Demonstrating Contexual Vocabulary Acquisition by Computing the meaning of *sardonic* using CASSIE

Shane Axtell

CSE 663

axtell@cse.buffalo.edu

December 15, 2006

Abstract

In the Contextual Vocabulary Acquisition (CVA) project, we seek to develop a set of heuristics that can be taught to students to increase their capacity to learn new words in context. We employ SNePS to create a semantic network that a computational intelligent agent can use to infer dictionary-like definitions for difficult words encountered in reading passages. In this paper, CASSIE (a computational agent) is used to infer the meaning of the word *sardonic* in a short passage. Information extracted from protocols with human subjects was used to create a knowledge base that contains necessary background knowledge and a representation of the sentence containing the difficult word. CASSIE was then allowed to make inferences based on the knowledge base and the representation. CASSIE successfully developed a dictionary-like definition for one sense of the target word. Immediate next steps include expanding the representation of the sentence chosen for this project and creating an algorithm for adjectives that can be used to package the inferences that CASSIE makes in a more human readable form. Longer-term goals might include analyzing and synthesizing the projects already done so that the needed heuristics can be developed and taught to students.

1 CVA Project and SNePS

Generally speaking, as human beings we do not take the time to look up every new word we come across in our reading. Yet we are still able to extract enough information from the surrounding text to be able to create *definitions* for the new words that we encounter. This general technique (i.e., developing a definition for a new word from the context in which it occurs) is called contextual vocabulary acquisition (CVA).

Two pieces of evidence lead one to believe that the majority of words that a person learns are acquired through contextual experience with those words. First, children learn to understand and speak their first language(s) entirely in context. They listen to their parents and repeat the sounds that they hear until they eventually learn to map those sounds onto individual meanings. Second, adults know the meanings of more words than they could have possibly been taught in school (Rapaport and Kibby 2002). Rapaport and Kibby (2002) cite the following findings:

> The *average* number of word families (e.g., 'help', 'helps', 'helped', 'helping', 'helper', 'helpless', 'helpful' are one word family) known by high school graduates is estimated at between 45,000 [55] and 60,000 [54]. *Excellent* students who read a great deal may know 120,000 word families [54]. (Rapaport and Kibby 2002: 4)

These authors state further that knowing as few as 45,000 words by age 18 means having acquired 2500 per year while no more that 400 words are explicitly taught in schools (i.e., 4800 words by the end of high school) (Rapaport and Kibby 2002). Given the foregoing averages, we learn roughly 90% of our vocabulary from written and oral contexts (Rapaport and Kibby 2002). Therefore, it must be the case that humans learn words in context more so than in explicit teaching environments such as grade school.

Since 2001, researchers William J. Rapaport (University at Buffalo, Department of Computer Science and Engineering) and Michael W. Kibby (University at Buffalo, Department of Learning and Instruction) have been collaborating on the CVA project. These researchers believe that the computational models of CVA produced by using computational intelligent agents to create definitions from contextual cues will help develop the needed heuristics that can be taught to school-aged children (and students of all ages) to increase their capacity to learn new concepts and approach various genres of text with confidence.

Previous efforts to develop curricula for teaching vocabulary acquisition by context have not heretofore produced solid, algorithmic steps. One such example is that of Clarke and Nation (1980)(as cited in Rapaport and Kibby 2002). Their suggested route to discovering the meaning of a word using context includes the following:

- look at the word itself and its surroundings to decide on the part of speech
- look at the immediate grammar context of the word, usually within a clause or sentence
- look at the wider context of the word usually beyond the level of the clause and often over several sentences
- *guess ... the word* and check ... that the guess is correct

The last item is of particular interest to the CVA project. Telling human beings to *guess* the meaning of the target word after giving several detailed steps makes the preceding steps defeasible. Why use them if all you are going to do is *guess*? Given the ubiquitous nature of CVA there must be some more informative way of discovering the desired meanings.

In the CVA project, we use SNePS as a model of the human mind. SNePS is a network-based, computational knowledge representation and reasoning system (Shapiro 2004; Rapaport 2006). According to Shapiro (2004),

> a semantic network is a labeled directed graph in which nodes represent entities, arc labels represent binary relations, and an arc labeled R going from node n to node m represents the fact that the entity represented by n bears the relation represented by R to the entity represented by m.

In addition, SNePS may be thought of as a propositional semantic network because each node in the network represents a proposition (Shapiro 2004). The directed arcs coming out of each node represent the relations between the nodes and can be thought of as the syntactic structure of their associated nodes (Shapiro 2004). A computational intelligent agent, such as CASSIE[1], can use SNIP (the SNePS Inference Package) to infer new propositions from this semantic network. A knowledge base with inference rules must be created and be a part of the semantic network to enable CASSIE to make the inferences (Shapiro 2004). This project is the result of creating such a knowledge base and allowing CASSIE to make inferences that lead to a *definition*-like meaning for an unknown word.

Using a computational agent[2] to infer definitions to an unknown word will assist in developing heuristics that can be used in teaching CVA to humans. This is beneficial to the overall CVA project because it will lead to better instructions than to *guess*. Telling a computational agent to *guess* will not produce any useful output. In order for an intelligent agent to extract any useful information from the text it must be given a solid, realistic algorithm to follow. Using the computational models that are created by having intelligent agents infer the meanings of unknown words we can develop solid, realistic heuristics that can be taught to humans. This will enable them to approach difficult literature with confidence.

