Contextual Vocabulary Acquisition

The meaning of xeric from context

Yalei Song

CSE 499

13 May 2011

Abstract:

No matter how many books and newspaper you have read, it is impossible that you can get to know every single (English) word in the world. "The second Edition of the 20-volume Oxford English Dictionary contains full entries for 171,476 words in current use, and 47,156 obsolete words", plus those new words are created every day. What do people usually do when they come across an unfamiliar word? Apparently not many people carry a dictionary wherever they go. People usually don't look it up, but try to "figure it out" by using their background knowledge and the information from the context. In this report, Contextual Vocabulary Acquisition (CVA) will be used on SNePS to teach Cassie (a computational SNePS based agent) in order to simulate the way that human beings think.

A description of CVA project

CVA—Contextual Vocabulary Acquisition, in other words, getting the meaning of a word from the context. However, it is not just simply guessing the meaning of a word, but using reader's prior knowledge (background knowledge), textual clues, and any other knowledge that may help readers to acquire the meaning. There are many ways to do so. One is using reader's background knowledge. For example, if you know what the word "context" means, you should probably know the meaning of "contextual", even though you may never have used it, because you know the postfix "-tual" is for adjective. One is deducing from the context. For example, the context "He is such a short and tiny guy that his friends call him 'midget'.", for the people who don't know the word "midget", it is not hard to find out from the context that midget means something/someone who is short and tiny.

How can we make use of CVA in computer science field? Imagine a computer (like robot) that can understand a word by itself. Computers are made by human beings, but no matter how smart they are, they are just doing what humans want them to do. CPU is not a brain. But we can implement the artificial intelligence algorithms to CVA to simulate the process that human beings think, that is, instead of “guessing” the meaning of a word, we are making computers to “compute” it.

A description of SNePS (http://www.cse.buffalo.edu/sneps/)

SNePS is used to understand human language. Cassie, a computerized cognitive agent, is designed to be able to understand natural language by implementing cognitive architectures.

In this project, I used SNePSUL to represent background knowledge and context information by using different case frames.

Word and Context

In this experiment, the word is “xeric”:

> When city of Lakewood horticulture manager Greg Foreman inherited the park at Kendrick Lake in 2002, there were three garden beds and an acre of dead turf. But instead of seeing the park as half empty, he saw the

opportunity to design drought-tolerant gardens to promote the benefits of planting native and ***xeric*** plants.

---Torpey, Jodi (2010), "Kendrick Lake Park a Xeric Wonder", *The Denver Post* (21 May)

Human protocols

I interviewed four people to observe how they figure out the meaning of an unfamiliar word. I summarized the results in to two models.

Model A:

Greg wants a better garden

➔ Greg wants improve the diversity of the plants in the garden

➔ native plants and xeric plants are the only two kinds of plants that are mentioned in the context

➔ xeric and native have the opposite meaning

➔ xeric means "non-native"

Model B:

Greg wants a better garden

➔ design a drought-tolerant garden

➔ xeric plants can grow with little water

➔ xeric means “drought-tolerant”

It is interesting to see how different people see things differently. Model A and Model B are almost irrelevant. Model A uses “native”as a hint to derive “xeric” but doesn't think “xeric”may be related to “drought-tolerant”. Model B derive the meaning of “xeric” directly from “drought-tolerant”, which has nothing to do with “native”.

Both of them seem reasonable and logical. However, if we look it up in the dictionary, this is what we will get: (of an environment or habitat) containing little moisture; very dry. Compare with HYDRIC and MESIC. (Oxford Dictionary). So Model B figures out the correct definition of “xeric”.

Even though Model A did not happen to be the correct way to think of it, I still think it is a case and logic that is worth studying. Due to the lack of time, I will leave Model B for future work.

SNePS representation

Step 1: Using Emacs

Using Emacs is not the only way to do this project, vi, nano and some other editors can also be used for lisp. I thought it is interesting and useful to use Emacs because of its easy-use, easy-edit feature. It is "the extensible, customizable, self-documenting, real-time display editor" as describe in Emacs manual.

Here is the instruction how you set up and start Emacs.

http://www.cse.buffalo.edu/sneps/instructions.html

Note: There is confusion about the Emacs's M-<key>. M is the Meta, it is key with symbol "◆" on Sun Microsystem's keyboard. It corresponds to the Alt key on today's PC keyboard(most widely used keyboard).

Have a quick look of the Emacs manual, you may find some features you like. I found it is very efficient and easy to edit when you are running SNePS on Emacs, and the layout of text is clear and delightful compare to typing on a terminal window.

Another thing that I like about Emacs is, you only need to load SNePS once after you open a Emacs window. It is much more convenient and less troublesome as if you are at the demo stage.

Step 2: Extracting information from the context

When city of Lakewood horticulture manager Greg Foreman inherited the park at Kendrick Lake in 2002, there were three garden beds and an acre of dead turf. But instead of seeing the park as half empty, he saw the opportunity to design drought-tolerant gardens to promote the benefits of planting native and ***xeric*** plants.

Not all the information from the above context is needed in Model A.

We can simplify the context into this: Greg wants a better garden. There are native and xeric plants.

Step3: Implementing background knowledge (details will be covered in the next section)

Background knowledge is the knowledge that a reader needs to understand a context and deduce information.

