CSE663

Contextual Vocabulary Acquisition Project

Inferring the Meaning of Quail

Lunarso Sutanto

December 15, 2003

Abstract

This project discusses research into a knowledge-based contextual vocabulary acquisition (CVA), specifically the contribution that the author has made by making Cassie (a computational cognitive agent based on SNePS) infer the meaning of the word 'quail' from the context of a passage containing the unknown word.

The CVA project is a long-term undertaking by William J. Rapaport and Michael W. Kibby to investigate how readers accurately guess the meaning of a previously unknown word from background knowledge and the word's usage in context ([RK02]). The goal is to develop a computational algorithm that will eventually be used in the classroom to help students with understanding their readings. The algorithm in developed in conjunction with the use of SNePS ([SR87]), a knowledge representation and reasoning system based on semantic networks, to represent background knowledge and as the 'brain' doing the inferences to arrive at the meaning of the unknown word.

Using a passage from Tolkien's Lord of the Rings [Tol54] containing the word 'quail', the author ran an informal experiment among friends, asking them to figure out the word and describe how they went about figuring out the meaning of the word. Based on these experiments, background knowledge and rules of inference were developed, which allowed Cassie to figure out the correct connections between the unknown word and pieces of context, although the CVA algorithm did not run correctly.

1

The CVA Project

The Contextual Vocabulary Acquisition Project's purpose is to investigate and formalize the process that people go through when trying to understand the meaning of a new word that they encounter in their readings. The chief end of the formalization is its utilization in education, teaching the process to students to help them understand better the process of comprehension and acquiring new vocabulary, thus aiding then to increase their learning ability in the classroom.

The first step in developing this formalization is to examine how people figure out the meaning of a new word when they encounter it in their readings, apart from looking it up in a dictionary or asking someone else what it means ([RK02]). In other words, they must deduce the meaning from the context of the passage read. This step must run concurrent to developing a computational algorithm for vocabulary acquisition: as new insights are revealed in how people 'make the connection' as they read a passage, we implement these ideas in a knowledge base and improve our algorithm. In fact, this report paper describes this very step: a passage is selected containing a hard word. The passage is shown to several people, asking them to figure out the meaning of the word from the context of the passage. Their thinking process is then crystallized into a formal rule in a knowledge representation system and will eventually be incorporated into the main vocabulary acquisition algorithm.

1.1 Knowledge Representation and Reasoning: SNePS

The knowledge representation and reasoning system that the CVA project uses is SNePS, which is an abbreviation for Semantic NEtwork Processing System [SR87]. SNePS itself is an ongoing project being developed in SUNY at Buffalo's Computer Science and Engineering department, with the goal of developing a computational cognitive agent capable of using natural language. This cognitive agent is called Cassie by the SNePS Research Group (SNeRG). Here the distinction must be stressed between SNePS and Cassie: we represent the passage and related background knowledge in Cassie's mind, using the SNePS format. It is incorrect to think of SNePS and Cassie as interchangable names for the same entity, a common misconception to which the author has been known to fall into in the past.

Cassie may be considered as our subject, a hypothetical person reading a passage containing a word not in her knowledge base (outside of her previous encounters). When asked to define the meaning of the word, she will form connections from the context of the passage (assuming that she knows all other words in the passage) and define the word in terms of how the word relates to other words.

1.2 Selection of the Passage

The most natural way to select a hard word is from one's own readings. A passage from J.R.R. Tolkien's "The Lord of The Rings" was selected, containing the archaic verb *'quail'*. Here is the complete passage:

> The dark figure streaming with fire raced towards them. The orcs yelled and poured over the stone gangways. Then Boromir raised his horn and blew. Loud the challenge rang and bellowed, like the shout of many throats under the cavernous roof. For a moment the orcs *quailed* and the fiery shadow halted. Then the echoes died as suddenly as a flame blown out by a dark wind, and the enemy advanced again.

* For anyone interested in Tolkien lore, this passage is taken verbatim from Book Two of The Fellowship of the Ring, Chapter 5. It describes events occuring just after the fellowship discovered that they are facing a Balrog (a big, bad demon).

It should be noted that in order to simplify representation in SNePS, all words are changed to use the present tense. The reader is urged to try to deduce the meaning of the word quail and note the thinking

process in inferring the meaning from the passage (if the reader has never encountered the word before). Most likely, the reader's guess will be quite close to the actual meaning of the word; and the thinking process will be similar to the background rule used in the project.

The word in this context means "to shrink back in fear, to cower" ([dic03]). It comes from the Middle English word 'quailen', which means to "give way". 'Quail' is not likely to be used as a verb in our time; much more likely, it is used a noun for a type of bird.

1.3 Experimentation on Friends

Once the passage and the hard word is approved for use in this research, the first step taken was to show the passage to friends and ask them what they thought the meaning of the word is. A good percentage of them managed to figure out that 'quail' is somehow related to the word 'halt', even though 'quail' only appears once in the passage.

Inference Rule

The common threads of reasoning that runs amongst the test subjects are these 'common-sense' logic:

1. If a person A and a person B is advancing toward a person C; and if something happens that causes person A to *stop what it is doing* and person B does something unknown, then it is most likely that the unknown action is related to stopping (what person A is doing).

2. If a person A advances *again* after doing something unknown, then the unknown action is probably stopping.

Most test subjects use both of these rules in tandem to reach the conclusion that 'quail' means something like 'halt' or 'stop'. Either of the rules used by itself is sufficient to reach the correct conclusion, though having the two is like having a double punch to confirm one's guess better. There is something quite different about the two common-sense rules above though: Rule 1 relies more on logical reasoning, while Rule 2 relies more on the meaning of the word 'again'. Rule 1 captures the logical reasoning that if two people is performing a similar act and something happens to cause the first person to perform act

X and the other person to perform some act Y, then it is highly likely that act X and act Y is related. Rule 1 as given above is quite specific to the passage (usage of the words 'advance' and 'stop'), but it can be generalized to the form given previously (using general variable X and Y instead of actual English words). Rule 2 does not rely on such a causal rule of inference, instead it is the word 'again', plus an understanding that the unknown word is most likely an antonym of the word 'advance'.

