CVA: Defining Ravage Using the Semantic Network of SNEPS

Areea Mostofi

May 2011

1 Abstract

Contextual Vocabulary Acquisition uses the semantic graph system of SNePS, which is known as CASSIE, to represent the human mind when interpreting a literary context with an unknown word. The programmer must be give CASSIE the background knowledge necessary to define the unknown word and give the context in which the word is being used. Once this is all inserted, there is an algorithm that parses this information and defines the unknown word. For my project, I represented the word 'ravage' and the background knowledge necessary for CASSIE to understand the word. I ran the Becker's verb algorithm to get the most optimal and acceptable results. Within my context, I had a parenthetical phrase that gives the definition of 'ravage.' I did not have the time to represent the phrase in CASSIE's mind. Hopefully in the future, I will be able to complete the representation including the parenthetical phrase. If completed, it would be the first time a grammatical representation will be used as inference in CASSIE's mind.

2 CVA, CASSIE and SNePS

When people are reading a passage and they come across a word they don't know there is a number of possibilities on how they treat this word. They will either be lazy and just skip over it, look it up online or in a dictionary, or attempt to define the word using the context. They use their background knowledge, the context, and maybe even their knowledge of etymology to guess what the word might mean. The Contextual Vocabulary Acquisition project uses computation to represent this process using a semantic network and algorithms to solve this problem. This information is represented using the SNePSUL language and inserted into a cognitive agent known as CASSIE.

CASSIE is a cognitive agent that processes the information given to her in a semantic network. A semantic network is a directed graph that uses edges as relations to connect together nodes objects, actions, etc. She uses relations to understand what is going on in her mind and the information given to her. Using these relations she builds a knowledge network which she can parse through to answer questions that are asked of her by the programmer. Using this parsing process she receives information, from the programmer, to add to her knowledge network. To do all of this she takes in information using the SNePS programming language. There are two versions of this, SNePSUL and SNePSLOG. This language is logic based in both versions, and both output very similar information; however the difference is how you input the information. Therefore, either could be used successfully, but for my project I used SNePSUL.

There are many ways to go about the CVA project to come to an output. The main process includes four main steps. The first and probably the hardest part is picking a context with an unknown word. The second step is showing the context to the subjects and interviewing them about the unknown word. The third step is representing the thought process needed to understand the context, in a semantic network. The final step is taking that semantic representation from the third step and translating it into SNePSUL. A context with an unknown word has to be easy enough to be understandable from the context, but not too easy that a synonym can be found. The context must also have enough background knowledge so that it can help the subjects understand the context. This will help them answer questions about their thought process. Some of these questions include what background knowledge they used at coming to the conclusion of the meaning, and what in the context helped them come to a conclusion. Background knowledge can include not only meanings of words, but also grammatical knowledge included in the English writing. After getting the information from the verbal protocol, the best information is taken and translated into a graphical representation. This representation needs to have enough background knowledge to help CASSIE give a definition. From the graphical representation it is an easy translation into SNEPSUL code for the programmer. Then the algorithm is used to parse through CASSIE's mind give a definition of the unknown word.

3 My Context

I picked a context that would have an interesting way for representation and give interesting verbal protocal. For my context I selected:

"The invading armies proceeded to RAVAGE-completely ruin and destroy-the local churches, schools and public buildings."

The unknown word is 'ravage.' The meaning of the 'ravage' by the Oxford English Dictionary is "to devastate, lay waste (land, a country, etc.), as by deliberate destruction or plunder." Because this is a very common word, I decided to replace the word with a Farsi translation 'verani.' The reason I replaced the word is because the word 'ravage' is common among the vocabulary of a college student. This would make the interviews useless because these subjects would not use the context or background knowledge to infer the meaning.

In my context there is also a parenthetical phrase, "completely ruin and destroy," which is preceded by the word 'ravage.' The point of a parenthetical phrase is to define the preceding word for the reader. Many times it is used when the word is out of the scope of the readers' vocabulary knowledge.