2 The Passage

Developing a curriculum for doing CVA that is based on algorithmic steps and that can be taught to students is a large and long-term research endeavor. The purpose of this paper is to design a knowledge base that contains all and only the necessary information (in the form of rules of inference and the SNePS representation of a sentence) needed to produce a contextual definition for one difficult word in one small context. It is hoped that the results from this smaller project may be useful in defining heuristics for discovering the meanings of adjectives that are unknown to human reader.

The word and its context chosen for this paper are given in the following sentence:

A dear and now deceased friend said to me years ago when I had said something sardonic,

[1]CASSIE is the name of the intelligent agent used for this project. Essentially, CASSIE consists of the knowledge base created for this project and she uses SNIP and SNePS to derive the meaning of the unknown word.

[2]The terms*computational intelligent agent*, *intelligent agent*, and *computational agent* are used interchangeably in this paper.

"You could have gone all day without saying that." His one-liner reproof was lovingly stated, illustrating how correction can be an act of affection. (Maxwell 2004)

The target word for this study is the adjective *sardonic*.

Much of the information contained in the above sentences is superfluous to the task of defining the target word. For example, tense and aspect have no bearing whatever on the contextual meaning of *sardonic* (i.e., *sardonic* could occur in sentences with verbs that have any tense or aspect marking and this would not change the meaning intended for *sardonic*). Therefore, this information was essentially ignored in producing the knowledge base rules and representing the sentences. Additionally, the fact that the *friend* spoken of in the first sentence is *dear* and *now deceased* is also superfluous to the meaning of *sardonic*. This information was, therefore, also ignored. Finally, the phrase "You could have gone all day without saying that" is itself a reproof and a correction in the sentence. This makes it non-essential to the meaning of *sardonic* given that the words *reproof* and *correction* are used later.

Based on the observation that much of the information contained in the original sentences is superfluous, the sentences were modified and condensed to the following:

A friend reproved and corrected me when I said something sardonic.

This modified sentence is structurally very different from the original sentences but it maintains the essence of the original meaning that is being targeted in this project. Removing this information from the original sentences does not alter the meaning of *sardonic* within the specified context.

3 Verbal Protocols

The original version of the sentences was given to five human subjects to see what steps they would take in determining the meaning of *sardonic*. For each subject, the sentence was displayed on a computer screen using a text editor. The subjects were told that they would see a sentence on the computer screen that contained a difficult word. The subjects were then instructed to read the sentence aloud and try to discern the meaning of the difficult word. They were told to "think out loud" as they thought through the possible meanings.

In the list below are given some of the comments that each subject made as they "thought out loud" about what *sardonic* might mean:

- Subject 1: ". . . the phrase 'You could have gone all day without saying that' kind of gives me the indication that it has that sense [that it's something you regret having said]"
- Subject 2: "the way that the reprimand is phrased, 'You could have gone all day without saying that', means that it probably was also completely unnecessary to say"
- Subject 3: "It's something unnecessary"
- Subject 4: "something insulting or dark . . . like not optimistic . . . something pessimistic"
- Subject 5: "there's a negative connotation to it . . . it's something that shouldn't have been said or uncalled for"

All of the subjects seemed to agree that *sardonic* has a negative connotation and that it is most likely something that should not have been said in the first place. They all came to that conclusion after combining their interpretation of the phrase "You could have gone all day without saying that" with the speaker's use of the words *one-liner reproof* and *correction.*

4 A SNePS Representation of the Sentence

4.1 Syntax and Semantics of the SNePS Case Frames

Since *sardonic* is an adjective and no CVA algorithm currently exists for adjectives, the choice of case frames was quite open. However, three out of the five case frames used in this project are typical CVA case frames and the syntax and semantics for these are given on the CVA web site[3].

The case frames for this project are:

- *lex*
- *agent/act/action/object*

[3]`http://www.cse.buffalo.edu/~rapaport/CVA/CaseFrames/case-frames/`

- *object/property*
- *mod/object*
- *effect/cause*

The syntax and semantics of the last two case frames are given below:

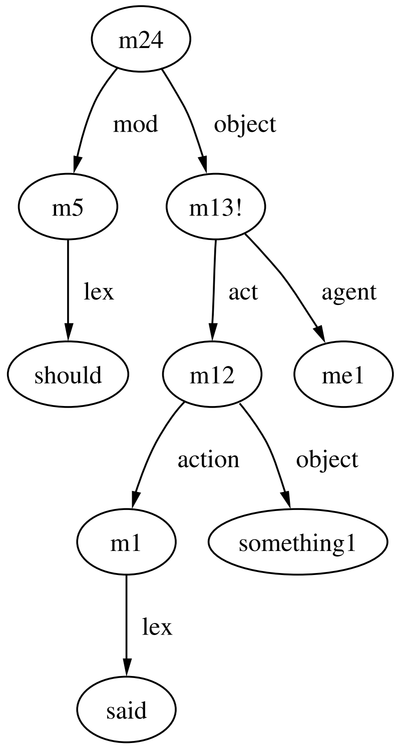

Figure 1: Case frame syntax for *mod/object*

```
mod/object = [[m24]] is the proposition that the modifier [[m5]]
             modifies the object [[m13!]].
```