Different people have different background knowledge. For example, the mass-energy equivalence $E = mc^2$ give people who learned physics a lot of information, but to 5 years old children in the kindergarten it is just consist of some random symbols.

So what background knowledge does the computational cognitive agent need?

BK 1:

If Greg wants a better garden, he wants plants.

This connects Greg, better garden and plants.

BK 2:

If x is a member of class S, and has property N; If y is a member of class S, and has property X; Q is unknow. Then if z is a member of class S, it has property N xor X.

This can also be represented use first-order logic as:

$\forall$ x,y[(x ∈ S ∧ N(x) ∧ y ∈S ∧ X(y) ∧ Unknow(X)]

-> $\forall$z [z∈S -> N(z) xor X(z)]

BK 3:

If that if z is a member of class plants, then z is native xor xeric; then that if z is xeric, then it's not the case that it is native.

This combines BK 3 with information from the context.

;If that if z is a member of class plants, then z is native xor xeric; then that if z is xeric, then it's not the case that it is native.

$\forall$ z[z∈plants -> Native(z) xor Xeric(z)]

-> z [Xeric(z) -> ~Native(z)]

Take a close look at BK2 and BK3, you will notice that BK2 triggers BK3.

This is also a simulation of how human beings connect their background knowledge to get the result/answer.

Step 4: Coding (for the complete demo refer to Appendix A, for the sample run refer to Appendix B)

When people are given a strange word, they will use the background knowledge they already have and to read through the context. So background knowledge comes before context in the coding part.

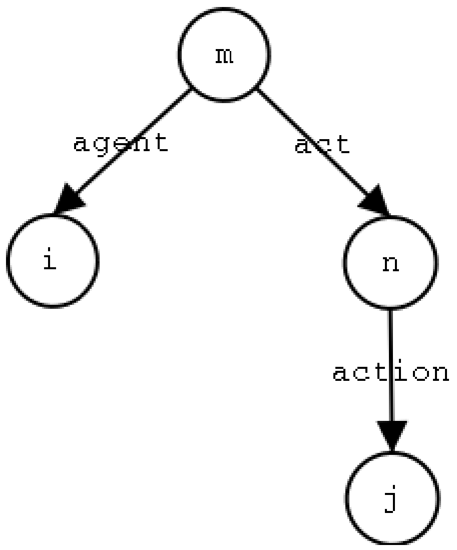
Background knowledge(BK):

BK 1:

```
; BACKGROUND KNOWLEDGE:

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

;If Greg wants a better garden, he wants plants.

(describe (assert forall ($p $g $pl)

                    &ant (build agent *p act (build action (build lex wants) object *g))
                    &ant (build member *g class (build lex "better garden"))

                    &ant (build member *pl class (build lex plants))

                    cq (build agent *p act (build action (build lex wants) object *pl))))
```

Here I used case frame agent/act/action, member/class, lex and forall/ant/cq.

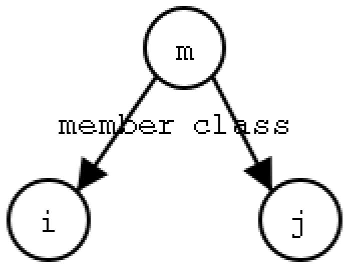
1. Agent/act/action



Semantics:

[[m]] is the proposition that agent [[i]] performs action [[j]]

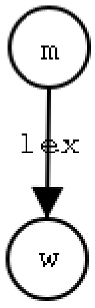
2. Member/class



Semantics:

[[m]] is the proposition that [[i]] is a member of the class [[j]].

3. lex



Semantics:

[[m]] is the concept expressed by uttering [[w]]

4. Forall/ant/cq

Forall —universal quantifier (∀)