As such, Rule 1 is the only rule selected to be used in this project, since it seems to be the more general of the two. Rule 2 will be quite interesting to implement as well; it will give Cassie an understanding of the word 'again'.

1.4 Passage Representation

For the purpose of this project, the passage is represented in SNePS sentence by sentence. After that the rule of inference is added and Cassie figures out that 'quail' is related to 'halt'. After this we run the general VerbDefinition algorithm and see if the algorithm can make any more general connections [RE00].

A general theme that runs through these representations is *simplification*. Knowledge engineering is not just about putting in everything with as much detail as possible. In this project, it is enough to put in a sufficient amount of detail for the logic to work. If it works with a little detail, then it should work for a larger amount of detail.

Since we are using a causal rule of inference, representations of time and causality are issues that have to be dealt with. Suggestions on how to represent time can be found in [Alm95], and a very simple version of that is used in this project. Causality however, is a huge field in the discipline of Artificial Intelligence; with branches such as situation calculus [MH69] and event calculus [Sha99]. Using these systems in this project is simply overkill, although they might very well prove to be important in the future. One hitch to using situation calculus in this project is that actions occur concurrently in the narrative, thus requiring quite a bit of work in extending situation calculus to handle concurrency. The representation of time and actions in this project however, is much closer to the spirit of event calculus, though very much simplified.

The representation of the narrative assumes three general time periods, not necessarily overlapping:

1. The first time period when both the Balrog and the orcs are chasing after the fellowship.

2. The middle time period when Boromir blows his horn.

3. The final time period when the Balrog halted and the orcs quailed.

We are leaving out the actual final event from the excerpt where the Balrog and the orcs advanced again after some time since this event is not related to the rule that we have developed.

The way the sentences are represented assume that the reader is familiar with the standard SNePS case frames, which can be found in [SNe96]. There is only one addition to the standard case frame, for which the syntax and semantics can be found in the appendix to this report. The SNePS representation is given in detail in the following pages, with diagrams showing how the representation look visually. Be warned that some of the diagrams are huge and can be hard to understand.

Note these points before using the diagram:

- Oval node labels do not carry over from diagram to diagram. These nodes are for propositions.
- Rectangular node labels *do* carry over from diagram to diagram. These nodes are reserved for objects and lexicals.
- Arc labels are always referring to the arcs to the label's *left*
- These diagrams are courtesy of the `dot` program by E. R. Gansner, S. C. North and K. P. Vo

The Fiery Shadow raced towards Boromir

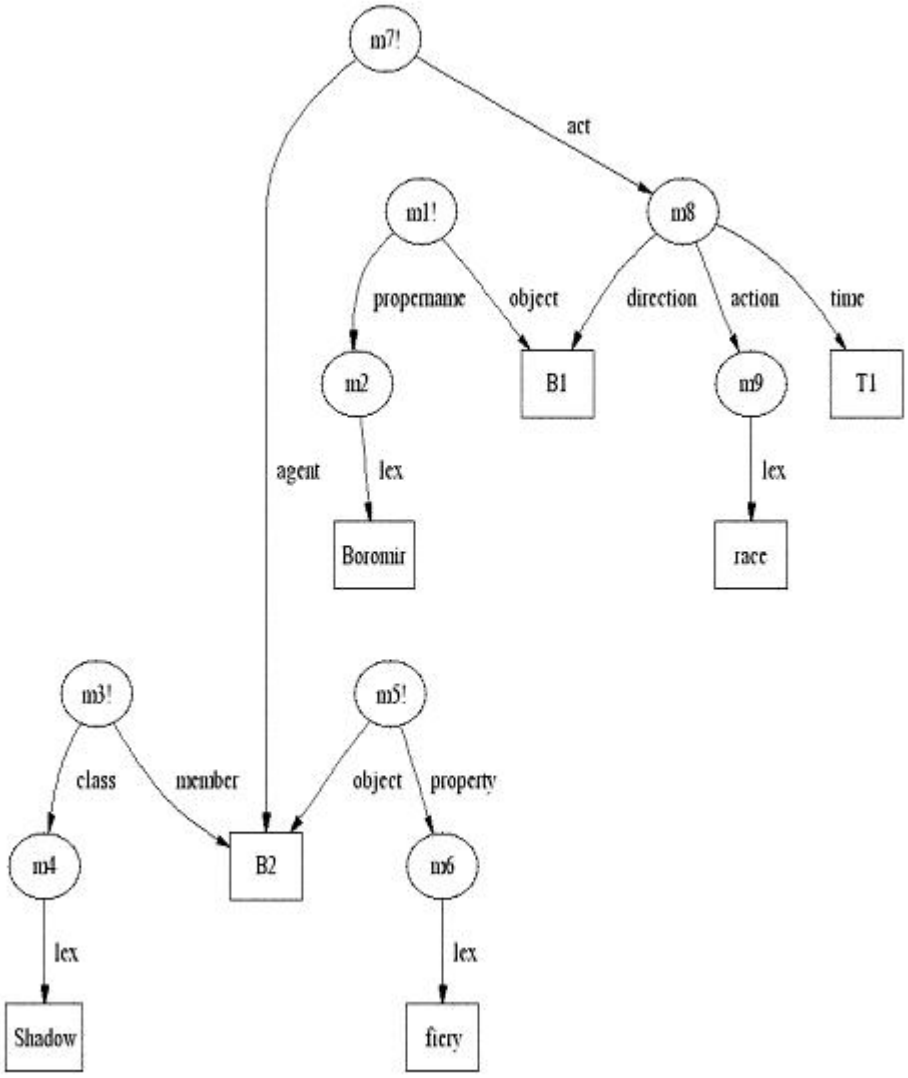
This is a simplification of the original sentence "The Dark Figure streaming with fire raced towards them". 'Them' refers to the Fellowship of the Ring, but because it does not matter to the logic that we are using, 'them' is changed to just 'Boromir', so we are leaving out the fellowship throughout the rest of the passage.

```
(describe (assert propername (build lex "Boromir") object #boromir))
```

```
(describe (assert member #shadow class (build lex "shadow")))
```

```
(describe (assert object *shadow property (build lex "fiery")))
```

```
(describe (assert agent *shadow
                  act (build action (build lex "race") = race
                  direction *boromir)
                  time #time-of-event-1))
```



The orcs yelled and advanced towards Boromir

This is a simplification of "The orcs yelled and poured over the stone gangways".