4 Verbal Protocols (Interviews)

The use of the parenthetical phrase and use of 'verani', made my verbal protocols give an interesting perspective of how to interpret the context. I felt that using the parenthetical phrase for all of my verbal protocols would not provide enough information that I could use for my representation. In addition, an experiment that I found interesting was about whether or not people know the purpose of a parenthetical phrase. Therefore, I decided to perform four interviews in total: two with the parenthetical phrase and two without the parenthetical phrase. As I stated before, I decided to use 'verani' in the place of 'ravage' because 'ravage' can be a common place word. Out of the two interviews with the parenthetical phrase both subjects knew what the parenthetical phrase was for; both tried their best not to use the information in the phrase to define the word. The subjects that didn't have the parenthetical phrase had difficulty defining 'verani' as being associated to destruction. Instead they concentrated on invasion which is suggested the context. I decided that only two of the four protocols, one from each group, provided usable information for my research.

First, I will begin with the interviewee that was given the parenthetical phrase. (For purposes of privacy I will just call her AL). AL understood what the parenthetical phrase was and when asked the meaning of 'verani', she explicitly stated, "Destroy is already used to define the word." In her own words, she defined 'verani' as "armies conquer, capture, and take over." Then when I asked her how she inferred this information she cited the context and her background knowledge. She stated that from her background knowledge she knows that invading armies "burn down and demolish the areas they are invading." So from the context, she concluded that the armies were

invading armies and used her background knowledge to infer what their actions might be. Then when asked why do they 'verani' the local churches, schools and public buildings, she replied, "They do this in order to destroy order and cripple government." So the general background knowledge I received from this verbal protocol is that invading armies burn down and demolish the areas they are invading in order to destroy organization.

The second interviewee was Mr. O, and I did not provide for him the context with the parenthetical phrase. When I asked him for a definition of the word 'verani,' he interpreted the word as "to take over." Then when I asked him to elaborate, he gave two perspectives on what invading armies do when they "take over." First, he stated that invading armies could perform harmful actions to public buildings. Second he mention, for example, the United States was an invading army that helped the people of Italy, during World War II; they helped preserve public buildings. Later in the interview, he explained that the invading armies did not have to be armies in the strictest sense of soldiers and troops, but could be also understood as bed bugs. Around this time, the bed bug epidemic was occurring in New York City and surrounding areas, and he therefore felt that the "bed bugs are invading cities." Although I did not use his information directly in my representation it gave an interesting point of view on how people can process contextual information.

5 Representation

There were many ways to represent my background knowledge and context in such a way to get CASSIE to understand the passage. Background knowledge does not only include general knowledge statements but also has to contain logic rules to represent what a person's mind does naturally. Originally, I was going to use Mr. O's idea that armies can preserve in some instances. Instead I decided that the majority of the time armies destroy, so there is no need to represent the preservation as an act of armies. Therefore, I have represented that armies only destroy.

5.1 Background Knowledge

First, I needed some background knowledge that will help CASSIE run the algorithm. I needed to represent that invading armies destroy churches, schools, and public buildings. I did not represent churches, schools and public buildings seperately, as I did for my verbal protocol, but decided that all three are public buildings. This will make the representation easier for insertion into CASSIE's mind.

"Invading armies destroy public buildings."

Using this statement, I created a general rule that would activate CASSIE's inference engine whenever she sees the word 'ravage.' This rule would take in all variables because it should be able to be used in different situations. This rule, however, is generally just catered to this special case.

For all x, y, act1, act2. If x does act1 to y and x does act2 to y and act2 is unknown then if x does act2 to y then x does act1 to y.

The general rule activated to say for all invading armies(x), public buildings(y), destroy(act1), and 'ravage'(act2). If invading armies destroy public buildings, and invading armies 'ravage' public buildings, and 'ravage' is unknown, then if invading armies 'ravage' public buildings then invading armies destroy public buildings.

5.2 Context

I represented the context exactly as above using public buildings jointly instead of separately as churches, schools and public buildings. I must also represent 'ravage' as being unknown.

Invading armies ravage public buildings.

Ravage is the unknown word.

6 Code Representation

The code is represented using SNEPSUL sub-language of SNePS. SNePS uses Case Frames as a simple way to write and understand the code. These case frames are used to represent the given world in relationships. Before I explain the use of case frames, there are special tokens that are used in order to use these case frames. First 'describe' is used to give an output of what CASSIE understood of the representation. Then the use of 'assert' and 'build' are very similar in the sense that they build nodes. 'Assert' is used to create main nodes, while 'build' creates the secondary nodes based off of that assertion.