Ant ---antecednet of an if-then rule

Cq ---consequent of the rule

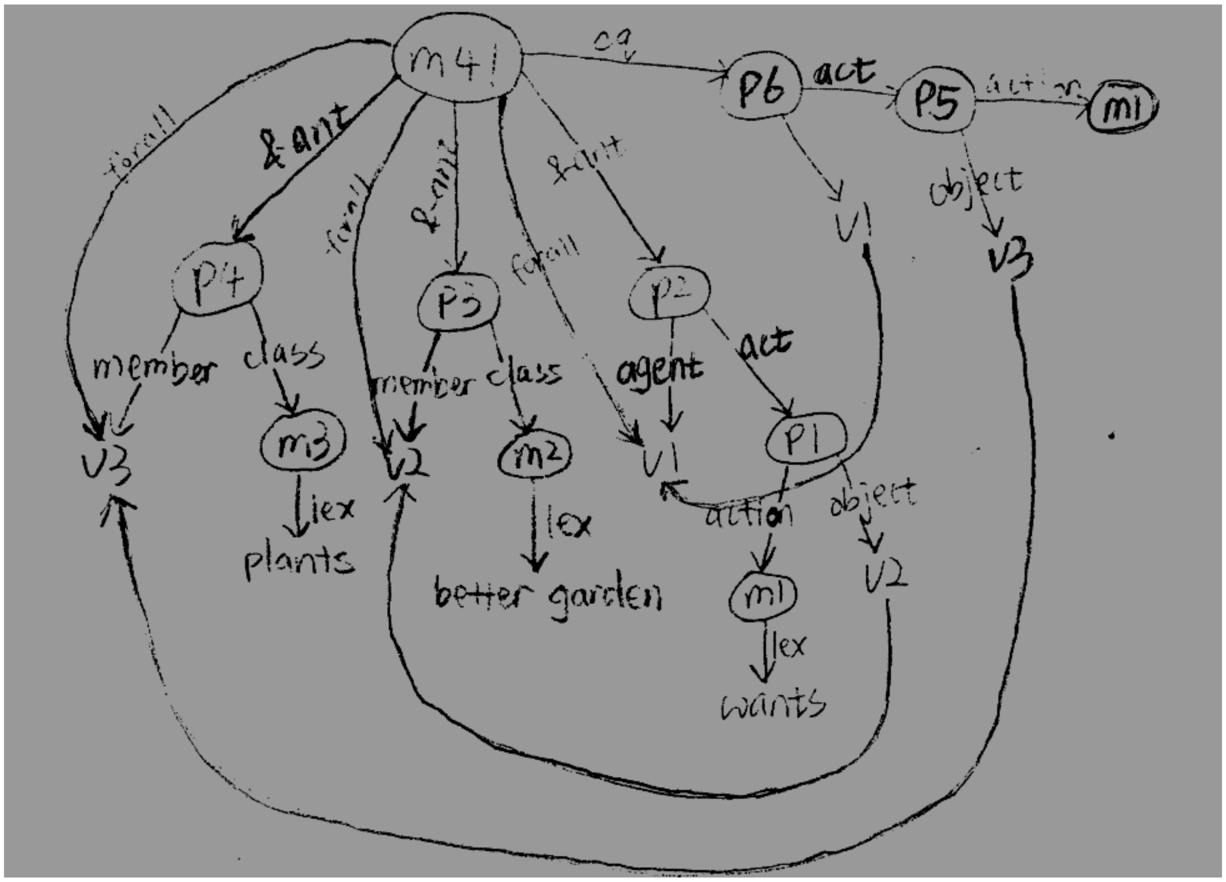


Fig 1.Network Diagram—BK1 (If Greg wants a better garden, he wants plants)

From the Fig 1., we can tell that there are 3 antecedents, and they all have the universal quantifier.

3 ants:

a. if v3 is a member of class of plants

b. if v2 is a member of class better garden

c. if v1 wants v2

cq: then v1 wants v2

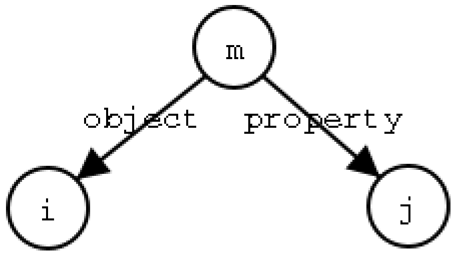
BK 2:

If x is a member of class S, and has property N; If y is a member of class S, and has property X; X is unknown. Then if z is a member of class S, it has property N xor X.

```
(describe (assert forall ($x $N $y $X $S)
             &ant (build member *x class *S)
             &ant (build object *x property *N)
             &ant (build member *y class *S)
            &ant (build object *y property *X)
            &ant (build object *X property (build lex unknown))
              cq (build forall $z
                    ant (build member *z class *S)
                    cq  (build min 1 max 1
                                   arg ((build object *z property *N)
                                         (build object *z property *X))))))
```

In this rule I used member/class, lex, forall/ant/cq and another case frame object/property and min/max.

Object/property



Semantics:

[[m]] is the proposition that [[i]] has the property [[j]].

Min X max Y

Min X max Y arg (P1 … Pn) represents: at least X and at most Y of the propositions P1, … , Pn.

For example we have 2 propositions, n = 2.

If X =0, Y=0 means negation. If X=1, Y = 1 means exclusive disjunction(xor). If X =1, Y=2 means inclusive disjunction(or).