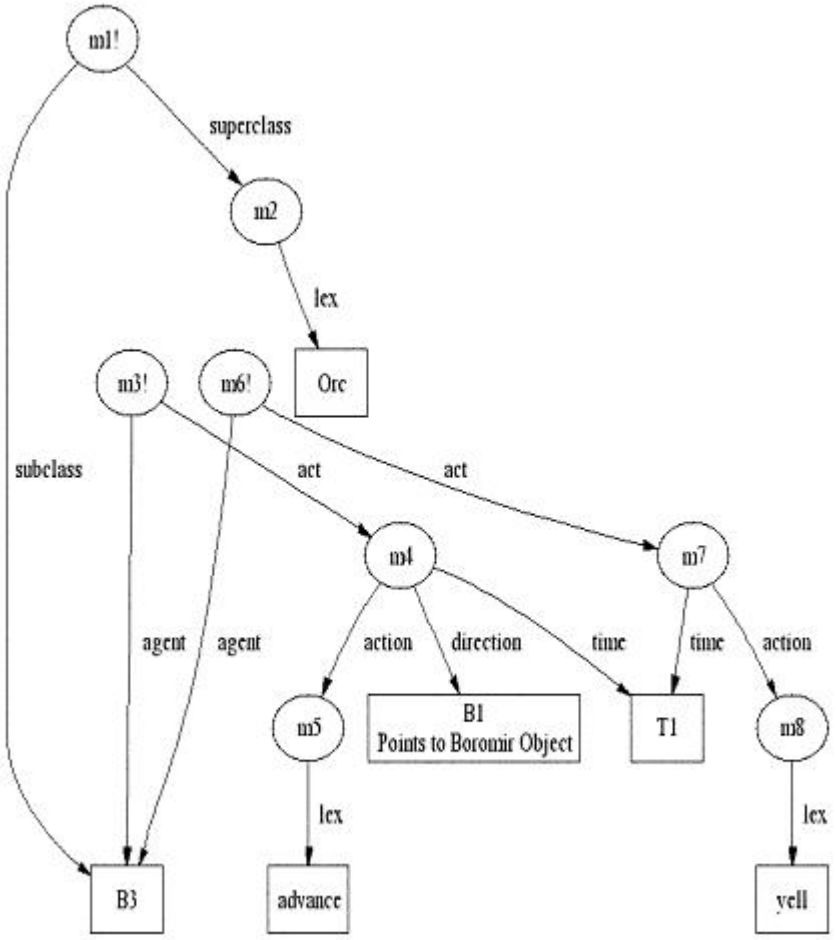
```
(describe (assert subclass #orcs superclass (build lex "orcs")))

(describe (assert agent *orcs
                  act (build action (build lex "yell"))
                  time *time-of-event-1))

(describe (assert agent *orcs
                  act (build action (build lex "advance") = advance
                  direction *boromir)
                  time *time-of-event-1))
```

The best way to represent the orcs is as a subclass of the general class of orcs, particularly the subclass that lives in the Mines of Moria and are chasing after the fellowship. This too, is not stated explicitly since this information is extraneous and not needed by the logic of the verb definition algorithm.

One thing to note here is that this event is occuring at the same time as the event of the Balrog advancing towards Boromir.



Boromir blew his horn, which challenged the orcs

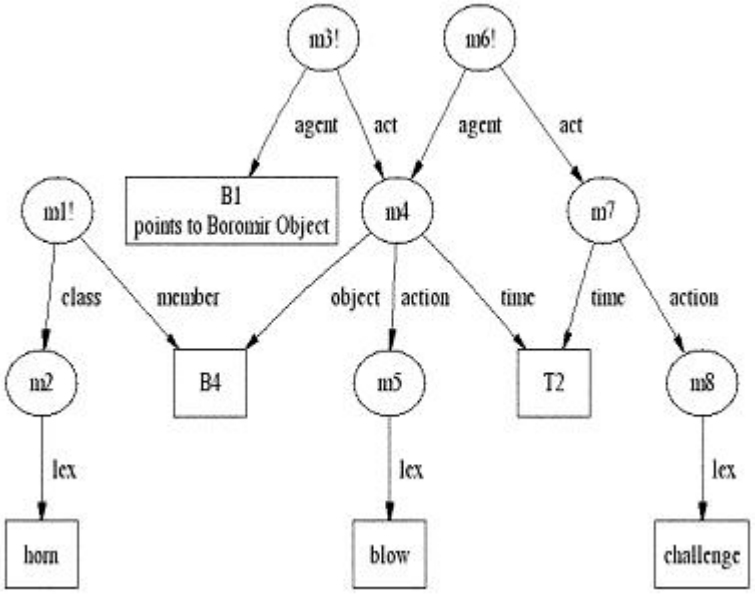
This is the form that I chose to represent of the sentence "Then Boromir raised his horn and blew. Loud the challenge rang and bellowed, like the shout of many throats under the cavernous roof." A point worth mentioning about the representation of this passage is that the action of blowing the horn is what challenged the orcs, not Boromir.

```
(describe (assert member #horn class (build lex "horn")))
```

```
(describe (assert agent *boromir
                  act (build action (build lex "blow")
                  object *horn) = blowaction
```

```
                    time #time-of-event-2))

(describe (assert agent *blowaction
                    act (build action (build lex "challenge")
                    object *orcs)
                    time *time-of-event-2))
```