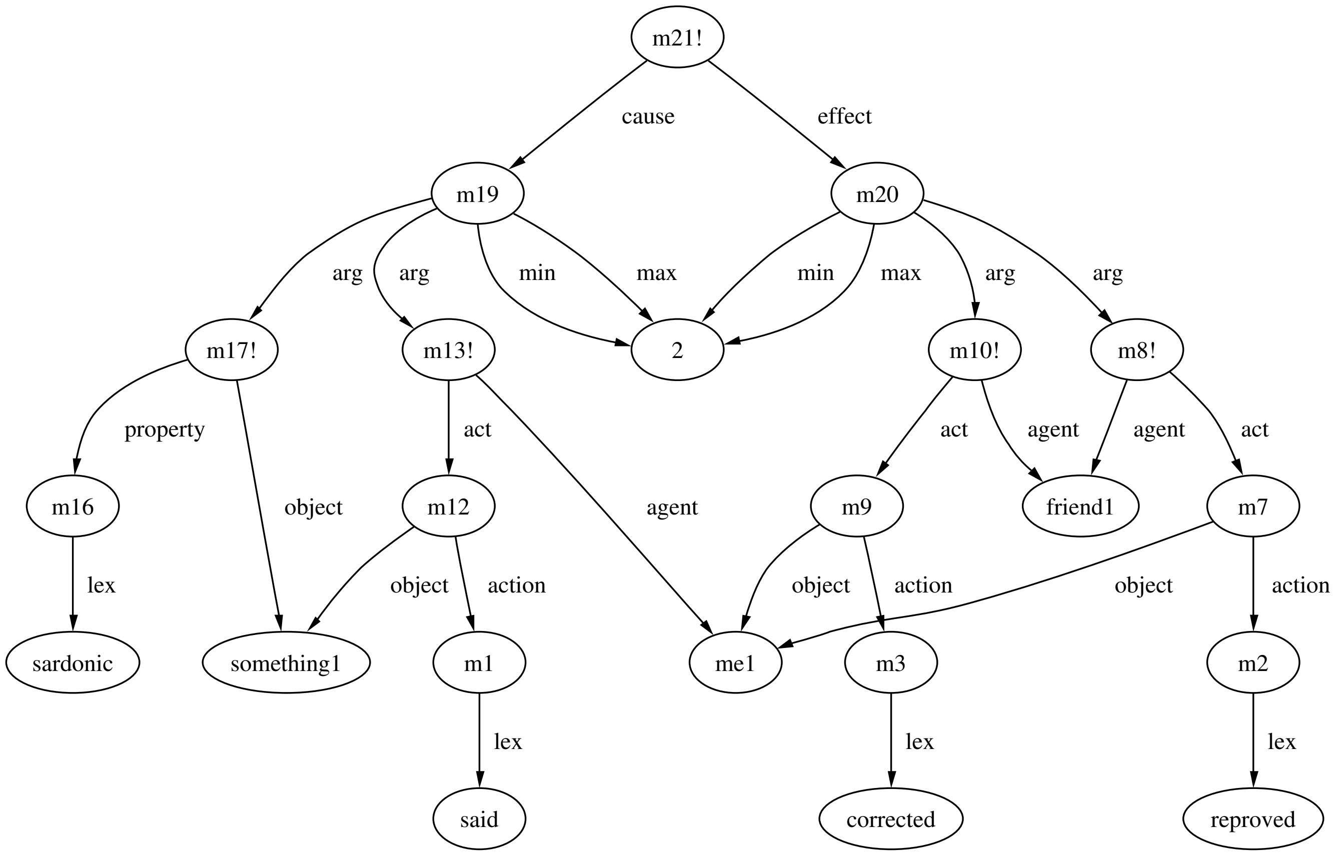

Figure 2: Case frame syntax for *effect/cause*

```
effect/cause = [[m21!]] is the proposition that the effect [[m20]]
                  was caused by [[m19]].
```

The syntax for the *mod/object* case frame is given in Figure 1. This is an example from one of the representations of the target sentence. `M5` (which has a *lex* arc going out of it and going into the lexical item *should*) represents the modifier of `m13!`. The syntax for *effect/cause* is given in Figure 2. `M19` represents the cause and `m20` represents the effect (again, an example from the target sentence for this project).

4.2 Background Knowledge Representation

In order for CASSIE to be able to derive a meaning for an unknown word in a passage, she must be given some background knowledge that will assist her in inferring a dictionary-like definition of the unknown word. The background knowledge for this project includes two rules of inference that do not contain any mention of the unknown word from the passage or any reference to its real-world meaning.

