Contextual Vocabulary Acquisition-

Defining “Sedum” from Context

Bethany Griswold

CSE 499 RAP Contextual Vocabulary Acquisition Research Group

bethanyg@buffalo.edu

May 10, 2011

Abstract

CVA, or contextual vocabulary acquisition, is the process of using information within a context that a reader understands in order to understand the meaning of a word in the same context which is before analysis unknown to the reader. This report discusses the computational inference of a meaning for the word *sedum* using SNePS. The representation of knowledge and reasoning processes was taken from the mode of responses by four people questioned about the meaning of the word within a sentence. The meaning determined using SNePS and the process by which that meaning was acquired was only one of many possible based on the reasoning and prior knowledge of the questioned participants.

1 The CVA Project

CVA, or contextual vocabulary acquisition, is the process of attempting to infer a meaning for a word that a reader initially does not recognize within a cotext comprised of information that the reader does recognize. One may do this by drawing clues from the context of the word. Context

is, for the purposes of CVA, defined as the cotext of a word combined with the reader's background knowledge about things within that cotext. Cotext is a sentence or passage which includes the unknown word which may give hints as to the author's intended meaning for an unknown word (Rapaport, 2000). By understanding how to procedurally identify and use contextual clues, it should be possible to develop an effective method by which to teach people how to understand an unknown word from its context. This is one of the goals of the CVA project: to understand how we can help people improve their own understanding of words in a natural language by attempting to first teach computers in an explicit way.

But why do this? Is it not sufficient to look up the meaning of an unknown word in a dictionary? For those working on CVA, the answer is a resounding "no," for a variety of reasons. Often, a dictionary definition is unhelpful, or references other words that may be unknown to the reader. Other times, a dictionary reference is not available or the reader is simply too lazy or too involved in what they are doing to pause and find the definition in a dictionary. In any case, the process of CVA should be helpful not only in improving just people's CVA skills, but their reading comprehension ability overall. Far fewer words are explicitly taught through 12th grade than the number of words a student knows by the conclusion of high school (Adams, 2011). So where do students learn the words that they don't learn in school? It is reasonable to assume that it is from verbal or written context students encounter day-to-day. Vocabulary acquisition can be a passive process to some degree, but in teaching the process as an active, conscious activity, the rate of increase in the size of vocabulary could be vastly improved.

2 SNePS

SNePS is a Semantic Network Processing System. This means that it can represent ideas and the connections among them and is able to draw inferences by processing the information which it holds. This is structured with a series of nodes, which hold concepts of things, objects, or ideas, and arcs, which are labeled with indicators of the relations between the nodes that the arcs connect. There are several standard arc labels, which when used together represent case frames. SNePS can be used in CVA to build the representation of an unknown word's cotext and relevant background knowledge. In doing this, we create a cognitive agent, Cassie, which uses an algorithm to find one or more meanings for the unknown word, using possible sub-classes, super-classes, actions, and abilities as definitions. For this project, I used the snepsul variation of the system.

3 Unknown Word, Context, and Task

In order to help expand what we know about teaching a cognitive agent to infer the meaning of words, it is necessary to successfully teach a cognitive agent to infer possible meanings for a variety of words in many different contexts. In this project, I worked to help further this goal by representing the following passage from Nicholson Baker's A Box of Matches: "But now I'm up and little flames are growing like sedums from the cracks in today's log wall, and I still have a little while before I have to drive Phoebe to school." My task was to determine the definition of *sedums* (or the singular form, sedum) from context and reasonable background knowledge. The task was simplified somewhat in that the unknown word was a noun, and there is already an algorithm in place for retrieval of definition of nouns. I still needed to express in SNePS the cotext of the sentence, background knowledge that a reader might have that would help them in

inferring a meaning for *sedum*, and inference rules that would help draw conclusions based on the resulting context. Before representing these in SNePS, it was necessary for me to determine what the representation of the context would be to a human, what background knowledge a human reader would have, and what logical process a human would use to come to a conclusion about a possible meaning for the word.