I used five of the Case Frames of SNePS to represent my background knowledge. I will describe these case frames in English then show their use in my project. First, I used the member class case frame which states that x is a member of the class of y. An example of this is that Ford(x) is the member of the class of car companies(y). Another case frame that I used is the agent act action object. This is useful in describing a subject, who is doing some action to an object. For example the agent John does the action of reading the object "Of Mice and Men." The next case frame is the object property case frame. This describes a property that an object has. One example of this is the object car has the property of being yellow.

Within all of these case frames there is a building of a lex arc. This is created to represent the lexical name of the nodes that are created. The most important Case Frame that I used was the antecedent consequence Case Frame. This represented a universally quantified if-then statement in logic. Within the if-then statement, the building of other case frames occur. First the variables are quantified, then the antecedent followed by case frames, and finally the consequence followed

by case frames. In order for this case frame to work all of the embedded antecedent case frames must be able to processed before it moves to the consequence to create a rule.

6.1 Background Knowledge

The first idea I represented was the notion that armies destroy public buildings. In this representation, I used '#' and '*' because both are related. The use of the '#' is to "declare" a variable while the use of '*' is to utilize that variable. Therefore, I started by declaring and assigning member class to both armies and public buildings.

```
(describe (assert member #armies class (build lex "Invading Armies")))
```

```
(describe (assert member #buildings class (build lex "public buildings")))
```

From there I used these created variables to say that armies do the act of destroying public buildings.

```
(describe (assert agent *armies act (build action (build lex "destroy") object *buildings)))
```

From the explicit background knowledge and the context, this next rule fires using CASSIE's inference engine. This rule's purpose is described above. In this rule, the variables are first quantified using '$' then utilized by '*' like above. The &ant is used to show that there are multiple arguments to the antecedent. The cq represents the consequence, and within the cq there is another ant and cq rule.

```
(describe (assert forall ($x $y $act1 $act2)
        &ant ((build agent *x act (build action *act1 object *y))
              (build agent *x act (build action *act2 object *y))
              (build object *act2 property (build lex "uknown")))
        cq (build ant (build agent *x act (build action *act2 object *y))
              cq(build agent *x act (build action *act1 object *y)))))
```

6.2 Context

Once the background knowledge is given, the context must be given. The context code is very similar to the background knowledge, but one major difference is 'add' is being used in place of 'assert.' This is used to represent the new information that is being added to what CASSIE already knows in her mind. She then takes this information and makes inferences from it.

First I represent the context "the invading armies ravage public buildings." The variables that were declared before are also being used here because they are already in CASSIE's mind and do not need to be created again.

```
(describe (add agent *armies
          act (build action (build lex "ravage")
          object *buildings)))
```

Then I have to add that 'ravage' is the unknown word so CASSIE can run the algorithm on this word.

```
(describe (add object (build lex "ravage") property (build lex "unknown")))
```

7 Results

After all of the code is inputted into CASSIE's mind, I then ran the verb algorithm on the information. Instead of using the regular verb algorithm, I decided to use Becker's verb algorithm. The main reason why I used Becker's algorithm was because it provided a better output for my word. It was more clear and concise in defining the word and gave better descriptors with the output. The output that was given as:

```
verb:
      lex: ravage;
      property: unknown;
      similar action: (destroy);
      effect: (destroy);
      transitivity: (transitive);
```

The algorithm outputted exactly what was needed to show that CASSIE found a relation between 'ravage' and 'destroy' whenever given 'ravage' as an unknown. She found that not only is 'destroy' a similar action to 'ravage,' but also that the effect of ravaging is to destroy something.

8 Future Work

The way I represented this context and attempted to define 'ravage' is just one of many ways to complete this task. For my future work, there is one other way that I would like to represent my context and background knowledge in CASSIE's mind. Once again, my context is "The

invading armies proceed to ravage-completely ruin and destroy-the local churches, schools and public buildings." My way of representing this context to CASSIE, is by giving her the exact sentence. This would be a productive method to represent a context because a truly general rule could then be created for CASSIE, saying that anything in a parenthetical phrase is the definition of the preceding word.