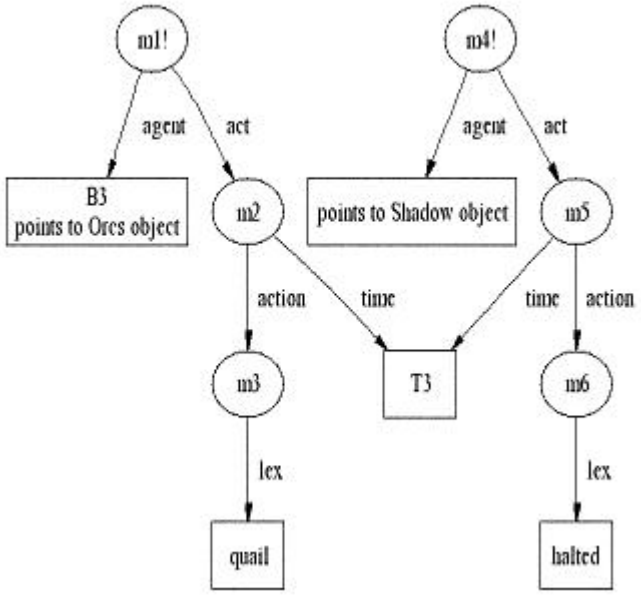


For a moment the orcs quailed and the fiery shadow halted

This sentence was represented without any modification.

```
(describe (assert agent *orcs
                    act (build action (build lex "quail")) = quail
                    time #time-of-event-3))

(describe (assert agent *shadow
                    act (build action (build lex "halt")) = halt
                    time *time-of-event-3))
```

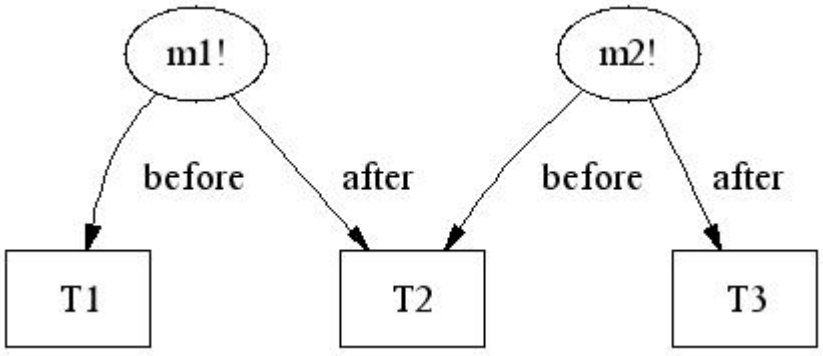


Time Axioms

These simply state that the event times are one after another, in the following fashion:

```
(describe (assert before *time-of-event-1 after *time-of-event-2))
```

```
(describe (assert before *time-of-event-2 after *time-of-event-3))
```

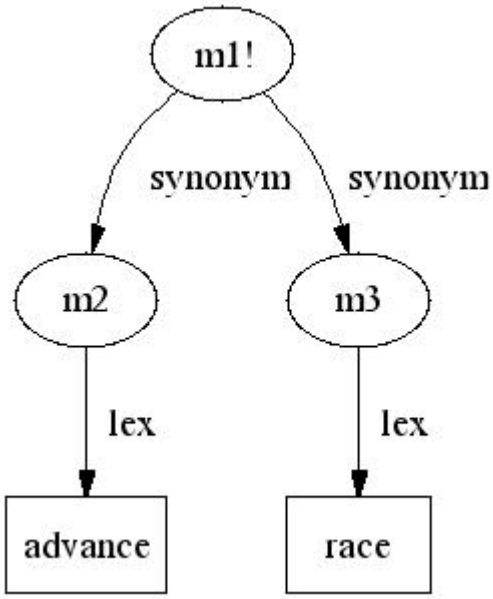


1.5 Background Knowledge

The background knowledge is like a person's common sense. Here is where the rules of logic and of language are kept. There are only two items that Cassie needs to connect 'quail' with 'halt'.

Racing towards something is close in meaning to advancing towards something

```
(describe (add synonym *advance synonym *race))
```



The case frame used here is synonym, which is not wholly correct. However, this is sufficient for the time being.

The Big Rule

As was previously discussed, representing causality (eg. The reason action Y occurs is because of action X) is quite a difficult issue. With the advice of Dr. Rapaport, we made the representation of causality simply be one big if-then rule with a lot of 'ifs'. The reason for representing the rule this way is that connected statements of the form:

if (if (if A then B) then C) then D

can really be represented as:

if (A and B and C) then D

One way of looking at this logic is by simply thinking of a *now* point in time. In the first form above, we consider the situation where the now point is moving from A to B to C and finally to D. In the second form, it can be thought of as the situation where event A and B and C has already occured (the now point is right after C). In the second form, we don't care how events A, B or C occured, just that they did in some past time. In both forms, when the now point is just after C, then the events A and B and C brings about D.

This simplification works great for this project and perhaps it may even be valid for a more general situation. However, more research is needed to really confirm that it is applicable in any situation.

The rule of inference used to make Cassie figure out that 'quail' is related to 'halt' is given here, but is presented in a rather specific form. The idea is this:

- IF agent A performs some action and
- agent B performs a *similar* action to A and
- some action occurs sometime after and
- agent A performs an action represented by a known word and
- agent B performs an action represented by an unknown word;
- THEN the action represented by the unknown word is likely to be a synonym of the known word.

