time : 0.00




 CPU time : 0.00

: ; OPTIONAL: ; UNCOMMENT THE FOLLOWING CODE TO TURN FULL FORWARD
INFERENCING ON: ; ^(cl:load "/projects/rapaport/CVA/STN2/ff")
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


 CPU time : 0.01




 CPU time : 0.00

: ; define frames here: ; ====================


 CPU time : 0.00

: ;define lex case frame define-frame thing-called(nil lex)
thing-called(x1) will be represented by {<lex, x1>}


 CPU time : 0.00




 CPU time : 0.00

: ;define object property case frame ;[[object-property(x,y)]] = x has
the property y define-frame object-property(nil object property)
object-property(x1, x2) will be represented by {<object, x1>,
<property, x2>}


 CPU time : 0.00




 CPU time : 0.00

: ;define member class case frame ;[[member-class(x,y)]] = x is a
member of the class y define-frame member-class(nil member class)
member-class(x1, x2) will be represented by {<member, x1>, <class, x2>}


 CPU time : 0.00




 CPU time : 0.00

: ;define agent act case frame ;[[agent-act(x,y)]] = x does the action
y define-frame agent-act(nil agent act)
agent-act(x1, x2) will be represented by {<agent, x1>, <act, x2>}
```

```
 CPU time : 0.00



 CPU time : 0.00

: ;define object1 rel object2 case frame ;[[obj1-rel-obj2(x,y,z)]] = x
is related to z by the relation y define-frame obj1-rel-obj2(nil
object1 rel object2)
obj1-rel-obj2(x1, x2, x3) will be represented by {<object1, x1>, <rel,
x2>, <object2, x3>}


 CPU time : 0.00



 CPU time : 0.00

: ;define action object case frame ;[[action-object(x,y)]] = x is done
on/to y define-frame action-object(nil action object)
action-object(x1, x2) will be represented by {<action, x1>, <object,
x2>}


 CPU time : 0.00



 CPU time : 0.00

: ;define subclass superclass case frame ;[[subclass-superclass(x,y)]]
= x is a subclass of the superclass y define-frame subclass-
superclass(nil subclass superclass)
subclass-superclass(x1, x2) will be represented by {<subclass, x1>,
<superclass, x2>}


 CPU time : 0.00



 CPU time : 0.00

: ;define unused arc labels for the defineNoun function to work
%(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)
 class is already defined.
 member is already defined.
```

```
 CPU time : 0.00



 CPU time : 0.00

: ; define paths here: ; ===================== ; (put annotated
SNePSLOG code for your paths here; ; be sure to include both syntax and
semantics; ; consult "/projects/rapaport/CVA/mkb3.CVA/paths/snepslog-
paths" ; for the proper syntax and some suggested paths; ; be sure to
define frames above for any paths that you need here!)


 CPU time : 0.00

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


 CPU time : 0.00




 CPU time : 0.00

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


 CPU time : 0.00

: ;if x contains y and z is sitting at y then x contains z all(x,y,z)(
{         obj1-rel-obj2(x,thing-called(contains),y),          agent-
act(z,action-object(thing-called(sitting),y)),         }         &=>
obj1-rel-obj2(x,thing-called(contains),z) )!

  wff3!:  all(z,y,x)({agent-act(z,action-object(thing-
called(sitting),y)),obj1-rel-obj2(x,thing-called(contains),y)} &=>
{obj1-rel-obj2(x,thing-called(contains),z)})

 CPU time : 0.00




 CPU time : 0.00

: ;if x contains z and z is sitting at y then x contains y all(x,y,z)(
{         obj1-rel-obj2(x,thing-called(contains),z),          agent-
```

```
act(z,action-object(thing-called(sitting),y)),         }         &=>
obj1-rel-obj2(x,thing-called(contains),y) )!

  wff4!:  all(z,y,x)({agent-act(z,action-object(thing-
called(sitting),y)),obj1-rel-obj2(x,thing-called(contains),z)} &=>
{obj1-rel-obj2(x,thing-called(contains),y)})

 CPU time : 0.00



 CPU time : 0.00

: ;if x contains y and z is ontop of y then x contains z all(x,y,z)(
{         obj1-rel-obj2(x,thing-called(contains),y),         obj1-rel-
obj2(z,thing-called(on),y)         }         &=>         obj1-rel-
obj2(x,thing-called(contains),z) )!

  wff6!:  all(z,y,x)({obj1-rel-obj2(z,thing-called(on),y),obj1-rel-
obj2(x,thing-called(contains),y)} &=> {obj1-rel-obj2(x,thing-
called(contains),z)})

 CPU time : 0.01



 CPU time : 0.00

: ;millicents are people         subclass-superclass(thing-
called(millicent),thing-called(people))!