In order to do this, the sentence must be represented in CASSIE's mind. There is a case frame in SNePS that can be used to represent this sentence. (For a more complete description see the SNePS manual.) From there a rule must be created to parse this sentence, looking for the dashes in the sentence. Once it finds a opening dash it takes all of the information until the closing dash and incorporates this as the definition or paraphrase of the preceding word. This example would not use the verb algorithm, such as my representation, but needs another algorithm or special questions to ask CASSIE to infer the meaning of 'ravage' from the parenthetical phrase.

CASSIE is a perfect example of the idea of a thinking computer. She can take information given to her and parse through it and answer questions. She can find contradictions and change her belief system when learning new idea. The future of CASSIE is a thinking machine that no longer needs programmed input but can parse through natural language data and think on its own. This is just the beginning of the future of Artificial Intelligence.

9 Appendix

A Demo File

```
; ============================================================
; FILENAME: AREEAMOSTOFI-RAVAGE-DEMO.txt
; DATE: 4/26/2011
; PROGRAMMER: Areea Mostofi
; Using the Becker Verb Algorithm
; Lines beginning with a semi-colon are comments.
; Lines beginning with "^" are Lisp commands.
; All other lines are SNePSUL commands.
;
; To use this file: run SNePs; at the SNePS prompt (*), type:
;
; (demo "AREEAMOSTOFI-RAVAGE-DEMO" :av)
;
; Make sure all necessary files are in the current working directory
; or else use full path names.
;================================================

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

;Loading of the (default) appropriate definition algorithm:
;Not use because of ambiguity of output.
;(load "/projects/rapaport/CVA/STN2/defun_verb.cl")

;Loading of Becker's Verb Algorithm
^(load "/projects/rapaport/CVA/verbalgorithm3.1/defun_verb.cl")

;Clear the SNePS network:
(resetnet t)

; Optional:
; UNCOMMENT THE FOLLOWING CODE TO TURN FORWARD INFERENCING ON:
;
; ;enter the "snip" package:
; ^(in-package snip)
;
; ;turn on full forward inferencing:
; ^(defun broadcast-one-report (represent)
; (let (anysent)
;   (do.chset (ch *OUTGOING-CHANNEL* anysent)
;     (when (isopen.ch ch)
;     (setq anysent
;   (of (try-to-send-report represent ch)
```

```
; anysent)))))
; nil)

; ;re-enter the "sneps package:
^(in-package sneps)

; load all pre-defined relations:
; NB: If "intext" causes a "nil not of expected type" error,
; then comment-out the "intext" command and then
; uncomment & use the load command below instead
;^(load "/projects/rapaport/CVA/STN2/demos/rels")
;(intext "/projects/rapaport/CVA/STN2/demos/rels")


;loading of Becker's Relations
^(load "/projects/rapaport/CVA/verbalgorithm3.1/rels")

; load all pre-defined path definitions:
;(intext "/projects/rapaport/CVA/mkb3.CVA/paths/paths")
;^(load "/projects/rapaport/CVA/mkb3.CVA/paths/paths")
;loading of Becker's
^(load "/projects/rapaport/CVA/verbalgorithm3.1/paths")

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

;Invading armies destroy public buildings

(describe (assert member #armies class (build lex "Invading Armies")))
(describe (assert member #buildings class (build lex "public buildings")))
(describe (assert agent *armies act (build action (build lex "destroy") object (*buildings))))


;RULE:
;For all x, y, act1, act2 IF x does act1 to y and x does act 2 to y and act2 is unknown
;THEN IF x does act2 to y THEN x does act1 to y.

(describe (assert forall ($x $y $act1 $act2)
&ant((build agent *x act (build action *act1 object *y))
(build agent *x act (build action *act2 object *y))
(build object *act2 property (build lex "unknown")))
cq (build ant (build agent *x act (build action *act2 object *y))
cq(build agent *x act (build action *act1 object *y)))))


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

;Invading armies ravage public buildings.
```

```
(describe (add agent *armies
act (build action (build lex "ravage")
object *buildings)))

;Ravage is the unknown word.
(describe (add object (build lex "ravage") property (build lex "unknown")))


;Ask Cassie what "RAVAGE" means:
^(defineVerb 'ravage)
```