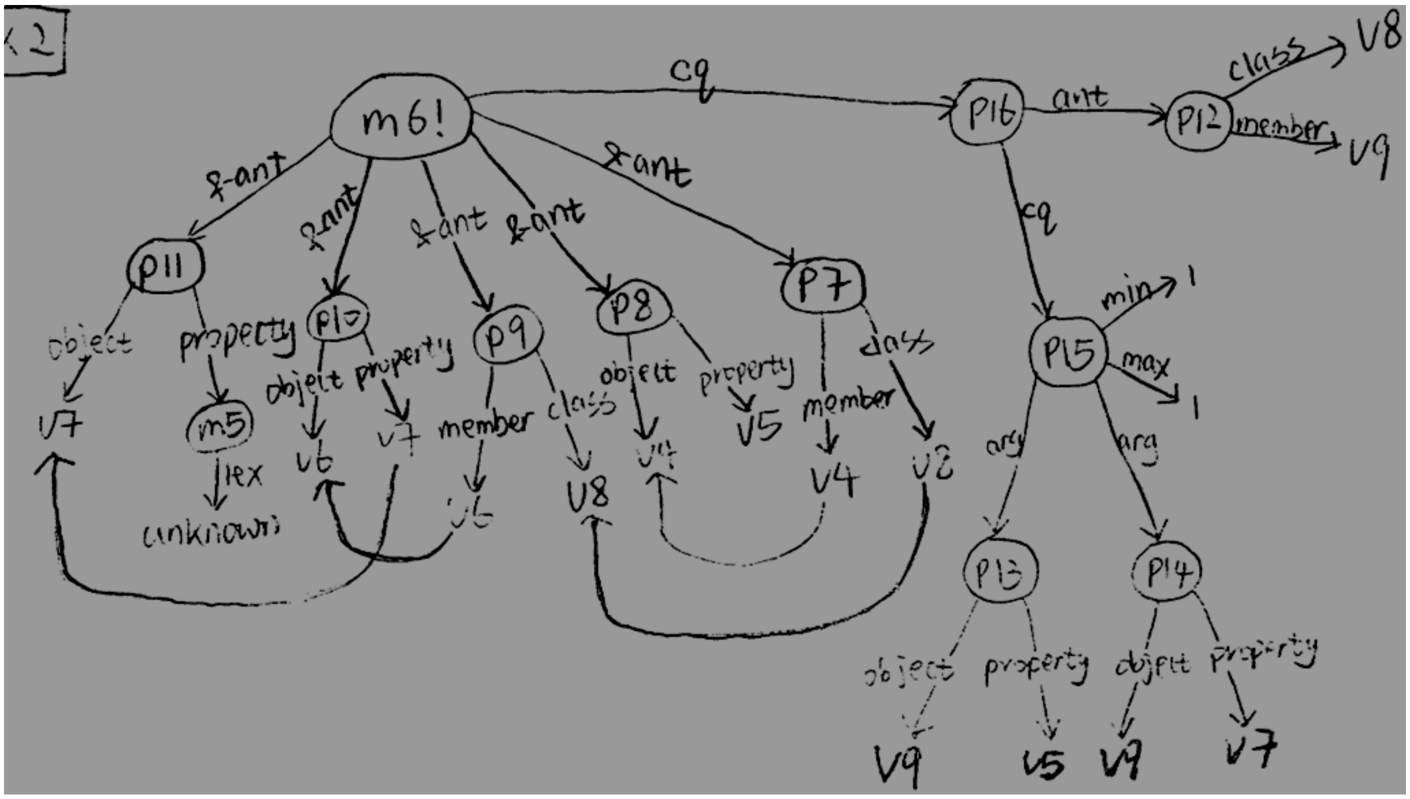


Fig 2.Network Diagram—BK2 (If x is a member of class S, and has property N; If y is a member of class S, and has property X; X is unknown. Then if z is a member of class S, it has property N xor X.)

5 ants:

a. if object v4 has the property v5

b. if v4 is a member of class v8

c. if object v6 has the property v7

d. if v6 is also a member of class v8

e. if v7 has the property v8

cq: this consequent has an inner antecedent and consequent.

Ant: if v8 is a member of class v9

Cq: then v9 has property v5 xor v9 has property v7.

BK 3:

```
;If it is the case that if z is a member of class plants, then z is native xor xeric, then
that if z is xeric, then it's not the case that it is native.

(describe (assert

        ant (build forall ($z)

                ant (build member *z class (build lex plants))

                 cq (build min 1 max 1

                        arg (build object *z property (build lex native))

                         arg (build object *z property (build lex xeric))))

        cq (build forall ($z)

               ant (build object *z property (build lex xeric))
```

cq (build min 0 max 0

arg (build object *z property (build lex native))))))

In this rule, case frames forall/ant/cq, member/class, object/property, lex and min/max are used.

