11 August 2003

Contextual Vocabulary Acquisition
Inferring the meaning of the verb ‘cripple’ from context

CSE 700: Independent Study
Summer Session II 2003

Adel Ahmed

Abstract

A verb definition algorithm has been developed as part of the work on the Contextual Vocabulary Acquisition (CVA) project. This paper presents a SNePS representation of a passage containing the verb ‘cripple’, along with the relevant background knowledge. The aim is to experiment with the verb algorithm as to whether or not it will be able to produce a definition for the verb ‘cripple’ using the developed representations. The results show that Cassie was able to infer the meaning of ‘cripple’ but the verb algorithm needs some changes to return the definition.

Introduction

The Contextual Vocabulary Acquisition (CVA) project is a research project concerned with developing strategies, algorithms and scientific methods for inferring the meaning of an unknown word from context and background knowledge, which may include prior encounters with the unknown word. The goal is to use the developed techniques and algorithms to help students improve their reading comprehension. Definition algorithms have been developed to infer the meaning of unknown nouns, verbs and adjectives. Refer to the CVA web page[1] for more information about this project.

This paper presents a representation, in SNePS[2], of a passage containing the word 'cripple' along with the necessary background knowledge. The passage is:
"Typhoon Vera killed or injured 218 people and crippled the seaport city of Keelung."[3]
The objective is to experiment with the verb algorithm and test whether or not it will be able to infer the meaning of the verb 'cripple' from context using the developed representations.

Cassie, the cognitive agent of SNePS, was successful in inferring the meaning, but the verb algorithm needs some modification to return the desired definition.

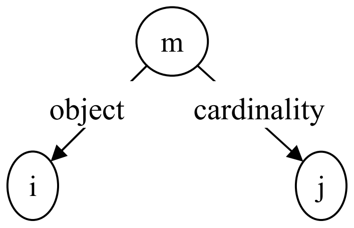
Implementation

The first step is to identify the relevant background knowledge that a "typical" human would have when reading the above passage. Then, representations of the passage and the background knowledge are to be developed and loaded in SNePS, along with the verb algorithm. Finally, run the verb algorithm, analyze the results, and make any necessary changes to the representations and/or the verb algorithm.

Case Frames

In addition the standard case frames found on the CVA and SNePS web pages, the following additional 'object/cardinality' case frame is needed:

Syntax



Semantics:
[[m]] is the proposition that the cardinality of [[i]] is [[j]].

Example:
Typhoon Vera killed 218 people.

```
(build object #vera proper-name Vera)
(build member *vera class (build lex "Typhoon"))
(build object #some-people cardinality 218)
(build subclass *some-people superclass (build lex "people"))
(build agent *vera  act (build object *some-people  action (build lex "kill")))
```

Background Knowledge

In order to know what background knowledge a "typical" human would have when reading the above passage, the concept of "think-aloud" protocols was used. The word 'crippled' was replaced by the made up word 'glivened' as follows:

"Typhoon Vera killed or injured 218 people and glivened the seaport city of Keelung."

Then, I asked a few people, individually, to think aloud as to what they think the meaning of the word ‘glivened’ was and why. They all agreed that ‘glivened’ is a negative and harmful thing. Since the action of killing or injuring is ‘bad’, then ‘glivened’ is most likely to be ‘bad’ as well. This is in addition to the fact that a typhoon usually has negative consequences. They came up with definitions like devastated, destroyed, damaged, and submerged. Based on this, the following rule was developed to be part of the background knowledge:

Rule 1: *If X does A and A has the property P, and X does B and B is unknown, then, possibly, B also has the property P.*

Since “killed or injured” involves killing, injuring or both, the following rule was also developed:

Rule 2: *If X kills or injures Y, then X kills Y or X injures Y or both.*

The background knowledge should include definitions of the words “typhoon”, “city”, and “seaport”, and the fact that killing or injuring is ‘bad’. For simplicity, the above words were defined as follows:

A typhoon is a violent windstorm.
A city is a large municipal area where lots of people live.
A seaport is a port on seashore that has facilities for seagoing ships.

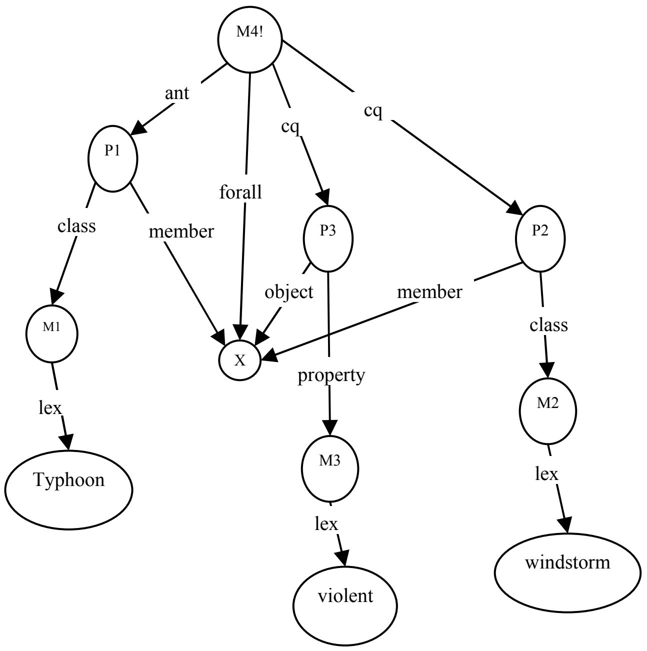
The fact that killing or injuring is ‘bad’ was expressed as follows:

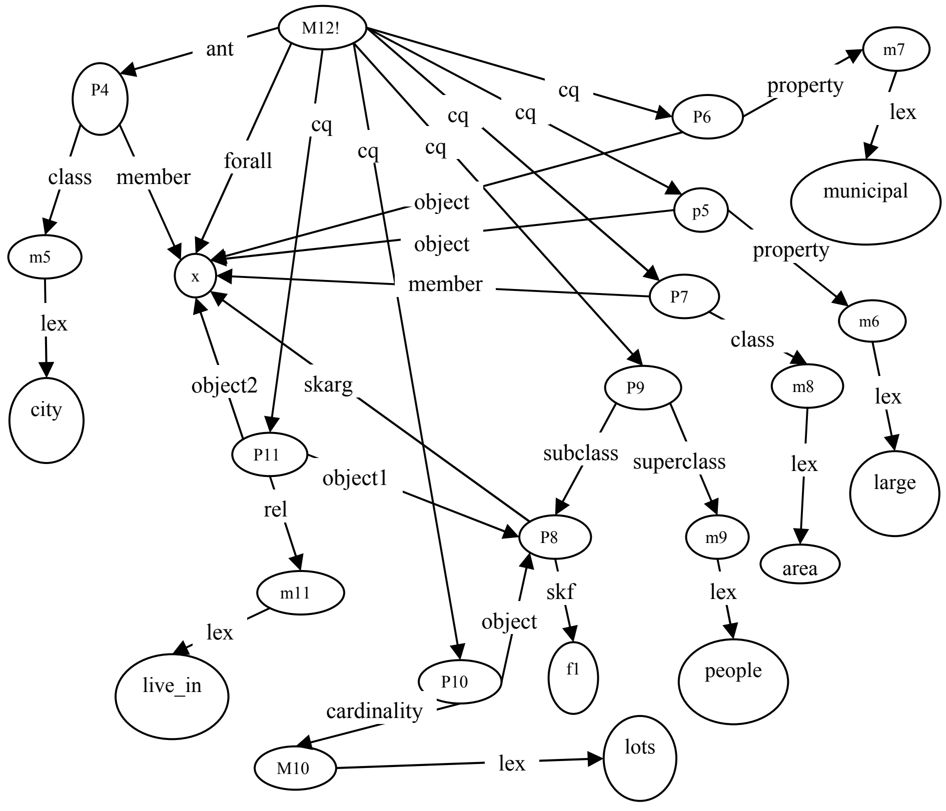
Killing has the property of being ‘bad’
Injuring has the property of being ‘bad’.

SNePS Representation of Background Knowledge

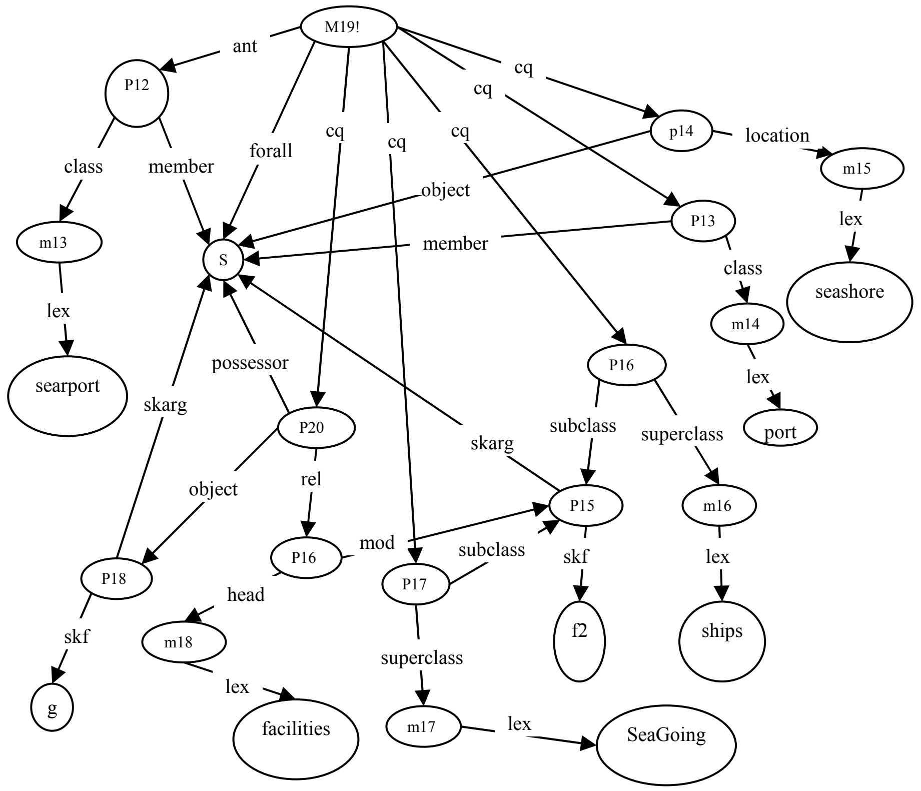
After careful analysis and several attempts, the background knowledge was represented as follows:

A typhoon is a violent windstorm.