The first of these two rules is (see Figure 3 in Appendix C for a SNePS network diagram of this rule):

```
;=========================================================================
; For all p, x, y, and z, if some person x says some y, some person
; p reproves and corrects x, y has some property z, and z is
; unknown, then it is the case that p reproved and corrected x because
; x said y and y has the property z.
;=========================================================================


all(p,x,y,z) ({(Performs (x, act (concept-called(said), y))),
               (Performs (p, act (concept-called(reproved), x))),
               (Performs (p, act (concept-called(corrected), x))),
               (Prop (y, z)),
               (Prop (z, unknown))}
              &=> Because ((Performs (p, act (concept-called(reproved),x))
                            and
                            Performs (p, act(concept-called(corrected), x))),
                            (Performs (x, act (concept-called(said), y))
                             and
                             Prop (y, z)))).
```

It was noted above that the phrase "You could have gone all day without saying that" was removed from the representation because it is a reproof and a correction itself. When each of the subjects read this phrase they linked it to the fact that it was a reproof and a correction. Therefore, the above rule states that if someone says something and someone else reproves and corrects that person and if what that person said has some as yet unknown property, then the latter person reproved and corrected the former person because he or she said something that has the unknown property. This leads in quick succession to the second rule in the knowledge base (see Figure 4 in Appendix C for a SNePS network diagram of this rule).

```
;========================================================================
; For all p, x, y, and z, if some person p reproves and corrects some
; person x because x says some y and y has property z, then for all a
; and b if some person a says some b and b has the same property z as before,
; then it is not the case that a should have said b.
;========================================================================


all(p,x,y,z) (Because ((Performs (p, act (concept-called(reproved),x))
                              and
                              Performs (p, act(concept-called(corrected), x))),
                              (Performs (x, act (concept-called(said), y))
                               and
                               Prop (y, z)))
               => all(a,b) ({(Performs (a, act (concept-called(said), b))),
                               (Prop (b, z))}
                               &=> ~Nec (concept-called(should),
                               Performs (a, act (concept-called(said), b))))).
```

This rule was a direct result of the responses made by the human subjects in the verbal protocols. Each subject noted that when they read the omitted phrase "You could have . . ." followed by the words *reproof* and *correction* in the original sentence they knew that *sardonic* must have a negative connotation and is, therefore, something you should not say. This rule says that if someone reproves and corrects someone else because that person said something that has an unknown property, then whenever anyone says something that has that property it is not the case that they should have said it (i.e., they should not have said it). Therefore, this rule captures the connotation of *sardonic* being something that people should not say but, nevertheless, do.

4.3 Representing the Passage

Given the second background rule above, the desired definition of *sardonic* for this project is the following:

> If someone says something that has the property of being sardonic then they should not have said it.

For this project, since the design includes all and only the information from the original sentence that is necessary for CASSIE to discover this one possible definition of *sardonic*, it follows that we should reduce the original sentences into a single sentence containing only this information. The condensed version of the original sentences is reiterated from above:

> A friend reproved and corrected me when I said something sardonic

The following are the SNePSLOG representations for the condensed version of the sentence (see Figures 5–9 in Appendix C for the SNePS network diagrams):

```
;=======================================================================
; friend1 (some friend of me1) reproved me1 (me).
;=======================================================================
Performs (friend1, act (concept-called(reproved), me1))!

;=======================================================================
; friend1 (the same friend as before) corrected me.
;=======================================================================
Performs (friend1, act (concept-called(corrected), me1))!
```

Even though an indefinite article is used in the English version of the condensed sentence (which signifies that the friend could be any one of a group of friends), the speaker presumably knows of which friend he is speaking. From the representations above, SNePS creates a node for `friend1` that represents a specific friend. It is important to note that the node created for this term is not a node with a lex arc coming into it that represents the lexical item. It is a node labeled `friend1` instead of, for example, `m1` (for whatever value is appropriate for `m`) and is the SNePS network representation of the friend in the sentence. Likewise, `me1` is a node created by SNePS to represent the speaker in the sentence.

```
;===========================================================================
; me1 (I) said something1.
;===========================================================================
Performs (me1, act (concept-called(said), something1))!
```

Just like `friend1` and `me1`, `Something1` is created as a node that represents the *something* in the sentence. This representation says that the speaker in the above sentence said something.

```
;===========================================================================
; something1 has the property of being sardonic.
;===========================================================================
Prop (something1, concept-called(sardonic))!
```

Crucially, this representation asserts that the `something1` said by `me1` has the property of being the lexical item `sardonic`, `sardonic` being the unknown word.

```
;===========================================================================
; The concept called sardonic is unknown.
;===========================================================================
Prop (concept-called(sardonic), unknown)!
```

This representation, which establishes *sardonic* as being the unknown word in the sentence, triggers the rules which cause CASSIE, through forward inference, to infer that if someone says something that is *sardonic* then that person should not have said it.

5 Results

After providing CASSIE with the background knowledge and SNePS representations of the input sentence, she is able to produce the following output (only the relevant `wffs` have been displayed here; see Appendix A for a complete, annotated transcript):

```
wff25!:

~Nec(concept-called(should),Performs(me1,act(concept-called(said),something1)))

wff22!:

all(b,a)

({Prop(b,concept-called(sardonic)),Performs(a,act(concept-called(said),b))}

&=>

{~Nec(concept-called(should),Performs(a,act(concept-called(said),b)))})
```

It is important to note that each background rule ends with a period (.). This causes CASSIE to assert each rule as nodes in the network. Each part of the representation of the sentence ends with an exclamation point (!). This causes CASSIE to make inferences through forward inferencing. Without the exclamation point at the end of each part of the representation of the sentence forward inferencing would not be triggered and CASSIE would not make any inferences about the meaning of *sardonic.* In addition, full forward inferencing has been turned on in the demo file to ensure that CASSIE makes inferences based on the representation of the sentence and according to the background rules.