Here is how such a rule is defined in SNePS:

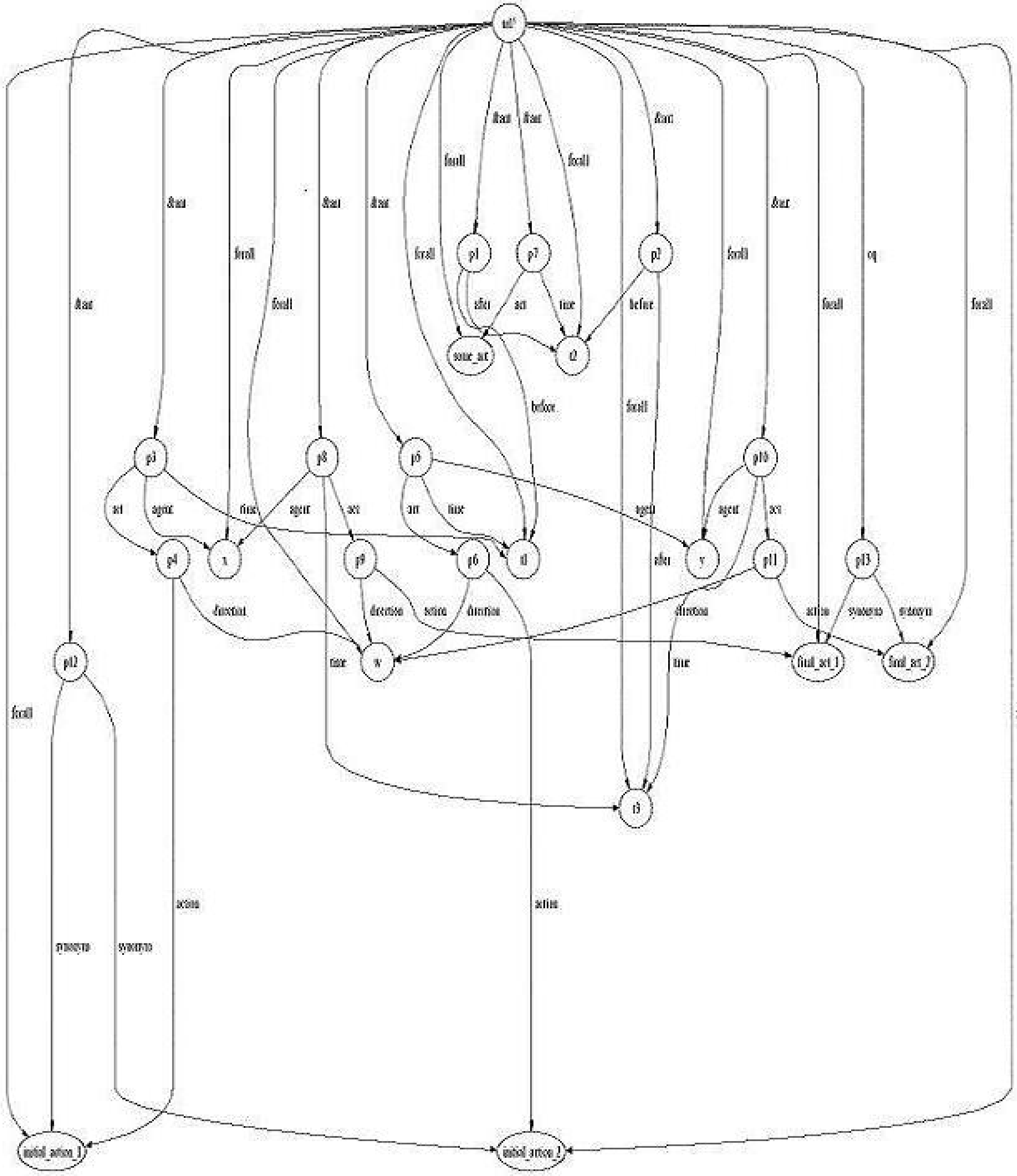
```
(describe (add forall ($x $y $w
                       $initial-action-1 $initial-action-2
                       $some-act
                       $final-act-1 $final-act-2
                       $t1 $t2 $t3)

                &ant   ((build before *t1 after *t2)
```

```
        (build before *t2 after *t3)
        (build agent *x
               act (build action *initial-action-1 direction *w)
               time *t1)
        (build agent *y
               act (build action *initial-action-2 direction *w)
               time *t1)
        (build act *some-act time *t2)
        (build agent *x
               act (build action *final-act-1)
               time *t3)
        (build agent *y
               act (build action *final-act-2)
               time *t3)
        (build synonym *initial-action-1
               synonym *initial-action-2))
 ; advancing and racing should be the same thing

cq     (build synonym *final-act-1 synonym *final-act-2)))
```

The following diagram is very big and rather hard to see. It is preferable to view the diagram electronically on a computer screen, and an electronic copy of the diagram can be made available on request.


p1
p7
p2
some_act
t2
p3
p8
p5
p10
p4
x
p9
p6
t1
y
p11
p13
p12
w
final_act_1
final_act_2
t3
initial_action_1
initial_action_2

1.6 Cassie's Results

Included in the appendix is a sample run of the verb definition algorithm on the passage representation and the background rule. The verb definition algorithm is not able to make any connection about the word 'quail' but is able to use the background rule to make 'quail' and 'halt' be synonyms of each other. The verb definition algorithm however, is not able to pick up the effect of this background rule firing. In fact, the verb definition algorithm is not even able to unify the fact that 'orcs can quail', instead being just left with 'something can quail'.

One reason that we suspect the verb definition algorithm is not able to unify to 'orcs can quail' is because of the way orcs is represented. We noticed that the verb definition algorithm is able to unify only with subjects that use the *member-class case frame*. In this project, orcs are represented using the *subclass-superclass case fram* and that might be a reason why the verb definition algorithm is not able to unify.

We ran a test, changing the representation of the orcs to use the *member-class case frame*. This time round, it was able to unify and give 'orcs can quail'. However, it is not able to infer any other facts than that.