A city is a large 'municipal' area where lots of people live.

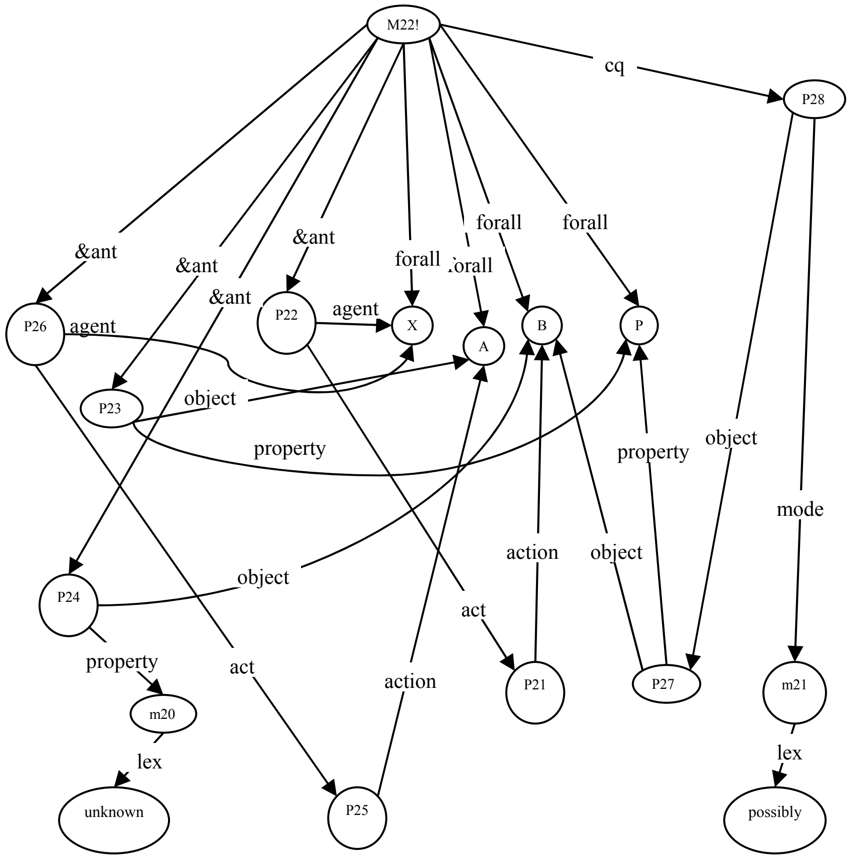


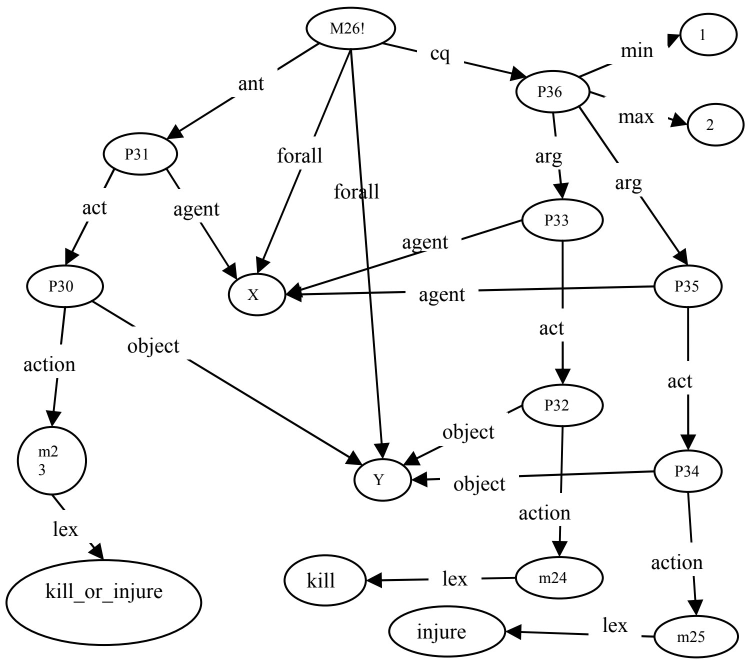
A seaport is a port on seashore that has facilities for seagoing ships.

If X does A and A has the property P, and X does B and B is unknown,

then, possibly,

B also has the property P.