4 Determining the Natural Human CVA Processes from Human Informants

Perhaps ironically, in order to teach computers how to do CVA so that we can teach people CVA, we are required to have untrained, yet naturally adept, people do CVA with our unknown word and context so that we know what to teach Cassie. I interviewed four human subjects, my "informants," on the passage. The subjects were all undergraduate students at UB between age 19 and 21. I asked them to speak the passage aloud, then "think aloud" as they went through the process of attempting to determine a meaning for the unknown word *sedum*. This "think aloud" process was intended to help me, the researcher, accurately determine what background knowledge and inference rules each informant used to find a definition for *sedum*. It also became surprisingly critical in helping me determine how a person might semantically represent a word differently from the grammatical representation likely intended by the author. However, despite the importance of and explicit instructions to "think aloud," it is often difficult to get the full procedure out of the initial description, as people will tend to skip steps in their process, and so "why" is a vital word in encouraging them to provide more information. Below are the transcripts of each informant's responses to the sentence, following the reading aloud of the sentence. Promptings for more information have been omitted, so only the useful information

provided by the informant remains.

A: “I think sedums are plants […] [because] the flames are like sedums, and the sentence says the flames are growing, so sedums must be something that grows, and the first thing that comes to mind that grows is plants.”

B: “The flames in the passage grow like sedums, so that means they grow in the way sedums grow, and conversely sedums probably grow in the way that flames grow, which in a log wall like the one referenced would be slowly branching outward from the point of origin of the flames, which vine plants also do. So I think it’s some kind of vine plant.”

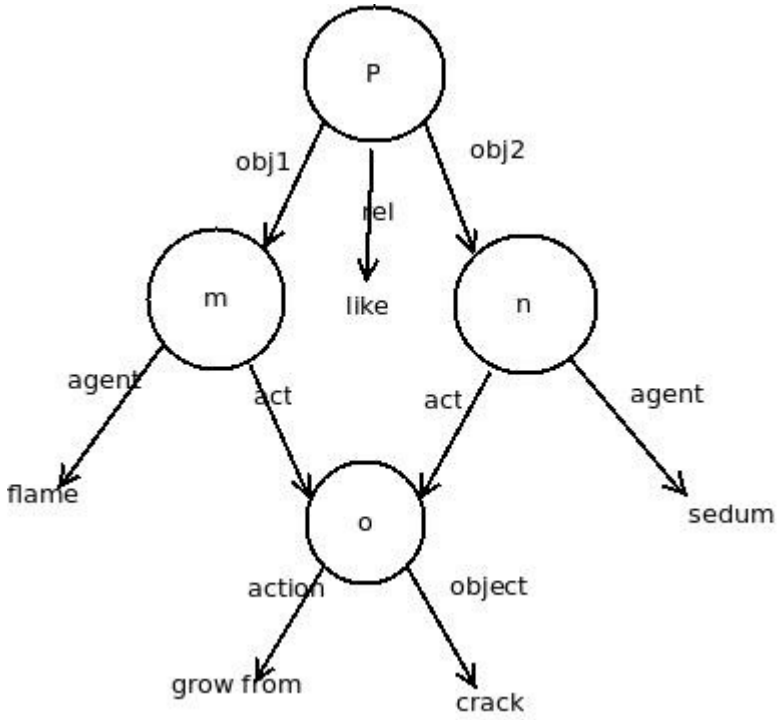
C: “The flames are little and growing like sedums from cracks, so I think that maybe a sedum is a type of small flame that grows from cracks.”

D: “Well, the flames are growing, and the flames are like sedums. So sedums must be something that grows […] [but] lots of things grow. Elephants grow, a city could grow in the sense that it could expand, but I don’t think it’s either of those things. I think it’s a plant, but I don’t know why. Ah, it must be because it grows from cracks, and plants are one of the few things that do that.”