1.7 Future Work

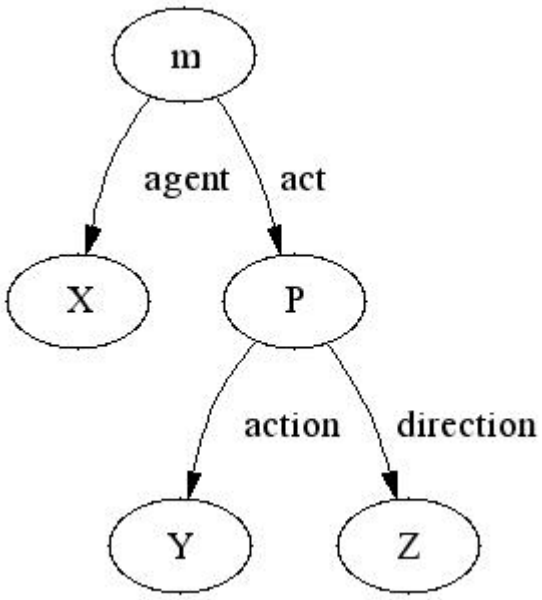
Given more time to work on the project, we would like to examine the verb definition algorithm closer, and hopefully see why a subclass-superclass case frame does not work. Also, we would like to see if we can add the feature that if some background rule adds a connection to the lexicon of the word, then the definition algorithm should pick up on the connection.

Overall however, it was a very satisfying project and gave a lot of insights into the field of contextual vocabulary acquisition. It seems that there is still much that can be done and needs to be done before we can truly have a computational cognitive agent that is able to fully infer the meaning of words from context.

2

Appendixes

2.1 New Case Frame



[[m]] is the proposition that agent [[x]] performs the act [[p]] which consists of doing an action [[y]] in the direction of [[z]].

2.2 SNePS Code

```
; Lunarso Sutanto
; CVA Project

(resetnet t)
```

```
^(setq snip:*infertrace* nil)

; Define case frames

; load all pre-defined relations:
(intext "/projects/rapaport/CVA/STN2/demos/rels")

; load all pre-defined path definitions:
(intext "/projects/rapaport/CVA/mkb3.CVA/paths/paths")


(define
        object propername
        lex
        member class
        subclass superclass
        object property
        agent act event
                action direction
        synonym
        time
        before after
)

 ^(load "/projects/rapaport/CVA/STN2/defun_verb.cl")



;; ============================
;; ORIGINAL PASSAGE, UNMODIFIED
;; ============================

; "The dark figure streaming with fire raced towards them. The orcs yelled and
; poured over the stone gangways. Then Boromir raised his horn and blew.
; Loud the challenge rang and bellowed, like the shout of many throats under
; the cavernous roof. For a moment the orcs *quailed* and the fiery shadow
; halted. Then the echoes died as suddenly as a flame blown out by a dark
; wind, and the enemy advanced again."


; The fiery shadow raced towards Boromir
; --------------------------------------
; "The dark figure streaming with fire" is changed to "fiery shadow"
; "them" is changed to simply "Boromir"
```

```
(describe (assert propername (build lex "Boromir") object #boromir))

(describe (assert member #shadow class (build lex "shadow")))


(describe (assert object *shadow property (build lex "fiery")))

(describe (assert agent *shadow
                  act (build action (build lex "race") = race
                  direction *boromir)
                  time #time-of-event-1))


; The orcs yelled and advanced towards Boromir
; ------------------------------------------------------------
; The orcs are a particular subclass of orcs,
; particularly the orcs in the Mines of Moria that were chasing after the
; Fellowship/ Boromir

(describe (assert subclass #orcs superclass (build lex "orcs")))

(describe (assert agent *orcs
                  act (build action (build lex "yell"))
                  time *time-of-event-1))

(describe (assert agent *orcs
                  act (build action (build lex "advance") = advance
                  direction *boromir)
                  time *time-of-event-1))


; Boromir blew his horn, which challenged the orcs
; ------------------------------------------------

(describe (assert member #horn class (build lex "horn")))

(describe (assert agent *boromir
                  act (build action (build lex "blow")
                  object *horn) = blowaction
                  time #time-of-event-2))

; Notice I labelled the act of blowing the horn as *blowaction, which will
; help in the SNePSUL-ization of "challenging the orcs"
; So the agent that challenged the orcs is the act of blowing the horn
```

```
(describe (assert agent *blowaction
                  act (build action (build lex "challenge")
                  object *orcs)
                  time *time-of-event-2))


; For a moment, the orcs quailed and the fiery shadow halted.
; -----------------------------------------------------------

(describe (assert agent *orcs
                  act (build action (build lex "quail")) = quail
                  time #time-of-event-3))
; QUAILED is the word that we are trying to define

(describe (assert agent *shadow
                  act (build action (build lex "halt")) = halt
                  time *time-of-event-3))
; This is the main connection, the fact that halted and quailed are nearly
; synonyms.


; TIME AXIOMS
; -----------

(describe (assert before *time-of-event-1 after *time-of-event-2))

(describe (assert before *time-of-event-2 after *time-of-event-3))



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

(describe (add synonym *advance synonym *race))

(describe (add forall ($x $y $w
                       $initial-action-1 $initial-action-2
                       $some-act
                       $final-act-1 $final-act-2
                       $t1 $t2 $t3)

               &ant   ((build before *t1 after *t2)
                       (build before *t2 after *t3)
                       (build agent *x
```

```
                        act (build action *initial-action-1 direction *w)
                        time *t1)
                 (build agent *y
                        act (build action *initial-action-2 direction *w)
                        time *t1)
                 (build act *some-act time *t2)
                 (build agent *x
                        act (build action *final-act-1)
                        time *t3)
                 (build agent *y
                        act (build action *final-act-2)
                        time *t3)
                 (build synonym *initial-action-1
                        synonym *initial-action-2))
          ; advancing and racing should be the same thing

        cq     (build synonym *final-act-1 synonym *final-act-2)))
```