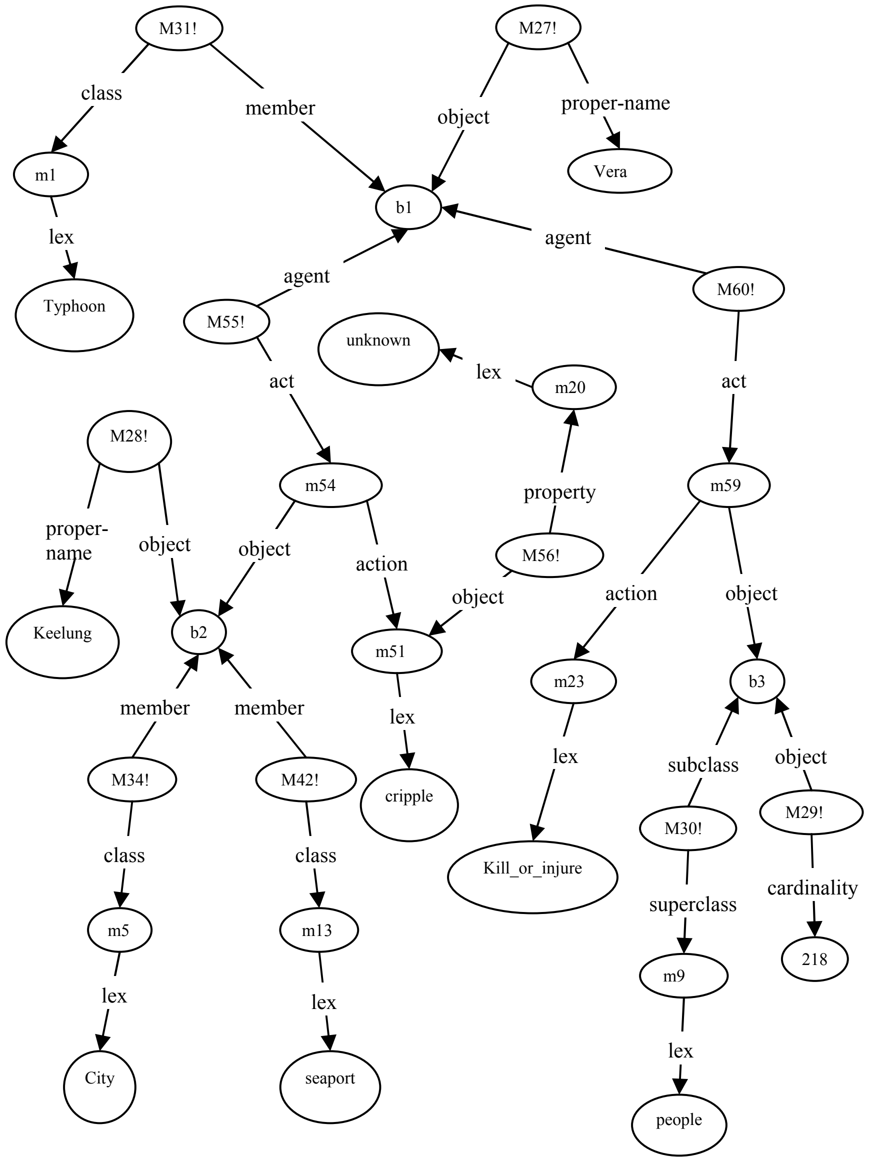




If 'X' kills_or_injures 'Y', then 'X' kills 'Y' or 'X' injures 'Y' or 'both'.

SNePS Representation of Passage:

Typhoon Vera killed or injured 218 people and crippled the seaport city of Keelung.



Results

Appendix I shows the SNePS code starting with the background knowledge and followed by the passage representations   Appendix II shows a sample run.

The following shows the results of running the verb algorithm in an attempt to define the word 'cripple':

```
^(defineVerb 'cripple);

"You want me to define the verb 'cripple'.

I'll start by looking at the predicate stucture of the sentences I know that use 'cripple'.  Here is what I know:

The most common type of sentences I know of that use 'cripple' are of the form:
    'A something can cripple.'

    'A something can cripple something.'

No superclasses were found for this verb.

 Sorting from the most common predicate case to the least common here is what I know.  I will first attempt
to unify the components of the sentences that use the verb giving a generalizaiton based on my background
knowledge:

A typhoon can cripple.
A windstorm can cripple.

A typhoon can cripple a city, area, seaport, port.
A windstorm can cripple a city, area, seaport, port.

Now, looking from the bottom up I want to get a sense of the categories that most of the agents, objects and
indirect objects belong to.  This is different from looking for the most unified case.  Instead I am looking for
the classes that contain approximately half of the agents, objects and indirect objects.  This is an attempt at
generalization but from another approach.

A typhoon can cripple.
A windstorm can cripple.

A typhoon can cripple a city, area, seaport, port.
A windstorm can cripple a city, area, seaport, port."
```

From looking at the output, we see that although the algorithm was successful in concluding that:

A typhoon can cripple a city, area, seaport, port.
A windstorm can cripple a city, area, seaport, port."

it was unable to produce a definition for 'cripple'. Cassie was successful in applying Rule 1 to infer that "cripple is possibly bad" as can be seen from (M69!) shown below:

```
;; killing or injuring is bad
(describe
(add object (build lex "kill_or_injure")
    property (build lex "bad")));
(m69! (mode (m21 (lex possibly)))
 (object (m68 (object (m51 (lex cripple))) (property (m66 (lex bad))))))
```

However, it seems that the verb algorithm was not able to make use of the object-property relation. Therefore, the verb algorithm may need to be modified for this. Further analysis of this is needed in order to determine how the algorithm is to be modified.