5 Passage Representation based on Informant Information

Let us re-examine the cotext of *sedums*: “But now I'm up and little flames are growing like sedums from the cracks in today's log wall, and I still have a little while before I have to drive

Phoebe to school." The grammar of the sentence implies the following representation of the part of the sentence relating flames and sedums:



In words, the growth of the flames that we read about is like the growth of some things called sedums. However, out of my informants, A, B, C, and D, only one, informant B, represented the sentence this way in the verbal think aloud performed. All the other informants directly linked flames and sedums with "like," saying that sedums must be like flames. Therefore, I represented the relationship as, "forall x and forall y, if x is a flame and y is a sedum, then x is like y."

Code:

==========

```
;builds the part "all flames are like all sedums"
(describe
    ;forall x and all y
    (add forall($x $y)
    ;if x is a flame and y is a sedum
    &ant ((build member *x class (build lex "flame"))
        (build member *y class (build lex "sedum")) )
    ;then x is like y.
    cq (build object1 *x rel (build lex "is like") object2 *y))
)
```

For the remainder of the representation, I decided to use information from informant D, because D achieved a somewhat precise definition using reasoning techniques that seemed fairly common, unlike the apparently uncommon analysis of informant B, yet complex enough to come to a conclusion that was not overly simplistic, as that from Informant C. D also had a better reason for thinking sedums were plants than informant A did. Informant A said that the first thing that came to mind that grew was a plant. Informant D, while initially unsure of why he thought it was a plant, was able to reason further that plant were one of the few things that grew from cracks. Because this seemed like an important distinction, I approached this next in my code. I made a rule that forall x and forall y, if x is a flame and y is a crack, then x grows from y.

Code:

```
;builds the part "flames grow from cracks"
(describe
    ;forall x and all y
    (add forall($x $y)
    ;if x is a flame and y is a crack
    &ant ((build member *x class (build lex "flame"))
        (build member *y class (build lex "crack")))
    ;then x grows from y
    cq (build agent *x act (build action (build lex "grow from") object *y)))
)
```

I also specified that sedum was the unknown word in a member-class relationship.

Code:

```
;sedum is unknown
(describe (add member (build lex "sedum") class (build lex "unknown")))
```

Then to round out the representation I represented the crack, flame, and sedum talked about in the passage, as well as a plant that many informants thought the sedum might be.

Code:

```
;something is a sedum
(describe (add member #asedum class (build lex "sedum")))
;something is a flame
(describe (add member #aflame class (build lex "flame")))
;something is a plant
(describe (add member #aplant class (build lex "plant")))
;something is a crack
(describe (add member #acrack class (build lex "crack")))
```

6 Background Knowledge and Inferences from Informants

Once the passage was represented, I needed to build in the code that would duplicate the background knowledge and inference rules my informants used to determine their definitions for sedums. I started with the basic background knowledge from informants A, B, and D. They all implied that plants could grow, and D added that plants could grow from cracks, and aptly demonstrated why the distinction was critical. For simplicity, I decided I would make the rule that forall x, if x was a plant and y was a crack, then x grew from y, rather than defining a new ability case frame. But I was not sure how to represent the “from” in “grow from,” as there is not yet a standard SNePS case frame for prepositional phrases. I finally decided to just use the action/object case frame, with “grow from” as my action and “crack” as my location.

Code:

```
;builds the outside knowledge "all plants grow from all cracks"
(describe
;forall x and forall y
(assert forall($x $y)
&ant
    ;if x is a plant
    ((build member *x class (build lex "plant"))
    ;if y is a crack
    (build member *y class (build lex "crack")))
cq
    ;then x grows from y
    (build agent *x act (build action (build lex "grow from") object *y)))
)
```

Next I needed to get into the more complex inference rules for defining an unknown word. My informants had decided that since flames were like sedums, and they didn't know what sedums were, sedums must have the same actions as the flames had been described to have, e.g., growing from cracks. So my first rule was that if something was unknown and describe as like something else, and the thing it was like had a particular action, then the unknown thing would also have that action.