Based on the second background rule above, CASSIE infers `wff22!` which is the desired definition. CASSIE then goes a step further by inferring that `me1` should not have said `something1` (presumably because `something1` has the property of being *sardonic*).

6 Future Direction

Since this representation included only the essential information to help CASSIE infer a definition for *sardonic*, there is much that could be done in the future (or even with one more week to work on the project). First, it would be desirable to include a representation for time in this project. In the original sentence, it is important to understand the time intervals that occur. For example, one has to know that when a sentence contains two verbs, the first in the simple past tense and the second in the past perfect tense, the event time for the second verb (the one in the past perfect tense) precedes the event time for the first verb (the one in the simple past tense). This implicit knowledge would cause CASSIE to infer that the speaker first said something *sardonic* and then the friend reproved him. As it stands, the only way for CASSIE to infer the correct definition is by using the background rules with no reference to the time intervals.

Second, to strictly maintain the integrity of the original sentence, it would be desirable to include the phrase "You could have gone all day without saying that". Handling this phrase was not a trivial task since it required creating an odd case frame (e.g., of the form (nil beg next next next next end) where each element of the case frame is a lexical item with a lex arc pointing to it). Since the representation of this sentence was cumbersome and since it is clear from the context that it is a reproof/correction, it was determined that it should be excluded from the final representation of the sentence. But with a little more time, an acceptable representation could be found.

Third, CVA algorithms for adjectives and adverbs will need to be created in the near future so that the contextual definitions of adjectives and adverbs can be packaged in a more human readable manner. Such an algorithm was not available for this project.

In the longer-term scheme of the project, the work that has been done on individual words will need to be analyzed and synthesized so that it can be used to create the algorithmic heuristics (spoken of earlier) that can be taught to humans. With such heuristics as tools for good reading, more people will enjoy delving into difficult literature.

7 Acknowledgments

I would like to thank Dr. Rapaport for his guidance in the development of this project. Without his feedback this project would not have been as enjoyable as it was. I would also like to thank Michael Kandefer for his help with a number of questions regarding SNePS.

A Demo File for the Project ("sardonic3.demo")

```
; =======================================================================
; FILENAME:     sardonic2.demo
; DATE:         15 December 2006
; PROGRAMMER:   Shane Axtell

;; 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 "sardonic2.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":



; 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:

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

; (put annotated SNePSLOG code of your defined frames here;

;  be sure to include both syntax and semantics)

; (also:  be sure to define frames for any paths that you

;  will need below!)


define-frame concept-called (nil lex)

define-frame Performs (nil agent act)

define-frame act (nil action object)

define-frame Prop (nil object property)

define-frame Nec (nil mod object)

define-frame Because (nil effect cause)


; BACKGROUND KNOWLEDGE:

; =====================
; (put annotated SNePSLOG code of your background knowledge here)


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

; For all p, x, y, and z, if some person x says some y and some person

; p reproves x and p corrects x and y has some property z and z is

; unknown, then it is the case that p reproved and corrected x because

; x said y and y has the property z.

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


all(p,x,y,z) ({(Performs (x, act (concept-called(said), y))),

               (Performs (p, act (concept-called(reproved), x))),

               (Performs (p, act (concept-called(corrected), x))),

               (Prop (y, z)),

               (Prop (z, unknown))}

              &=> Because ((Performs (p, act (concept-called(reproved),x))

                            and

                            Performs (p, act(concept-called(corrected), x))),

                            (Performs (x, act (concept-called(said), y))

                             and

                             Prop (y, z)))).


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

; For all p, x, y, and z, if some person p reproves and corrects some

; person x because x says some y and y has property z, then for all a

; and b if some a says some b and b has the same property z as before,

; then it is not the case that a should have said b.

;===========================================================================
all(p,x,y,z) (Because ((Performs (p, act (concept-called(reproved),x))

                              and

                              Performs (p, act(concept-called(corrected), x))),

                              (Performs (x, act (concept-called(said), y))

                               and

                               Prop (y, z)))

               => all(a,b) ({(Performs (a, act (concept-called(said), b))),

                               (Prop (b, z))}

                               &=> ~Nec (concept-called(should),

                                Performs (a, act (concept-called(said), b))))).



; CASSIE READS THE PASSAGE:

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

; (put annotated SNePSLOG code of the passage here)


;A friend reproved and corrected me when I said something sardonic.


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

; friend1 (some friend of me1) reproved me1 (me).

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


Performs (friend1, act (concept-called(reproved), me1))!


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

; friend1 (the same friend as before) corrected me.
```

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

Performs (friend1, act (concept-called(corrected), me1))!

;==========================================================================
; me1 (I) said something1.
;==========================================================================

Performs (me1, act (concept-called(said), something1))!

;==========================================================================
; something1 has the property of being sardonic.
;==========================================================================

Prop (something1, concept-called(sardonic))!