  wff9!:  subclass-superclass(thing-called(millicent),thing-
called(people))

 CPU time : 0.00



 CPU time : 0.00

: ;if x is a person and y is a desk and x is sitting at y then x is
working. all(x,y,z)(         {         member-class(x,thing-
called(person)),         agent-act(x,action-object(thing-
called(sitting),y)),         member-class(y,thing-called(desk))
}         &=>         agent-act(x,action-object(thing-called(work),y))
)!

  wff13!:  all(z,y,x)({member-class(y,thing-called(desk)),agent-
act(x,action-object(thing-called(sitting),y)),member-class(x,thing-
called(person))} &=> {agent-act(x,action-object(thing-
called(work),y))})

 CPU time : 0.01
```

```
 CPU time : 0.00



 CPU time : 0.00

: ;if x is working at y and y is a desk and x is a millicent  ;and z
contains y, then z is a policestation all(x,y,z)(         {
agent-act(x,action-object(thing-called(work),y)),         member-
class(y,thing-called(desk)),         member-class(x,thing-
called(millicent)),         obj1-rel-obj2(z,thing-called(contains),y)
}         &=>         member-class(z,thing-called(policestation)) )!

  wff15!:  all(z,y,x)({obj1-rel-obj2(z,thing-
called(contains),y),member-class(x,thing-called(millicent)),member-
class(y,thing-called(desk)),agent-act(x,action-object(thing-
called(work),y))} &=> {member-class(z,thing-called(policestation))})

 CPU time : 0.06



 CPU time : 0.00

: ;police stations are offices subclass-superclass(thing-
called(policestation),thing-called(office))!

  wff17!:  subclass-superclass(thing-called(policestation),thing-
called(office))

 CPU time : 0.00



 CPU time : 0.00

: ;if x contains a desk and there is a typewriter on the desk  ;then
the class that x is a member of, is a subclass of office


 CPU time : 0.00

: all(x,y,z,w)(         {         obj1-rel-obj2(x,thing-
called(contains),y),         member-class(y, thing-called(desk)),
member-class(z, thing-called(typewriter)),         obj1-rel-
obj2(z,thing-called(on),y),         member-class(x,w)         }
&=>         subclass-superclass(w, thing-called(office)) )!

  wff19!:  all(w,z,y,x)({member-class(x,w),obj1-rel-obj2(z,thing-
called(on),y),member-class(z,thing-called(typewriter)),member-
```

```
class(y,thing-called(desk)),obj1-rel-obj2(x,thing-called(contains),y)}
&=> {subclass-superclass(w,thing-called(office))})

 CPU time : 0.06




 CPU time : 0.00




 CPU time : 0.00

: ; Cassie READS THE PASSAGE: ; ========================


 CPU time : 0.00

: ;we came to a cantora agent-act(thing-called(we),action-object(thing-
called(come),                                          cantora1))!

  wff23!:  agent-act(thing-called(we),action-object(thing-
called(come),cantora1))

 CPU time : 0.01




 CPU time : 0.00

: ;cantora1 is a cantora member-class(cantora1,thing-called(cantora))!

  wff25!:  member-class(cantora1,thing-called(cantora))

 CPU time : 0.01




 CPU time : 0.00

: ;the cantora is neat object-property(cantora1,thing-called(neat))!

  wff27!:  object-property(cantora1,thing-called(neat))

 CPU time : 0.00




 CPU time : 0.00

: ;desk1 is a member of class desks member-class(desk1, thing-
called(desk))!
```

```
  wff28!:  member-class(desk1,thing-called(desk))

 CPU time : 0.01



 CPU time : 0.00

: ;desk2 is a member of class desks member-class(desk2, thing-
called(desk))!

  wff29!:  member-class(desk2,thing-called(desk))

 CPU time : 0.05



 CPU time : 0.00

: ;cantora contains desk 1 obj1-rel-obj2(cantora1,thing-
called(contains),desk1)!

  wff30!:  obj1-rel-obj2(cantora1,thing-called(contains),desk1)

 CPU time : 0.02



 CPU time : 0.00

: ;cantora contains desk 2 obj1-rel-obj2(cantora1,thing-
called(contains),desk2)!

  wff31!:  obj1-rel-obj2(cantora1,thing-called(contains),desk2)

 CPU time : 0.01



 CPU time : 0.00

: ;typewrite1 is a member of class typewriter member-class(typewriter1,
thing-called(typewriter))!

  wff32!:  member-class(typewriter1,thing-called(typewriter))

 CPU time : 0.01



 CPU time : 0.00
```

```
: ;typewrite2 is a member of class typewriter member-class(typewriter2,
thing-called(typewriter))!

  wff33!:  member-class(typewriter2,thing-called(typewriter))