B Trial Run

```
Script started on Wed 11 May 2011 07:32:01 PM EDT
nickelback {~/Desktop} > mlisp

International Allegro CL Enterprise Edition
8.2 [Linux (x86)] (Jul 9, 2010 16:05)
Copyright (C) 1985-2010, 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/sni   wu   iz/bin
Error: "/projects/snwiz/bin" is not a regular data file
  [condition type: file-error]

Restart actions (select using :continue):
 0: retry the load of /projects/snwiz/bin
 1: skip loading /projects/snwiz/bin
 2: Return to Top Level (an "abort" restart).
 3: Abort entirely from this (lisp) process.
[1] cl-user(2): :ld /sn      projects/sni   wiz/bin/sneps
;   Loading /projects/snwiz/bin/sneps.lisp
;;; Installing streamc patch, version 2.
Loading system SNePS...10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
SNePS-2.7 [PL:2 2011/04/19 17:07:58] loaded.
Type `(sneps)' or `(snepslog)' to get started.
[1] cl-user(3): (sneps)


   Welcome to SNePS-2.7 [PL:2 2011/04/19 17:07:58]
```

```
Copyright (C) 1984--2010 by Research Foundation of
State University of New York. SNePS comes with ABSOLUTELY NO WARRANTY!
Type ‘(copyright)’ for detailed copyright information.
Type ‘(demo)’ for a list of example applications.

   5/11/2011 19:32:28

* (demo "AREEAMOSTOFI-RAVAGE-DEMO.txt")

File /home/csdue/areeamos/Desktop/AREEAMOSTOFI-RAVAGE-DEMO.txt is now the source of input.


 CPU time : 0.00

*
; ============================================================
; FILENAME: AREEAMOSTOFI-RAVAGE-DEMO.txt
; DATE: 4/26/2011
; PROGRAMMER: Areea Mostofi
; Using the Becker Verb Algorithm
; Lines beginning with a semi-colon are comments.
; Lines beginning with "^" are Lisp commands.
; All other lines are SNePSUL commands.
;
; To use this file: run SNePs; at the SNePS prompt (*), type:
;
; (demo "AREEAMOSTOFI-RAVAGE-DEMO" :av)
;
; Make sure all necessary files are in the current working directory
; or else use full path names.
;================================================

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

;Loading of the (default) appropriate definition algorithm:
;Not use because of ambiguity of output.
;(load "/projects/rapaport/CVA/STN2/defun_verb.cl")

;Loading of Becker's Verb Algorithm
^(
--> load "/projects/rapaport/CVA/verbalgorithm3.1/defun_verb.cl")
;   Loading /projects/rapaport/CVA/verbalgorithm3.1/defun_verb.cl
;     Loading /projects/rapaport/CVA/verbalgorithm3.1/DataCollection.cl
;     Loading /projects/rapaport/CVA/verbalgorithm3.1/DataProcessing.cl
t
```

```
 CPU time : 0.01

*
;Clear the SNePS network:
(resetnet t)

Net reset


 CPU time : 0.00

*
; Optional:
; UNCOMMENT THE FOLLOWING CODE TO TURN FORWARD INFERENCING ON:
;
; ;enter the "snip" package:
; ^(in-package snip)
;
; ;turn on full forward inferencing:
; ^(defun broadcast-one-report (represent)
; (let (anysent)
;   (do.chset (ch *OUTGOING-CHANNEL* anysent)
;     (when (isopen.ch ch)
;     (setq anysent
;   (of (try-to-send-report represent ch)
; anysent)))))
; nil)

; ;re-enter the "sneps package:
^(
--> in-package sneps)
#<The sneps package>



 CPU time : 0.00

*
; load all pre-defined relations:
; NB: If "intext" causes a "nil not of expected type" error,
; then comment-out the "intext" command and then
; uncomment & use the load command below instead
;^(load "/projects/rapaport/CVA/STN2/demos/rels")
;(intext "/projects/rapaport/CVA/STN2/demos/rels")


;loading of Becker's Relations
```

```
^(
--> load "/projects/rapaport/CVA/verbalgorithm3.1/rels")
;   Loading /projects/rapaport/CVA/verbalgorithm3.1/rels
 act is already defined.
 action is already defined.
 object1 is already defined.
 object2 is already defined.
 effect is already defined.
t



 CPU time : 0.01

*
; load all pre-defined path definitions:
;(intext "/projects/rapaport/CVA/mkb3.CVA/paths/paths")
;^(load "/projects/rapaport/CVA/mkb3.CVA/paths/paths")
;loading of Becker's
^(
--> load "/projects/rapaport/CVA/verbalgorithm3.1/paths")
;   Loading /projects/rapaport/CVA/verbalgorithm3.1/paths
before implied by the path (compose before
                            (kstar (compose after- ! before)))
before- implied by the path (compose (kstar (compose before- ! after))
                             before-)
after implied by the path (compose after
                           (kstar (compose before- ! after)))
after- implied by the path (compose (kstar (compose after- ! before))
                            after-)
sub1 implied by the path (compose object1- superclass- ! subclass
                          superclass- ! subclass)
sub1- implied by the path (compose subclass- ! superclass subclass- !
                           superclass object1)
super1 implied by the path (compose superclass subclass- ! superclass
                            object1- ! object2)
super1- implied by the path (compose object2- ! object1 superclass- !
                             subclass superclass-)
superclass implied by the path (or superclass super1)
superclass- implied by the path (or superclass- super1-)
t



 CPU time : 0.00

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