Future Work

If I had more time, I would have liked to experiment with the verb algorithm and make the necessary changes to have it return a definition. I would have started by having the algorithm process object-property propositions. That is, for every proposition that has an "object" arc pointing to the unknown verb and another "property" arc pointing to some property, the property should be returned as part of the definition.

Also, since the noun algorithm is able to process object-property propositions, I would take a look at its code and make similar changes to the verb algorithm.

Also, it would be nice to enhance the verb algorithm to be able to group related things into categories. For example, “city, area, seaport, and port” are all geographical locations and “typhoon and windstorm” are considered “bad weather”.

References

1. William J. Rapaport, “Contextual Vocabulary Acquisition: from algorithm to curriculum”, http://www.cse.buffalo.edu/~rapaport/CVA/cva.html

2. “SNeRG, The SNePS Research Group”, http://www.cse.buffalo.edu/sneps

3. Clarke, D.F., & Nation, I.S.P. (1980), "Guessing the Meanings of Words from Context: Strategy and Techniques", *System* 8: 211-220.

Appendix I: Representations of background knowledge and passage in SNePS code.

```
;;; Reset the network
(resetnet t)

;;; Don't trace infer
^(setq snip:*infertrace* nil)

;;; Load all valid relations
(intext "./rels")

(define member class object proper-name subclass superclass cardinality act agent action lex);
(define possessor location rel property mode head mod skf skarg);

;;; Compose paths
;;(intext "/projects/rapaport/CVA/mkb3.CVA/paths/paths")
(intext "./paths")

;;; Load Ehrlich Algorithm
;;;^(load "/projects/rapaport/CVA/mkb3.CVA/src/defn3")

;;^(load "../src/synfn")

^(load "./defun_verb.cl")


;;BACKGROUND KNOWLEDGE
;;-----------------------------------------

;;A typhoon is a violent windstorm.

(describe
(add forall $x
 ant (build member *x class (build lex "typhoon"))

 cq (build member *x class (build lex "windstorm"))
 cq (build object *x property (build lex "violent"))));




;;A city is a large (municipal) area where lots of people live.

(describe
(add forall $x

     ant (build member *x class (build lex "city"))
     cq (build object *x property (build lex "large"))
     cq (build object *x property (build lex "municipal"))
     cq (build member *x class (build lex "area"))
     cq (build subclass (build skf f1 skarg *x) = lotsOfPeople
             superclass (build lex "people"))

     cq (build object *lotsOfPeople cardinality (build lex "lots"))

     cq (build object1 *lotsOfPeople
             rel (build lex "live_in")
             object2 *x)));








;; A seaport is a port on a seashore that has facilities for seagoing ships
```

```
(describe
(add forall $s
 ant (build member *s class (build lex "seaport"))

     cq (build member *s class (build lex "port"))
     cq (build object *s location (build lex "seaShore"))

     cq (build subclass (build skf f2 skarg *s) = seaGoingShips
             superclass (build lex "ships"))

     cq (build subclass *seaGoingShips
             superclass (build lex "seaGoing"))

     cq (build object (build skf g skarg *s)
             possessor *s
             rel (build mod *seaGoingShips
                     head (build lex "facilities")))));



;; Rule 1
;; IF x does a AND a has the property p;
;; AND x does b AND b is unknown ,
;; THEN 'possibly' b also has the property p.

(describe
(add forall ($x $a $b $p)

      &ant ( (build agent *x    act (build action *a))
             (build object *a   property *p)
             (build object *b   property (build lex "unknown"))
             (build agent *x    act (build action *b)))

      cq   (build mode (build lex "possibly")
              object (build object *b
                        property *p))));




;; Rule 2: IF x kill_or_injure y, THEN x kills y or x injures y or both;
(describe
(add forall ($x $y)
 ant (build agent *x
         act (build action (build lex "kill_or_injure")
                 object *y))

 cq  (build min 1 max 2
     arg (build agent *x
              act (build action (build lex "kill")
                      object *y))

     arg (build agent *x
             act (build action (build lex "injure")
                     object *y)))));


;; Passage: Typhoon Vera killed or injured 218 people and "crippled"
;;         the seaport city of Keelung.
;; ------------------------------------------------------------------------------

(describe
(add object #vera proper-name vera));

(describe
(add object #keelung proper-name keelung));


(describe
(add object #some-people cardinality 218));

(describe
(add subclass *some-people superclass (build lex "people")));


;; Vera is a Typhoon
(describe
(add member *vera class (build lex "typhoon")));

;; Keelung is a city
(describe
(add member *keelung class (build lex "city")));


;; Keelung is a seaport.
(describe
(add member *keelung class (build lex "seaport")));



;; Vera crippled Keelung
(describe
(add agent *vera
        act (build object *keelung
                action (build lex "cripple"))));


(describe
(add object (build lex "cripple") property (build lex "unknown")));



;; Vera killed_or_injured people
(describe
(add agent *vera
        act (build object *some-people
                action (build lex "kill_or_injure"))));


;;
;; killing or injuring is bad
(describe
(add object (build lex "kill_or_injure")
    property (build lex "bad")));


(describe
(add object (build lex "kill")
    property (build lex "bad")));

(describe
(add object (build lex "injure")
    property (build lex "bad")));


;; run the verb algorithm for the verb 'cripple'
^(defineVerb 'cripple);
```