;==========================================================================
; The concept called sardonic is unknown.
;==========================================================================

Prop (concept-called(sardonic), unknown)!
```

B Running Demo of the Project

```
: demo "sardonic3.demo"


File /home/linfaculty/axtell/CSE663/sardonic3.demo is now the source of input.



 CPU time : 0.04


: ; =====================================================================

; FILENAME:     sardonic2.demo

; DATE:         15 December 2006

; PROGRAMMER:   Shane Axtell


;; 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 "sardonic2.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


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 : 0.00


:

; Turn off inference tracing; this is optional.

; If tracing is desired, enter "trace" instead of "untrace":



; Clear the SNePS network:

clearkb

Knowledge Base Cleared



 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


:

; define frames here:

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

; (put annotated SNePSLOG code of your defined frames here;

;  be sure to include both syntax and semantics)

; (also:  be sure to define frames for any paths that you

;  will need below!)


define-frame concept-called (nil lex)

concept-called(x1) will be represented by {<lex, x1>}



 CPU time : 0.00


: define-frame Performs (nil agent act)

Performs(x1, x2) will be represented by {<agent, x1>, <act, x2>}
```

```
 CPU time : 0.00

: define-frame act (nil action object)

act(x1, x2) will be represented by {<action, x1>, <object, x2>}



 CPU time : 0.00

: define-frame Prop (nil object property)

Prop(x1, x2) will be represented by {<object, x1>, <property, x2>}



 CPU time : 0.00

: define-frame Nec (nil mod object)

Nec(x1, x2) will be represented by {<mod, x1>, <object, x2>}



 CPU time : 0.00

: define-frame Because (nil effect cause)

Because(x1, x2) will be represented by {<effect, x1>, <cause, x2>}



 CPU time : 0.00

:
```

```
; BACKGROUND KNOWLEDGE:

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

; (put annotated SNePSLOG code of your background knowledge here)


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

; For all p, x, y, and z, if some person x says some y and some person

; p reproves x and p corrects x and y has some property z and z is

; unknown, then it is the case that p reproved and corrected x because

; x said y and y has the property z.

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


all(p,x,y,z) ({(Performs (x, act (concept-called(said), y))),

               (Performs (p, act (concept-called(reproved), x))),

               (Performs (p, act (concept-called(corrected), x))),

               (Prop (y, z)),

               (Prop (z, unknown))}

              &=> Because ((Performs (p, act (concept-called(reproved),x))

                             and

                             Performs (p, act(concept-called(corrected), x))),

                             (Performs (x, act (concept-called(said), y))

                              and

                              Prop (y, z)))).


  wff4!:  all(z,y,x,p)({Prop(z,unknown),Prop(y,z),

                        Performs(p,act(concept-called(corrected),x)),

                        Performs(p,act(concept-called(reproved),x)),

                        Performs(x,act(concept-called(said),y))}
```

```
                    &=>{Because(Performs(p,act(concept-called(corrected),x))

                        and

                      Performs(p,act(concept-called(reproved),x)),

                      Prop(y,z) and Performs(x,act(concept-called(said),y)))})


 CPU time : 0.00


:

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

; For all p, x, y, and z, if some person p reproves and corrects some

; person x because x says some y and y has property z, then for all a

; and b if some a says some b and b has the same property z as before,

; then it is not the case that a should have said b.

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


all(p,x,y,z) (Because ((Performs (p, act (concept-called(reproved),x))

                             and

                             Performs (p, act(concept-called(corrected), x))),

                             (Performs (x, act (concept-called(said), y))

                              and

                              Prop (y, z)))

               => all(a,b) ({(Performs (a, act (concept-called(said), b))),

                              (Prop (b, z))}

                              &=> ~Nec (concept-called(should),

                               Performs (a, act (concept-called(said), b))))).


  wff6!: all(z,y,x,p)(Because(Performs(p,act(concept-called(corrected),x))
                                    and

                                    Performs(p,act(concept-called(reproved),x)),

                                    Prop(y,z)

                                    and

                                    Performs(x,act(concept-called(said),y)))

                               => (all(b,a)({Prop(b,z),

                                             Performs(a,act(concept-called(said),b))}

                               &=> {~Nec(concept-called(should),

                                     Performs(a,act(concept-called(said),b)))})))


 CPU time : 0.01


:



; CASSIE READS THE PASSAGE:

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

; (put annotated SNePSLOG code of the passage here)


;A friend reproved and corrected me when I said something sardonic.


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

; friend1 (some friend of me1) reproved me1 (me).

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


Performs (friend1, act (concept-called(reproved), me1))!


  wff8!:  Performs(friend1,act(concept-called(reproved),me1))
 CPU time : 0.02


:

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

; friend1 (the same friend as before) corrected me.

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


Performs (friend1, act (concept-called(corrected), me1))!


  wff10!:  Performs(friend1,act(concept-called(corrected),me1))


 CPU time : 0.01


:

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

; me1 (I) said something1.