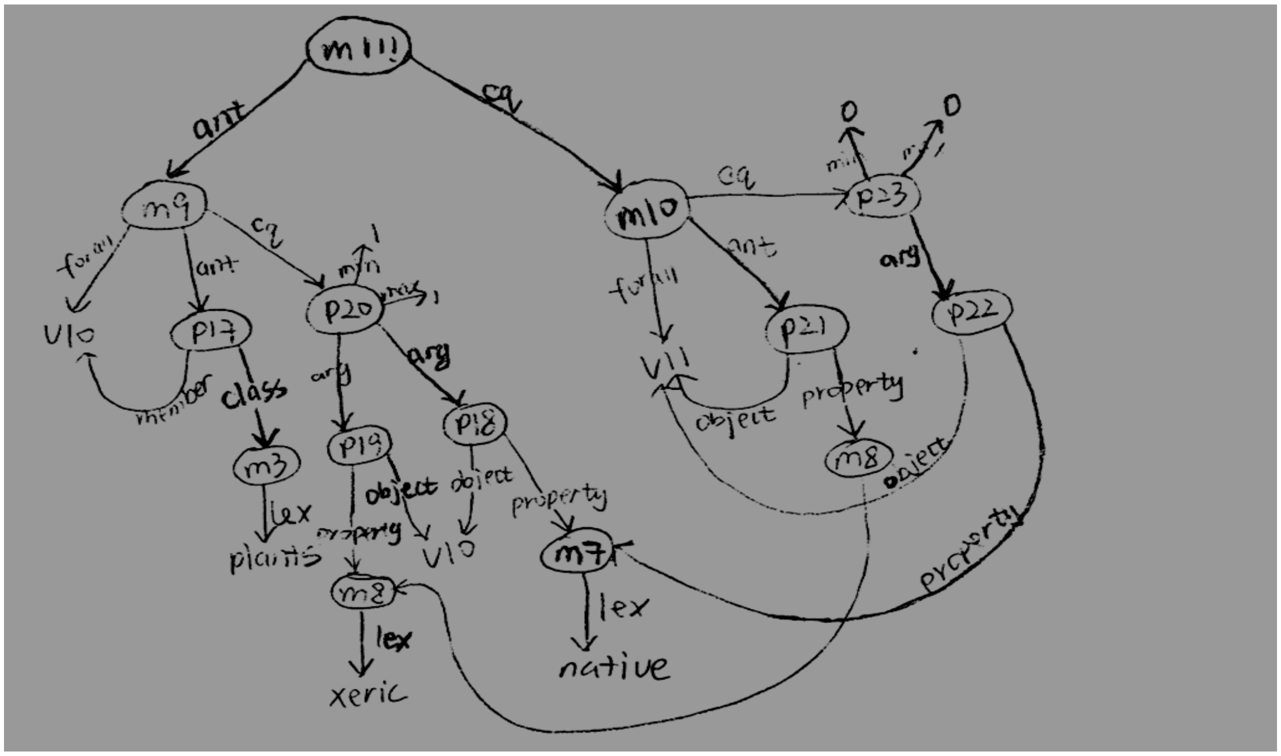


Fig 3. BK3—(If it is the case that if z is a member of class plants, then z is native xor xeric, then that if z is xeric, then it's not the case that it is native)

Ant: this antecedent has an inner antecedent and consequence

Ant: if v10 is a member of class plants

Cq: then v10 has property xeric xor v10 has property native

Cq: this consequence has an inner antecedent and consequence

Ant: if v11 has property xeric

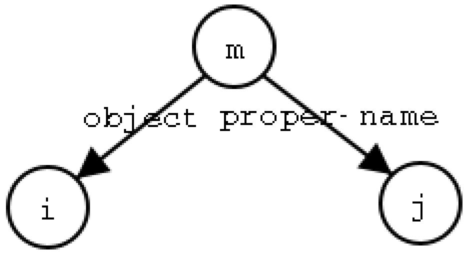
Cq, then it is not the case that v11 has property native

Context (CT):

CT 1:

```
; Greg wants a better garden.
(describe (add agent #greg act (build action (build lex wants) object
#bg)))


;*greg is Greg
(describe (add object *greg proper-name (build lex Greg)))


;*bg is a better garden
(describe (add member *bg class (build lex "better garden")))
```

Object/proper-name



Semantics:

[[m]] is the proposition that [[i]] is called by the proper name [[j]].

Case frames act/agent/action, member/class, lex and object proper-name are used.

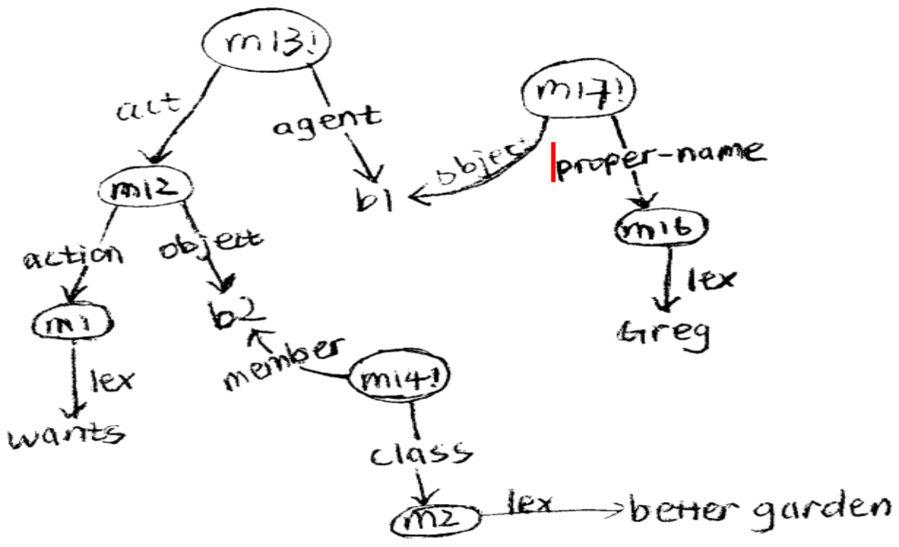


Fig 4. CT1(Greg wants a better garden)

Agent b1 wants b2.

B1 is called by the proper name Greg.

B1 is a member of class better garden.

CT 2:

;There are native plants.

;1. It's native.

(describe (add object #nplant property (build lex native)))

;2. It's plant.

(describe (add member *nplant class (build lex plants)))

Case frames object/property, member/class, lex are used.