2.3 Demo Output

```
============================================================
Starting image ‘/util/acl62/composer’
  with arguments ‘(-L /projects/snwiz/bin/sneps -e (sneps))’
  in directory ‘/home/csgrad/lsutanto’
  on machine ‘localhost’.

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.

   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.

   12/15/2003 1:31:30

* (demo "~/quail.demo")

 CPU time : 0.04

* ; Lunarso Sutanto
```

```
; CVA Project

(resetnet t)

Net reset


 CPU time : 0.00

* ^(
--> setq snip:*infertrace* nil)
nil



 CPU time : 0.01

*
; Define case frames

; load all pre-defined relations:
(intext "/projects/rapaport/CVA/STN2/demos/rels")
File /projects/rapaport/CVA/STN2/demos/rels is now the source of input.


 CPU time : 0.00

*

(a1 a2 a3 a4 after agent against antonym associated before cause class
 direction equiv etime event from in indobj instr into lex location manner
 member mode object on onto part place possessor proper-name property rel skf
 sp-rel stime subclass superclass subset superset synonym time to whole kn_cat)

 CPU time : 0.02

*

End of file /projects/rapaport/CVA/STN2/demos/rels


 CPU time : 0.02

*
; load all pre-defined path definitions:
```

```
(intext "/projects/rapaport/CVA/mkb3.CVA/paths/paths")
File /projects/rapaport/CVA/mkb3.CVA/paths/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 /projects/rapaport/CVA/mkb3.CVA/paths/paths


 CPU time : 0.01

*

(define
        object propername
        lex
        member class
        subclass superclass
        object property
        agent act event
                action direction
        synonym
        time
        before after
)
object is already defined.
lex is already defined.
member is already defined.
class is already defined.
subclass is already defined.
superclass is already defined.
object is already defined.
property is already defined.
agent is already defined.
act is already defined.
event is already defined.
action is already defined.
direction is already defined.
synonym is already defined.
time is already defined.
before is already defined.
after is already defined.

(object propername lex member class subclass superclass object property agent
 act event action direction synonym time before after)

 CPU time : 0.01