Appendix II: Sample Run

```
Script started on Thu 07 Aug 2003 08:19:40 PM ED
pollux {~/summer} > al□□[Kcl
International Allegro CL Enterprise Edition
6.2 [Solaris] (Aug 15, 2002 14:24)
Copyright (C) 1985-2002, Franz Inc., Berkeley, CA, USA.  All Rights Reserved.

This development copy of Allegro CL is licensed to:
   [4549] SUNY/Buffalo, N. Campus

;; 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/snwiz/bin/sneps
; Loading /projects/snwiz/bin/sneps.lisp
Loading system SNePS...10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
SNePS-2.6 [PL:0a 2002/09/30 22:37:46] loaded.
Type `(sneps)' or `(snepslog)' to get started.
cl-user(2): (sneps)


   Welcome to SNePS-2.6 [PL:0a 2002/09/30 22:37:46]

Copyright (C) 1984--2002 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.

   8/7/2003 20:20:14

* (demo "cripple.demo")

File /home/csgrad/akahmed/summer/cripple.demo is now the source of input.


 CPU time : 0.01

* ;;; Reset the network
(resetnet t)

Net reset


 CPU time : 0.01

*
;;; Don't trace infer
^(
--> setq snip:*infertrace* nil)
nil



 CPU time : 0.00

*
;;; Load all valid relations
(intext "./rels")
File ./rels is now the source of input.


 CPU time : 0.00

* □effect is already defined.
```

```
(A1 A2 A3 A4 ACT ACTION AFTER AGENT ANTONYM ASSOCIATED BEFORE CAUSE
 CLASS DIRECTION EFFECT EQUIV ETIME FROM IN INDOBJ INSTR INTO LEX
 LOCATION KN_CAT MANNER MEMBER MEMBERS MODE OBJECT OBJECTS OBJECT1
 OBJECTS1 OBJECT2 ON ONTO PART PLACE POSSESSOR PROPER-NAME PROPERTY
 PURPOSE REL SKF STIME SUBCLASS SUPERCLASS SYNONYM TIME TO WHOLE before
 after time cause effect)

 CPU time : 0.02

*

End of file ./rels


 CPU time : 0.02

*
(define member class object proper-name subclass superclass cardinality act agent action lex);□act is already defined.
□action is already defined.

(member class object
  proper-name subclass
  superclass cardinality
  act agent
  action lex)

 CPU time : 0.03

*
(define possessor location rel property mode head mod skf skarg);

(possessor location rel property mode head mod skf skarg)

 CPU time : 0.00

*

;;; Compose paths
;;(intext "/projects/rapaport/CVA/mkb3.CVA/paths/paths")
(intext "./paths")
File ./paths is now the source of input.


 CPU time : 0.00

*
before implied by the path (compose before
                         (kstar (compose after- ! before)))
before- implied by the path (compose (kstar (compose before- ! after))
                          before-)


 CPU time : 0.00

*
after implied by the path (compose after
                        (kstar (compose before- ! after)))
after- implied by the path (compose (kstar (compose after- ! before))
                         after-)


 CPU time : 0.00

*
sub1 implied by the path (compose object1- superclass- ! subclass
                       superclass- ! subclass)
sub1- implied by the path (compose subclass- ! superclass subclass- !
                        superclass object1)
 CPU time : 0.00

*
super1 implied by the path (compose superclass subclass- ! superclass
                          object1- ! object2)
super1- implied by the path (compose object2- ! object1 superclass- !
                           subclass superclass-)


 CPU time : 0.00

*
superclass implied by the path (or superclass super1)
superclass- implied by the path (or superclass- super1-)


 CPU time : 0.00

*

End of file ./paths


 CPU time : 0.01

*
;;; Load Ehrlich Algorithm
;;;^(load "/projects/rapaport/CVA/mkb3.CVA/src/defn3")

;;^(load "../src/synfn")

^(
--> load "./defun_verb.cl")
; Loading /home/csgrad/akahmed/summer/defun_verb.cl
t



 CPU time : 0.10


;;BACKGROUND KNOWLEDGE
;;-----------------------------------------


;;A typhoon is a violent windstorm.

(describe
(add forall $x
 ant (build member *x class (build lex "typhoon"))

 cq (build member *x class (build lex "windstorm"))
 cq (build object *x property (build lex "violent"))
));
(m4! (forall v1) (ant (p1 (class (m1 (lex typhoon))) (member v1)))
 (cq (p3 (object v1) (property (m3 (lex violent))))
  (p2 (class (m2 (lex windstorm))) (member v1))))

(m4!)

 CPU time : 0.00

*




;;A city is a large (municipal) area where lots of people live.
```

```
(describe
(add forall $x

     ant (build member *x class (build lex "city"))

     cq (build object *x property (build lex "large"))

     cq (build object *x property (build lex "municipal"))

     cq (build member *x class (build lex "area"))