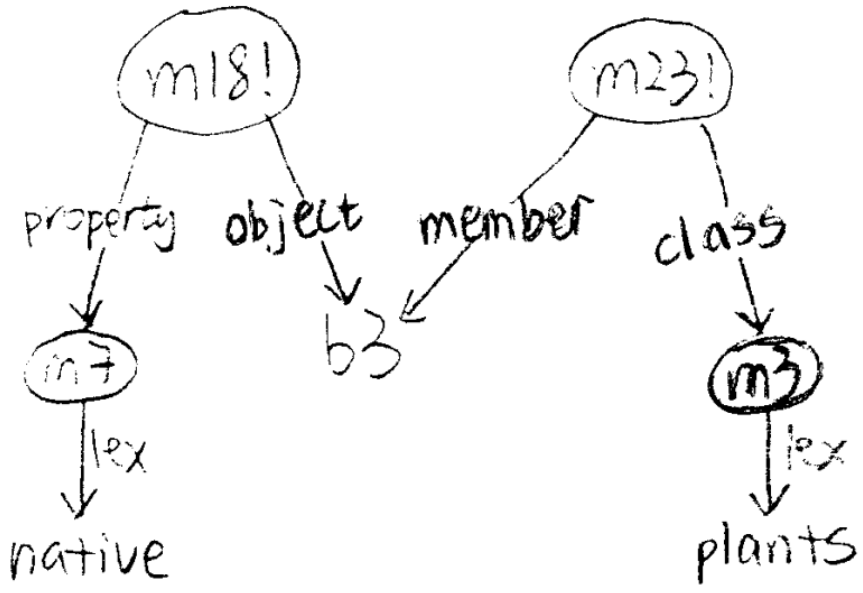


Fig 5. CT2(There are native plants)

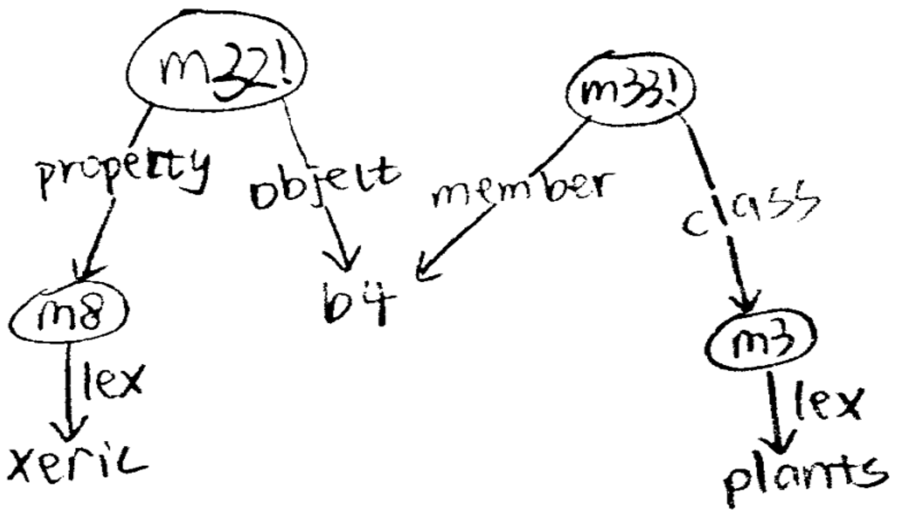
B3 has property native.

B3 is a member of class plants.

CT3:

```
;There are xeric plants.

;1. It's xeric.

(describe (add object #xplant property (build lex xeric)))

;2. It's plant.

(describe (add member *xplant class (build lex plants)))
```

Very similar with CT2, this part of context is also interpreted with case frames object/property, member/class and lex.



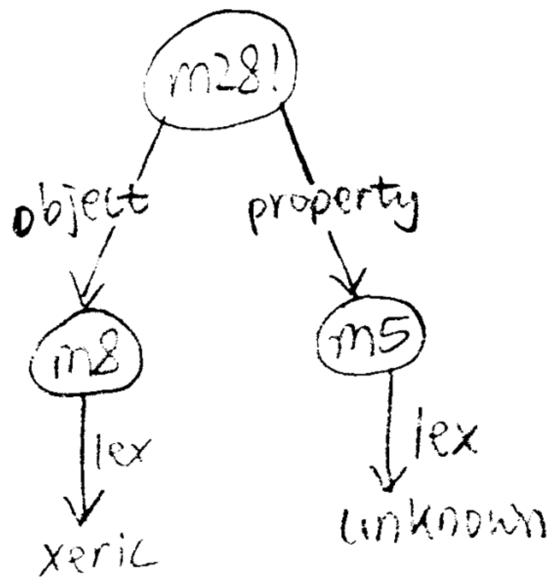
B4 has property xeric.

B4 is a member of class plants.

CT 4:

;Xeric plants is unknown.

```
(describe (add object (build lex xeric) property (build lex unknown)))
```



Achievement

As we can tell from the sample run, CASSIE inferred that xeric is being non-native, which is what we were expecting.

The rules are successfully fires and triggers which lead to a good result. Context information is well represented too. Model A is well accomplished.

Future work

1. Model A is straight forward and simpler compare to Model B. I believe it will be very helpful and interesting to study for Model B in the future.

2. In the sample run, I didn't ask Cassie any questions, I get the result from the information Cassie inferred instead. I was not sure about how to ask Cassie correctly. I believe there must be a way to ask Cassie the right question in order to get the result directly.