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


Performs (me1, act (concept-called(said), something1))!


  wff13!:  Performs(me1,act(concept-called(said),something1))

  wff10!:  Performs(friend1,act(concept-called(corrected),me1))

  wff8!:  Performs(friend1,act(concept-called(reproved),me1))


 CPU time : 0.03
```

```
:
;=========================================================================
; something1 has the property of being sardonic.
;=========================================================================


Prop (something1, concept-called(sardonic))!


  wff17!:  Prop(something1,concept-called(sardonic))


 CPU time : 0.00


:
;=========================================================================
; The concept called sardonic is unknown.
;=========================================================================


Prop (concept-called(sardonic), unknown)!


  wff25!:
~Nec(concept-called(should),Performs(me1,act(concept-called(said),something1)))


  wff22!:
all(b,a)({Prop(b,concept-called(sardonic)),
          Performs(a,act(concept-called(said),b))}
    &=> {~Nec(concept-called(should),Performs(a,act(concept-called(said),b)))})


  wff21!:
```

```
Because(Performs(friend1,act(concept-called(corrected),me1))

        and

        Performs(friend1,act(concept-called(reproved),me1)),

        Prop(something1,concept-called(sardonic))

        and

        Performs(me1,act(concept-called(said),something1)))

  wff18!:  Prop(concept-called(sardonic),unknown)

  wff17!:  Prop(something1,concept-called(sardonic))


 CPU time : 0.04


:


CPU time : 0.00


:


End of /home/linfaculty/axtell/CSE663/sardonic3.demo demonstration.