     cq (build subclass (build skf f1 skarg *x) = lotsOfPeople
            superclass (build lex "people"))

     cq (build object *lotsOfPeople cardinality (build lex "lots"))

     cq (build object1 *lotsOfPeople
            rel (build lex "live_in")
            object2 *x)
));
(m12! (forall v2) (ant (p4 (class (m5 (lex city))) (member v2)))
 (cq
  (p11 (object1 (p8 (skarg v2) (skf f1))) (object2 v2)
   (rel (m11 (lex live_in))))
  (p10 (cardinality (m10 (lex lots))) (object (p8)))
  (p9 (subclass (p8)) (superclass (m9 (lex people))))
  (p7 (class (m8 (lex area))) (member v2))
  (p6 (object v2) (property (m7 (lex municipal))))
  (p5 (object v2) (property (m6 (lex large))))))

(m12!)

 CPU time : 0.01

*



;; A seaport is a port on a seashore that has facilities for seagoing ships

(describe
(add forall $s
 ant (build member *s class (build lex "seaport"))

     cq (build member *s class (build lex "port"))
     cq (build object *s location (build lex "seaShore"))

     cq (build subclass (build skf f2 skarg *s) = seaGoingShips
            superclass (build lex "ships"))

     cq (build subclass *seaGoingShips
            superclass (build lex "seaGoing"))

     cq (build object (build skf g skarg *s)
            possessor *s
            rel (build mod *seaGoingShips
                   head (build lex "facilities")))

));
(m19! (forall v3) (ant (p12 (class (m13 (lex seaport))) (member v3)))
 (cq
  (p20 (object (p18 (skarg v3) (skf g))) (possessor v3)
   (rel
    (p19 (head (m18 (lex facilities)))
     (mod (p15 (skarg v3) (skf f2))))))
  (p17 (subclass (p15)) (superclass (m17 (lex seaGoing))))
  (p16 (subclass (p15)) (superclass (m16 (lex ships))))
  (p14 (location (m15 (lex seaShore))) (object v3))
  (p13 (class (m14 (lex port))) (member v3))))
```

```
(m19!)

 CPU time : 0.00

*
;; Rule 1
;; IF x does a AND a has the property p;
;; AND x does b AND b is unknown ,
;; THEN 'possibly' b also has the property p.

(describe
(add forall ($x $a $b $p)

       &ant ( (build agent *x
                 act (build action *a))

             (build object *a
                   property *p)

             (build object *b
                   property (build lex "unknown"))

             (build agent *x
                   act (build action *b)))

       cq   (build mode (build lex "possibly")
                 object (build object *b
                             property *p))
));
(m22! (forall v7 v6 v5 v4)
 (&ant (p26 (act (p25 (action v6))) (agent v4))
  (p24 (object v6) (property (m20 (lex unknown))))
  (p23 (object v5) (property v7))
  (p22 (act (p21 (action v5))) (agent v4)))
 (cq
  (p28 (mode (m21 (lex possibly)))
   (object (p27 (object v6) (property v7))))))

(m22!)

 CPU time : 0.02

*




;; Rule 2: IF x kill_or_injure y, THEN x kills y or x injures y or both;

(describe
(add forall ($x $y)
 ant (build agent *x
          act (build action (build lex "kill_or_injure")
                   object *y)
     )

 cq  (build min 1 max 2
      arg (build agent *x
                act (build action (build lex "kill")
                         object *y))

      arg (build agent *x
               act (build action (build lex "injure")
                         object *y))
      )
));
(m26! (forall v9 v8)
 (ant
  (p31 (act (p30 (action (m23 (lex kill_or_injure))) (object v9)))
   (agent v8)))
 (cq
  (p36 (min 1) (max 2)
   (arg
    (p35 (act (p34 (action (m25 (lex injure))) (object v9)))
     (agent v8))
    (p33 (act (p32 (action (m24 (lex kill))) (object v9)))
     (agent v8))))))

(m26!)

 CPU time : 0.01

*




;; Passage: Typhoon Vera killed or injured 218 people and "crippled"
;;          the seaport city of Keelung.
;; -------------------------------------------------------------------------------

(describe
(add object #vera proper-name vera));
(m27! (object b1) (proper-name vera))

(m27!)

 CPU time : 0.04

*

(describe
(add object #keelung proper-name keelung));
(m28! (object b2) (proper-name keelung))

(m28!)

 CPU time : 0.01

*

(describe
(add object #some-people cardinality 218));
(m29! (cardinality 218) (object b3))

(m29!)

 CPU time : 0.00

*

(describe
(add subclass *some-people superclass (build lex "people")));
(m30! (subclass b3) (superclass (m9 (lex people))))

(m30!)

 CPU time : 0.01

*




;; Vera is a Typhoon
(describe
(add member *vera class (build lex "typhoon")));
(m33! (class (m2 (lex windstorm))) (member b1))