3. (defineNoun "WORD") and (defineVerb "WORD") are used to ask Cassie what "WORD" means. However, the word I use is an adjective, which has no existed algorithm. It would be a big improvement and more competed if there is one for adjective.

Appendix A: demo file

```
; =======================================================================
; FILENAME:    Xeric.demo
; DATE:        April 16.2011
; PROGRAMMER:  Yalei Song

;; this template version:     snepsul-template.demo-20061005.txt

; 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 "Xeric.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)

; Clear the SNePS network:
(resetnet t)

; OPTIONAL:
; UNCOMMENT THE FOLLOWING CODE TO TURN FULL 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-CHANNELS* anysent)
;                 (when (isopen.ch ch)
;                 (setq anysent
;                       (or (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")

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

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

; BACKGROUND KNOWLEDGE:
; =====================
;If Greg wants a better garden, he wants plants.
(describe (assert forall ($p $g $pl)
                 &ant (build agent *p act (build action (build lex wants) object
*g))
                 &ant (build member *g class (build lex "better garden"))
                 &ant (build member *pl class (build lex plants))
                 cq (build agent *p act (build action (build lex wants) object
*pl))))



;If x is a member of class S, and has property N; If y is a member of class S,
and has property X; X is unknown. Then if z is a member of class S, it has
property N xor X.
(describe (assert forall ($x $N $y $X $S)
          &ant (build member *x class *S)
          &ant (build object *x property *N)
          &ant (build member *y class *S)
           &ant (build object *y property *X)
           &ant (build object *X property (build lex unknown))
          cq (build forall $z
             ant (build member *z class *S)
             cq  (build min 1 max 1
                arg ((build object *z property *N)
                     (build object *z property *X))))))

;If that if z is a member of class plants, then z is native xor xeric; then that
if z is xeric, then it's not the case that it is native.
(describe (assert
          ant (build forall ($z)
                     ant (build member *z class (build lex plants))
                     cq (build min 1 max 1
                              arg (build object *z property (build lex native))
                               arg (build object *z property (build lex
xeric))))
          cq (build forall ($z)
             ant (build object *z property (build lex xeric))
                 cq (build min 0 max 0
                         arg (build object *z property (build lex native))))))

; CASSIE READS THE PASSAGE:
; ========================
; Greg wants a better garden.
(describe (add agent #greg act (build action (build lex wants) object #bg)))

;*greg is Greg
(describe (add object *greg proper-name (build lex Greg)))

;*bg is a better garden
(describe (add member *bg class (build lex "better garden")))

;There are native plants.
;1. It's native.
```

```
(describe (add object #nplant property (build lex native)))
;2. It's plant.
(describe (add member *nplant class (build lex plants)))

;There are xeric plants.
;1. It's xeric.
(describe (add object #xplant property (build lex xeric)))
;2. It's plant.
(describe (add member *xplant class (build lex plants)))

;Xeric plants is unknown.
(describe (add object (build lex xeric) property (build lex unknown)))

; Ask Cassie what "WORD" means:
```