 CPU time : 0.27
```

C SNePS Network Diagrams

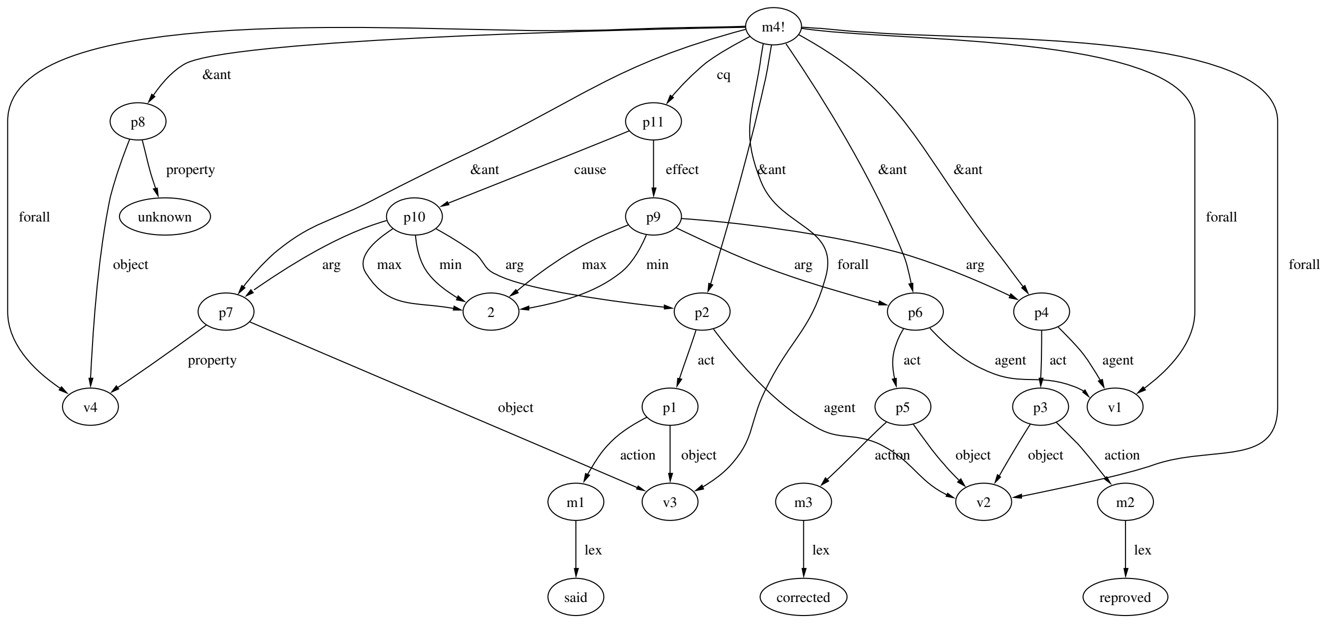

Figure 3: Rule 1 of the Background Knowledge

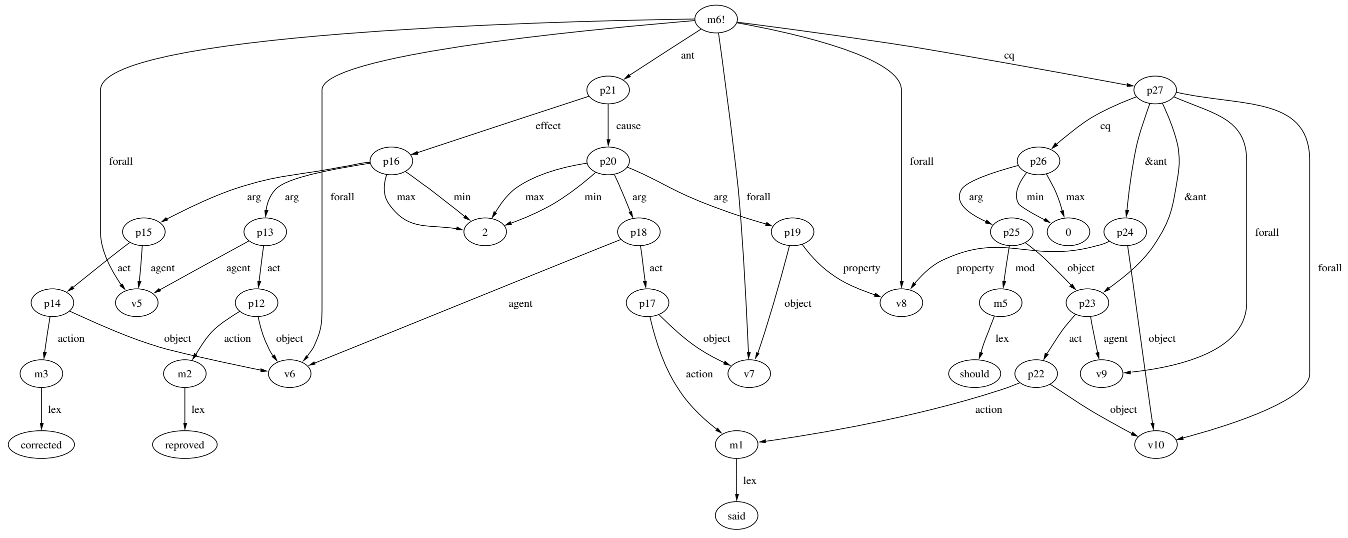

Figure 4: Rule 2 of the Background Knowledge

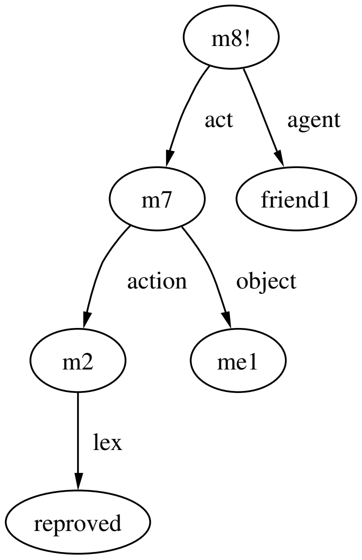

Figure 5: “A friend reproved me”

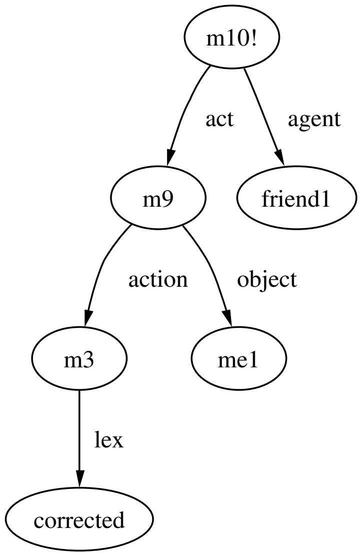

Figure 6: “A friend corrected me”

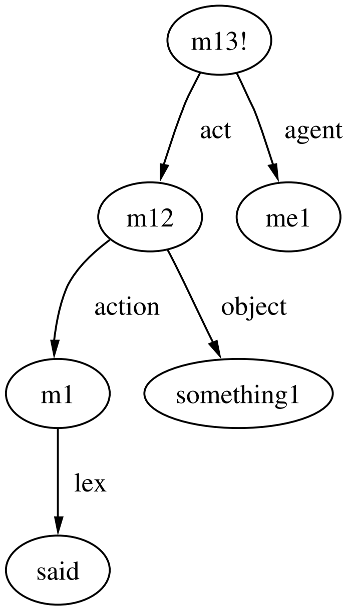

Figure 7: “I (`me1`) said something (`something1`)”

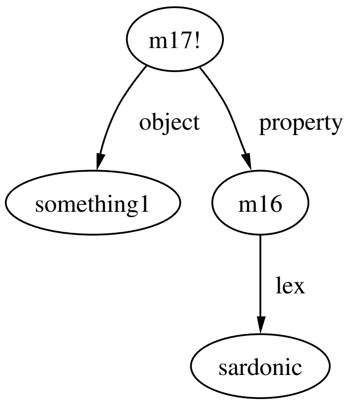

Figure 8: "`Something1` has the property of being *sardonic*"

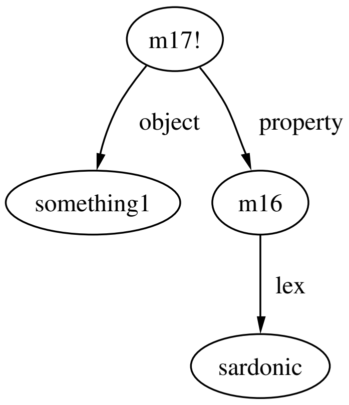

Figure 9: "*Sardonic* is the unknown word"

References

Maxwell, Neal A. (2004), “Remember How Merciful the Lord Hath Been”, *Ensign*, General Conference Report, April 2004.

Rapaport, William J., & Kibby, Michael W. (2002), “ROLE: Contextual Vocabulary Acquisition: From Algorithm to Curriculum”,

Rapaport, William J. (2006), “How Helen Keller Used Syntactic Semantics to Escape from a Chinese Room”, in press.

Shapiro, Stuart C. (2004), “SNePS 2.6.1 User’s Manual”.