*
```

```
 ^(
--> load "/projects/rapaport/CVA/STN2/defun_verb.cl")
; Loading /projects/rapaport/CVA/STN2/defun_verb.cl
t



 CPU time : 0.06

*


;; ============================
;; ORIGINAL PASSAGE, UNMODIFIED
;; ============================

; "The dark figure streaming with fire raced towards them. The orcs yelled and
; poured over the stone gangways. Then Boromir raised his horn and blew.
; Loud the challenge rang and bellowed, like the shout of many throats under
; the cavernous roof. For a moment the orcs *quailed* and the fiery shadow
; halted. Then the echoes died as suddenly as a flame blown out by a dark
; wind, and the enemy advanced again."


; The fiery shadow raced towards Boromir
; --------------------------------------
; "The dark figure streaming with fire" is changed to "fiery shadow"
; "them" is changed to simply "Boromir"

(describe (assert propername (build lex "Boromir") object #boromir))

(m2! (object b1) (propername (m1 (lex Boromir))))

(m2!)

 CPU time : 0.00

*
(describe (assert member #shadow class (build lex "shadow")))

(m4! (class (m3 (lex shadow))) (member b2))

(m4!)

 CPU time : 0.01
```

```
*

(describe (assert object *shadow property (build lex "fiery")))

(m6! (object b2) (property (m5 (lex fiery))))

(m6!)

 CPU time : 0.00

*
(describe (assert agent *shadow
                  act (build action (build lex "race") = race
                  direction *boromir)
                  time #time-of-event-1))

(m9! (act (m8 (action (m7 (lex race))) (direction b1))) (agent b2) (time b3))

(m9!)

 CPU time : 0.00

*

; The orcs yelled and advanced towards Boromir
; ------------------------------------------------------------
; The orcs are a particular subclass of orcs,
; particularly the orcs in the Mines of Moria that were chasing after the
; Fellowship/ Boromir

(describe (assert subclass #orcs superclass (build lex "orcs")))
(m11! (subclass b4) (superclass (m10 (lex orcs))))

(m11!)

 CPU time : 0.00

*

(describe (assert agent *orcs
                  act (build action (build lex "yell"))
                  time *time-of-event-1))

(m14! (act (m13 (action (m12 (lex yell))))) (agent b4) (time b3))
```

```
(m14!)

 CPU time : 0.00

*
(describe (assert agent *orcs
                  act (build action (build lex "advance") = advance
                  direction *boromir)
                  time *time-of-event-1))

(m17! (act (m16 (action (m15 (lex advance))) (direction b1))) (agent b4)
 (time b3))

(m17!)

 CPU time : 0.00

*

; Boromir blew his horn, which challenged the orcs
; ------------------------------------------------

(describe (assert member #horn class (build lex "horn")))

(m19! (class (m18 (lex horn))) (member b5))

(m19!)

 CPU time : 0.01

*
(describe (assert agent *boromir
                  act (build action (build lex "blow")
                  object *horn) = blowaction
                  time #time-of-event-2))

(m22! (act (m21 (action (m20 (lex blow))) (object b5))) (agent b1) (time b6))

(m22!)

 CPU time : 0.00

*
; Notice I labelled the act of blowing the horn as *blowaction, which will
```

```
; help in the SNePSUL-ization of "challenging the orcs"
; So the agent that challenged the orcs is the act of blowing the horn

(describe (assert agent *blowaction
                  act (build action (build lex "challenge")
                  object *orcs)
                  time *time-of-event-2))

(m25! (act (m24 (action (m23 (lex challenge))) (object b4)))
 (agent (m21 (action (m20 (lex blow))) (object b5))) (time b6))

(m25!)

 CPU time : 0.00

*

; For a moment, the orcs quailed and the fiery shadow halted.
; -----------------------------------------------------------

(describe (assert agent *orcs
                  act (build action (build lex "quail")) = quail
                  time #time-of-event-3))

(m28! (act (m27 (action (m26 (lex quail))))) (agent b4) (time b7))

(m28!)

 CPU time : 0.00

* ; QUAILED is the word that we are trying to define

(describe (assert agent *shadow
                  act (build action (build lex "halt")) = halt
                  time *time-of-event-3))

(m31! (act (m30 (action (m29 (lex halt))))) (agent b2) (time b7))

(m31!)

 CPU time : 0.00

* ; This is the main connection, the fact that halted and quailed are nearly
; synonyms.
```

```
; TIME AXIOMS
; -----------

(describe (assert before *time-of-event-1 after *time-of-event-2))

(m32! (after b6) (before b3))

(m32!)

 CPU time : 0.00

*
(describe (assert before *time-of-event-2 after *time-of-event-3))

(m33! (after b7) (before b6))

(m33!)

 CPU time : 0.00

*


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

(describe (add synonym *advance synonym *race))

(m34! (synonym (m15 (lex advance)) (m7 (lex race))))

(m34!)

 CPU time : 0.00

*
(describe (add forall ($x $y $w
                      $initial-action-1 $initial-action-2
                      $some-act
                      $final-act-1 $final-act-2
                      $t1 $t2 $t3)

               &ant   ((build before *t1 after *t2)
                       (build before *t2 after *t3)
```

```
                        (build agent *x
                               act (build action *initial-action-1 direction *w)
                               time *t1)
                        (build agent *y
                               act (build action *initial-action-2 direction *w)
                               time *t1)
                        (build act *some-act time *t2)
                        (build agent *x
                               act (build action *final-act-1)
                               time *t3)
                        (build agent *y
                               act (build action *final-act-2)
                               time *t3)
                        (build synonym *initial-action-1
                               synonym *initial-action-2))
                 ; advancing and racing should be the same thing

                cq     (build synonym *final-act-1 synonym *final-act-2)))

(m56! (synonym (m29 (lex halt)) (m26 (lex quail))))
(m55! (act (m38 (action (m7 (lex race))))) (time b3))
(m54! (act (m42 (action (m15 (lex advance))))) (time b3))
(m53! (act (m45 (action (m20 (lex blow))))) (time b6))
(m52! (act (m48 (action (m23 (lex challenge))))) (time b6))
(m51! (act (m30 (action (m29)))) (time b7))
(m50! (act (m27 (action (m26)))) (time b7))
(m49! (act (m48)) (agent (m21 (action (m20)) (object b5))) (time b6))
(m47! (act (m24 (action (m23)) (object b4))) (time b6))
(m46! (act (m45)) (agent b1) (time b6))
(m44! (act (m21)) (time b6))
(m43! (act (m42)) (agent b4) (time b3))
(m41! (act (m16 (action (m15)) (direction b1))) (time b3))
(m40! (act (m13 (action (m12 (lex yell))))) (time b3))
(m39! (act (m38)) (agent b2) (time b3))
(m37! (act (m8 (action (m7)) (direction b1))) (time b3))
(m36! (after b7) (before b3))
(m35! (forall v11 v10 v9 v8 v7 v6 v5 v4 v3 v2 v1)
 (&ant (p12 (synonym v5 v4))
  (p11 (act (p10 (action v8))) (agent v2) (time v11))
  (p9 (act (p8 (action v7))) (agent v1) (time v11)) (p7 (act v6) (time v10))
  (p6 (act (p5 (action v5) (direction v3))) (agent v2) (time v9))
  (p4 (act (p3 (action v4) (direction v3))) (agent v1) (time v9))
  (p2 (after v11) (before v10)) (p1 (after v10) (before v9)))
 (cq (p13 (synonym v8 v7))))
(m34! (synonym (m15) (m7)))
```

```
(m33! (after b7) (before b6))
(m32! (after b6) (before b3))
(m31! (act (m30)) (agent b2) (time b7))
(m28! (act (m27)) (agent b4) (time b7))
(m17! (act (m16)) (agent b4) (time b3))
(m14! (act (m13)) (agent b4) (time b3))
(m9! (act (m8)) (agent b2) (time b3))

(m56! m55! m54! m53! m52! m51! m50! m49! m47! m46! m44! m43! m41! m40! m39!
 m37! m36! m35! m34! m33! m32! m31! m28! m17! m14! m9!)

 CPU time : 0.07

*

End of /home/csgrad/lsutanto/cse663/quail.demo demonstration.


* (describe (find synonym ?x))

(m56! (synonym (m29 (lex halt)) (m26 (lex quail))))
```

Notice that Cassie is able to make the inference that quail and halt are synonyms.

```
(p13 (synonym v8 v7))
(p12 (synonym v5 v4))
(m34! (synonym (m15 (lex advance)) (m7 (lex race))))

(m56! p13 p12 m34!)

 CPU time : 0.00

* ^(defineVerb 'quail)

--> "You want me to define the verb 'quail'.

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

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



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:



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.
"
```

But the verb definition algorithm is not yet able to, and cannot even unify the action with an actor.

Bibliography

[Alm95] Michael J. Almeida. Time in narratives. In Lynne E. Hewitt Judith F. Duchan, Gail A. Bruder, editor, *Deixis in Narrative: A Cognitive Science Perspective*. Lawrence Erlbaum Associates, INC, 1995.

[dic03] dictionary.com, 2003.

[MH69] John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[RE00] William J. Rapaport and Karen Ehrlich. A computational theory of voculary acquisition. In L. Iwanska and Stuart C. Shapiro, editors, *Natural Language Processing and Knowledge Representation*, pages 347–375, Menlo Park, CA, 2000. AAAI Press.

[RK02] William J. Rapaport and Michael Kibby. Contextual vocabulary acquisition: From algorithm to curriculum, 2002.

[Sha99] Murray Shanahan. The event calculus explained. *Artificial Intelligence Today*, (1600):409–430, 1999.

[SNe96] SNeRG. A dictionary of sneps case frames, 1996.

[SR87] Stuart C. Shapiro and William J. Rapaport. Sneps considered as a fully intensional propositional semantic network. *The Knowledge Frontier: Essays in the Representation Knowledge*, 1987.

[Tol54] J.R.R. Tolkien. *The Lord of the Rings*. Houghton Mifflin, 1954.