Appendix B: Script of running demo

```
* ; ==========================================================================
; FILENAME:     Xeric.demo
; DATE:         April 16.2011
; PROGRAMMER:   Yalei Song

;; this template version:       snepsul-template.demo-20061005.txt

; 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 "Xeric.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)
nil



 CPU time : 0.00

*
; Clear the SNePS network:
(resetnet t)

Net reset


 CPU time : 0.00

*
; OPTIONAL:
; UNCOMMENT THE FOLLOWING CODE TO TURN FULL 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-CHANNELS* anysent)
;                (when (isopen.ch ch)
;                (setq anysent
;                      (or (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 file /projects/rapaport/CVA/STN2/demos/rels.


 CPU time : 0.01

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


 CPU time : 0.00

*
; BACKGROUND KNOWLEDGE:
; ====================
;If Greg wants a better garden, he wants plants.
(describe (assert forall ($p $g $pl)
                 &ant (build agent *p act (build action (build lex wants) object
*g))
                 &ant (build member *g class (build lex "better garden"))
                 &ant (build member *pl class (build lex plants))
                 cq (build agent *p act (build action (build lex wants) object
*pl))))

(m4! (forall v3 v2 v1)
 (&ant (p4 (class (m3 (lex plants))) (member v3))
  (p3 (class (m2 (lex better garden))) (member v2))
  (p2 (act (p1 (action (m1 (lex wants))) (object v2))) (agent v1)))
 (cq (p6 (act (p5 (action (m1)) (object v3))) (agent v1))))

(m4!)

 CPU time : 0.00

*


;If x is a member of class S, and has property N; If y is a member of class S,
and has property X; X is unknown. Then if z is a member of class S, it has
property N xor X.
(describe (assert forall ($x $N $y $X $S)
          &ant (build member *x class *S)
          &ant (build object *x property *N)
          &ant (build member *y class *S)
          &ant (build object *y property *X)
          &ant (build object *X property (build lex unknown))
          cq (build forall $z
             ant (build member *z class *S)
             cq  (build min 1 max 1
                arg ((build object *z property *N)
                     (build object *z property *X))))))

(m6! (forall v8 v7 v6 v5 v4)
 (&ant (p11 (object v7) (property (m5 (lex unknown))))
  (p10 (object v6) (property v7)) (p9 (class v8) (member v6))
  (p8 (object v4) (property v5)) (p7 (class v8) (member v4)))
 (cq
  (p16 (forall v9) (ant (p12 (class v8) (member v9)))
   (cq
    (p15 (min 1) (max 1)
     (arg (p14 (object v9) (property v7))
      (p13 (object v9) (property v5))))))))

(m6!)

 CPU time : 0.00

*
;If that if z is a member of class plants, then z is native xor xeric; then that
if z is xeric, then it's not the case that it is native.
(describe (assert
          ant (build forall ($z)
                        ant (build member *z class (build lex plants))
                        cq (build min 1 max 1
                                  arg (build object *z property (build lex
native))
                                  arg (build object *z property (build lex
xeric))))
          cq (build forall ($z)
             ant (build object *z property (build lex xeric))
                  cq (build min 0 max 0
                            arg (build object *z property (build lex native))))))

(m11!
 (ant
  (m9 (forall v10) (ant (p17 (class (m3 (lex plants))) (member v10)))
   (cq
    (p20 (min 1) (max 1)
     (arg (p19 (object v10) (property (m8 (lex xeric))))
      (p18 (object v10) (property (m7 (lex native))))))))))
 (cq
  (m10 (forall v11) (ant (p21 (object v11) (property (m8))))
   (cq
    (p23 (min 0) (max 0) (arg (p22 (object v11) (property (m7)))))))))

(m11!)

 CPU time : 0.00

*
; CASSIE READS THE PASSAGE:
; ========================
; Greg wants a better garden.
(describe (add agent #greg act (build action (build lex wants) object #bg)))

(m13! (act (m12 (action (m1 (lex wants))) (object b2))) (agent b1))

(m13!)

 CPU time : 0.00

*
;*greg is Greg
(describe (add object *greg proper-name (build lex Greg)))

(m17! (object b1) (proper-name (m16 (lex Greg))))

(m17!)

 CPU time : 0.00

*
;*bg is a better garden
(describe (add member *bg class (build lex "better garden")))
(m14! (class (m2 (lex better garden))) (member b2))

(m14!)

 CPU time : 0.00

*

;There are native plants.
;1. It's native.
(describe (add object #nplant property (build lex native)))

(m18! (object b3) (property (m7 (lex native))))
(m14! (class (m2 (lex better garden))) (member b2))

(m18! m14!)
```

```
 CPU time : 0.03

* ;2. It's plant.
(describe (add member *nplant class (build lex plants)))

(m31! (act (m30 (action (m1 (lex wants))) (object b3))) (agent b1))
(m23! (class (m3 (lex plants))) (member b3))

(m31! m23!)

 CPU time : 0.00

*
;There are xeric plants.
;1. It's xeric.
(describe (add object #xplant property (build lex xeric)))

(m32! (object b4) (property (m8 (lex xeric))))

(m32!)

 CPU time : 0.00

* ;2. It's plant.
(describe (add member *xplant class (build lex plants)))

(m35! (act (m34 (action (m1 (lex wants))) (object b4))) (agent b1))
(m33! (class (m3 (lex plants))) (member b4))

(m35! m33!)

 CPU time : 0.01

*
;Xeric plants is unknown.
(describe (add object (build lex xeric) property (build lex unknown)))

(m43! (min 0) (max 0)
 (arg (m40 (object b3) (property (m8 (lex xeric))))))
(m42! (min 0) (max 0)
 (arg (m38 (object b4) (property (m7 (lex native))))))
(m41! (min 1) (max 1) (arg (m40) (m18! (object b3) (property (m7)))))
(m39! (min 1) (max 1) (arg (m38) (m32! (object b4) (property (m8)))))
(m33! (class (m3 (lex plants))) (member b4))
(m28! (object (m8)) (property (m5 (lex unknown))))
(m26! (forall v9) (ant (p26 (class (m3)) (member v9)))
 (cq
  (p35 (min 1) (max 1)
   (arg (p34 (object v9) (property (m8)))
    (p33 (object v9) (property (m7)))))))
(m23! (class (m3)) (member b3))
(m10! (forall v11) (ant (p21 (object v11) (property (m8))))
 (cq (p23 (min 0) (max 0) (arg (p22 (object v11) (property (m7)))))))

(m43! m42! m41! m39! m33! m32! m28! m26! m23! m18! m10!)

 CPU time : 0.03
```

```
*
; Ask Cassie what "WORD" means:


End of /home/cendue/song4/cva/X2.demo demonstration.


 CPU time : 0.08
```

References:

1. Napieralski, Scott (Fall 2002), “Dictionary of CVA SNePS Case Frames (Fall 2002)”, [http://www.cse.buffalo.edu/~rapaport/CVA/CaseFrames/case-frames/]

2. Sturart, Shapiro C. and Rapaport， William J.(1998), “SNePS: An interactive approach”.

3. Rapaport, William J. and Kibby, Michael W. (2006), “Contextual vocabulary acquisition as computational philosophy and as philosophical computation”, *Journal of Experimental & Theoretical Artificial Intelligence, Vol. 19, No. 1, March 2007, 1-17 .*

4. *How many words are there in the English language?* http://www.oxforddictionaries.com/page/howmanywords