 CPU time : 0.01



 CPU time : 0.00

: ;typewriter1 is on desk1 obj1-rel-obj2(typewriter1,thing-
called(on),desk1)!

  wff36!:  subclass-superclass(thing-called(cantora),thing-
called(office))
  wff35!:  obj1-rel-obj2(cantora1,thing-called(contains),typewriter1)
  wff34!:  obj1-rel-obj2(typewriter1,thing-called(on),desk1)

 CPU time : 0.07



 CPU time : 0.00

: ;typewriter2 is on desk2 obj1-rel-obj2(typewriter2,thing-
called(on),desk2)!

  wff38!:  obj1-rel-obj2(cantora1,thing-called(contains),typewriter2)
  wff37!:  obj1-rel-obj2(typewriter2,thing-called(on),desk2)
  wff36!:  subclass-superclass(thing-called(cantora),thing-
called(office))

 CPU time : 0.03



 CPU time : 0.00

: ;flowers1 is a member of class flowers member-class(flowers1, thing-
called(flowers))!

  wff40!:  member-class(flowers1,thing-called(flowers))

 CPU time : 0.00



 CPU time : 0.00

: ;flowers2 is a member of class flowers member-class(flowers2, thing-
called(flowers))!
```

```
  wff41!:  member-class(flowers2,thing-called(flowers))

 CPU time : 0.00



 CPU time : 0.00

: ;flowers1 is on desk1 obj1-rel-obj2(flowers1,thing-called(on),desk1)!

  wff43!:  obj1-rel-obj2(cantora1,thing-called(contains),flowers1)
  wff42!:  obj1-rel-obj2(flowers1,thing-called(on),desk1)

 CPU time : 0.02



 CPU time : 0.00

: ;flowers2 is on desk2 obj1-rel-obj2(flowers2,thing-called(on),desk2)!

  wff45!:  obj1-rel-obj2(cantora1,thing-called(contains),flowers2)
  wff44!:  obj1-rel-obj2(flowers2,thing-called(on),desk2)

 CPU time : 0.04



 CPU time : 0.00

: ;the specific chief desk member-class(chiefdesk1,thing-called(desk))!

  wff46!:  member-class(chiefdesk1,thing-called(desk))

 CPU time : 0.01



 CPU time : 0.00

: ;chiefdesk1 has the property of being the chief desk object-
property(chiefdesk1,thing-called(chiefdesk))!

  wff48!:  object-property(chiefdesk1,thing-called(chiefdesk))

 CPU time : 0.01



 CPU time : 0.00
```

```
: ;the cantora contains the chief desk obj1-rel-obj2(cantora1,thing-
called(contains),chiefdesk1)!

  wff49!:  obj1-rel-obj2(cantora1,thing-called(contains),chiefdesk1)

 CPU time : 0.02




 CPU time : 0.00

: ;top millicent member-class(topmillicent1,thing-called(millicent))!

  wff51!:  member-class(topmillicent1,thing-called(people))
  wff50!:  member-class(topmillicent1,thing-called(millicent))

 CPU time : 0.02




 CPU time : 0.00

: ;topmillicent1 has the property of being the top millicent object-
property(topmillicent1,thing-called(topmillicent))!

  wff53!:  object-property(topmillicent1,thing-called(topmillicent))

 CPU time : 0.00




 CPU time : 0.00

: ;the top millicent was sitting at the chief desk agent-
act(topmillicent1,action-object(                thing-
called(sitting),chiefdesk1))!

  wff56!:  obj1-rel-obj2(cantora1,thing-called(contains),topmillicent1)
  wff55!:  agent-act(topmillicent1,action-object(thing-
called(sitting),chiefdesk1))
  wff49!:  obj1-rel-obj2(cantora1,thing-called(contains),chiefdesk1)

 CPU time : 0.06




 CPU time : 0.00

: ; Ask Cassie what "cantora" means: ^(snepsul::defineNoun "cantora")
 Definition of cantora: nil


 CPU time : 0.05
```

```
 CPU time : 0.00

: ;;Ask if Cassie knows if cantora is a subclass of anything. - snepsul
find %(snepsul::find (compose superclass- ! subclass lex) cantora)


 CPU time : 0.00




 CPU time : 0.00

: ;;Ask if Cassie knows if cantora is a subclass of anything. -
snepslog find subclass-superclass(thing-called(cantora),?x)?

  wff36!:  subclass-superclass(thing-called(cantora),thing-
called(office))

 CPU time : 0.01

:

End of /home/unmdue/mrzorn/CSE663/CVA/cantora.demo demonstration.


 CPU time : 1.06

: lisp

Bye
nil
cl-user(3): :ex
; Exiting
pollux {~/CSE663/CVA}  > ^D__exit

script done on Tue Dec 12 19:05:42 2006
```