(m32! (object b1) (property (m3 (lex violent))))
```

```
(m31! (class (m1 (lex typhoon))) (member b1))

(m33! m32! m31!)

 CPU time : 0.01

*

;; Keelung is a city
(describe
(add member *keelung class (build lex "city")));
(m41! (object b2) (property (m6 (lex large))))
(m40! (object b2) (property (m7 (lex municipal))))
(m39! (class (m8 (lex area))) (member b2))
(m38! (subclass (m35 (skarg b2) (skf f1)))
 (superclass (m9 (lex people))))
(m37! (cardinality (m10 (lex lots))) (object (m35)))
(m36! (object1 (m35)) (object2 b2) (rel (m11 (lex live_in))))
(m34! (class (m5 (lex city))) (member b2))

(m41! m40! m39! m38! m37! m36! m34!)

 CPU time : 0.02

*


;; Keelung is a seaport.
(describe
(add member *keelung class (build lex "seaport")));
(m50! (class (m14 (lex port))) (member b2))
(m49! (location (m15 (lex seaShore))) (object b2))
(m48! (subclass (m44 (skarg b2) (skf f2)))
 (superclass (m16 (lex ships))))
(m47! (subclass (m44)) (superclass (m17 (lex seaGoing))))
(m46! (object (m43 (skarg b2) (skf g))) (possessor b2)
 (rel (m45 (head (m18 (lex facilities))) (mod (m44)))))
(m42! (class (m13 (lex seaport))) (member b2))

(m50! m49! m48! m47! m46! m42!)

 CPU time : 0.05

*



;; Vera crippled Keelung
(describe
(add agent *vera
       act (build object *keelung
               action (build lex "cripple"))));
(m55! (act (m54 (action (m51 (lex cripple))))) (agent b1))
(m53! (act (m52 (action (m51)) (object b2))) (agent b1))

(m55! m53!)

 CPU time : 0.00



*



(describe
(add object (build lex "cripple") property (build lex "unknown")));
(m56! (object (m51 (lex cripple))) (property (m20 (lex unknown))))

(m56!)
```

```
 CPU time : 0.01

*



;; Vera killed_or_injured people
(describe
(add agent *vera
       act (build object *some-people
               action (build lex "kill_or_injure"))));
(m65! (min 1) (max 2)
 (arg
  (m64 (act (m63 (action (m24 (lex kill))) (object b3))) (agent b1))
  (m62 (act (m61 (action (m25 (lex injure))) (object b3))) (agent b1))))
(m60! (act (m59 (action (m23 (lex kill_or_injure))))) (agent b1))
(m58! (act (m57 (action (m23)) (object b3))) (agent b1))

(m65! m60! m58!)

 CPU time : 0.01

*



;; killing or injuring is bad
(describe
(add object (build lex "kill_or_injure")
    property (build lex "bad")));
(m69! (mode (m21 (lex possibly)))
 (object (m68 (object (m51 (lex cripple))) (property (m66 (lex bad))))))
(m67! (object (m23 (lex kill_or_injure))) (property (m66)))

(m69! m67!)

 CPU time : 0.01

*



(describe
(add object (build lex "kill")
    property (build lex "bad")));
(m70! (object (m24 (lex kill))) (property (m66 (lex bad))))

(m70!)

 CPU time : 0.00

*

(describe
(add object (build lex "injure")
    property (build lex "bad")));
(m71! (object (m25 (lex injure))) (property (m66 (lex bad))))

(m71!)

 CPU time : 0.01

*




;; run the verb algorithm for the verb 'cripple'
^(
--> defineVerb 'cripple);


Do you want to debug it? no
Do you want to debug it? no
"You want me to define the verb 'cripple'.

I'll start by looking at the predicate stucture of the sentences I know that use 'cripple'.  Here is what I know:

The most common type of sentences I know of that use 'cripple' are of the form:
    'A something can cripple.'

    'A something can cripple something.'



No superclasses were found for this verb.

 Sorting from the most common predicate case to the least common here is what I know.  I will first attempt to unify the components
of the sentences that use the verb giving a generalizaiton based on my background knowledge:

A typhoon can cripple.
A windstorm can cripple.


A typhoon can cripple a city, area, seaport, port.
A windstorm can cripple a city, area, seaport, port.


Now, looking from the bottom up I want to get a sense of the categories that most of the agents, objects and indirect objects belong to.
This is different from looking for the most unified case.  Instead I am looking for the classes that contain approximately half of the
agents, objects and indirect objects.  This is an attempt at generalization but from another approach.

A typhoon can cripple.
A windstorm can cripple.


A typhoon can cripple a city, area, seaport, port.
A windstorm can cripple a city, area, seaport, port.


"



 CPU time : 0.14

*


End of /home/csgrad/akahmed/summer/cripple.demo demonstration.


 CPU time : 0.54

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

script done on Thu 07 Aug 2003 08:20:36 PM ED
```