```
;Invading armies destroy public buildings

(describe (assert member #armies class (build lex "Invading Armies")))

(m2! (class (m1 (lex Invading Armies))) (member b1))

(m2!)

 CPU time : 0.00

* (describe (assert member #buildings class (build lex "public buildings")))

(m4! (class (m3 (lex public buildings))) (member b2))

(m4!)

 CPU time : 0.00

* (describe (assert agent *armies act (build action (build lex "destroy") object (*buildings))))

(m7! (act (m6 (action (m5 (lex destroy))) (object b2))) (agent b1))

(m7!)

 CPU time : 0.00

*

;RULE:
;For all x, y, act1, act2 IF x does act1 to y and x does act 2 to y and act2 is unknown
;THEN IF x does act2 to y THEN x does act1 to y.

(describe (assert forall ($x $y $act1 $act2)
&ant((build agent *x act (build action *act1 object *y))
(build agent *x act (build action *act2 object *y))
(build object *act2 property (build lex "unknown")))
cq (build ant (build agent *x act (build action *act2 object *y))
cq(build agent *x act (build action *act1 object *y)))))

(m9! (forall v4 v3 v2 v1)
 (&ant (p5 (object v4) (property (m8 (lex unknown))))
  (p4 (act (p3 (action v4) (object v2))) (agent v1))
  (p2 (act (p1 (action v3) (object v2))) (agent v1)))
 (cq (p6 (ant (p4)) (cq (p2)))))

(m9!)

 CPU time : 0.00
```

```
*

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

;Invading armies ravage public buildings.
(describe (add agent *armies
act (build action (build lex "ravage")
object *buildings)))

(m12! (act (m11 (action (m10 (lex ravage))) (object b2))) (agent b1))
(m7! (act (m6 (action (m5 (lex destroy))) (object b2))) (agent b1))

(m12! m7!)

 CPU time : 0.00

*
;Ravage is the unknown word.
(describe (add object (build lex "ravage") property (build lex "unknown")))

(m15!
 (ant
  (m12! (act (m11 (action (m10 (lex ravage))) (object b2)))
   (agent b1)))
 (cq
  (m7! (act (m6 (action (m5 (lex destroy))) (object b2))) (agent b1))))
(m13! (object (m10)) (property (m8 (lex unknown))))

(m15! m13! m7!)

 CPU time : 0.00

*

;Ask Cassie what "RAVAGE" means:
^(
--> defineVerb 'ravage)

 verb:
       lex:  ravage;
       property:  unknown;
       similar action: (destroy)
       effect: (destroy)
       transitivity: (transitive)
nil
```

```
 CPU time : 0.01

*

End of /home/csdue/areeamos/Desktop/AREEAMOSTOFI-RAVAGE-DEMO.txt demonstration.
```

C Semantic Network