Code:

```
;builds the inference rule that if w is unknown, and x is
;like w, and x performs action y, then w performs action y
(describe
;forall w,x,y, and z
(assert forall ($w $x $y $z)
;if object1 is like object 2
&ant    ((build object1 *x rel (build lex "is like") object2 *w)
        ;and w is a member of the class z
        (build member *w class *z)
        ;and z is unknown
        (build member *z class (build lex "unknown"))
        ;and x does the act y
        (build agent *x act *y))
cq
        ;the unknown word performs act y
        (build agent  *w act *y))
)
```

Next, I had to determine the inference rule for why this led to my informants believing

that sedums were plants. They said it was because plants had the ability to grow from cracks, which I represented as all plants growing from all cracks. Therefore, they must have inferred that if something unknown (sedums) was able to do something (grow from cracks), and there was a known object (plants) which could do the same thing (grow from cracks), then the unknown (sedums) must be a subclass of the known thing (plants).

Code:

==========

```
;builds the inference rule that if x is unknown, and x performs action y,
;and z performs action y, then x is a subclass of z.
(describe
;forall x, y, z, q, and c
(assert forall ($x $y $z $q $c)
&ant(
    ;if x is part of the class q
    (build member *x class *q)
    ;if q is part of the class of unknowns
    (build member *q class (build lex "unknown"))
    ;and x performs y
    (build agent *x act *y)
    ;and z performs y
    (build agent *z act *y)
```

```
        ;and z is a member of some class c
        (build member *z class *c)
    )
cq
        ;then q is a subclass of c
        (build subclass *q superclass *c)
)
)
```

Because of the way I chose to represent things, I did not need to use any new case frames. Following inference, Cassie declared the following definition for sedum:

Definition of sedum:

Class Inclusions: plant, flame,

Possible Actions: grow from crack, nil

(nil is a stub from the noun definition inference algorithm)

A full transcript of code and run of the SNePS demo can be found at the end of this paper.

7 Next Steps – Refinement

A few alterations to the SNePS code would be useful for a more accurate representation of the knowledge my informants used to come to their conclusions.

My first step in improving the algorithm would be to include an “ability” case frame, for

the purposes of making more accurate background knowledge, as not all plants grow from all cracks. However, many plants have the ability to grow from many kinds of cracks, and I would specify as more particular background knowledge what kinds of plants, such as vines and moss, and perhaps small plants, were most likely to have the ability to grow from cracks.

The next important step is the inclusion of a property or class "contextual." Most of my informants used the properties of flames that were explicitly stated in the context to draw conclusions as to what sedums might be. They thought that sedums were like flames perhaps in their smallness or ability to grow from cracks, but no one stated that sedums were hot or had the ability to burn things. I should make an amendment to the rules so that an unknown only takes on contextually mentioned things that it is like, rather than all the properties of the thing it is like. I would add non-contextual properties of flames in the code to ensure that the rule was working.

Next, I would add the contextual piece of information that the flames in the passage were "little," which I glossed over before. Then, in order to take this into account in the meaning inference, I would create an inference rule that said if something is unknown, then it has all the contextual properties, not just the contextual actions, of the thing it was like. Eventually, I would redo the rule which said that if an unknown thing had the same action as a known thing, then the unknown thing was a subclass of the known thing. I would alter it so that the unknown would be a subclass of something that performed all the actions that the unknown performed and had all the properties that the unknown had.

Finally, it might be useful to create a rule that eliminates the eventual assertion of Cassie that sedums are a subclass of flames. My informant C and Cassie's belief that sedums are flames is in part due to a simplistic reasoning which fails to reason that if one thing is like another, then

the two things are also somehow different, and not either the same thing or in a subclass-superclass or member-class relation.

8 Long-Term Steps

One thing I feel is most important about my project is the realization of the need to address prepositional phrases in a standard and accurate way. While simply using the preposition as a part of my action worked for my purposes, I don't necessarily feel that it is the most appropriate representation, and there should be some discussions on other possible methods of preposition representation and whether there is a need to create a standard case frame for it.

9 Full SNePS code

```
;
==========================================================================
==
; FILENAME:     sedum-demo.txt
; 2_22_11:            03-21-11
; PROGRAMMER:   bethanyg

; 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 "bethanyg-SEDUM-demo.txt" :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)

;using the noun algorithm

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

; 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")
^(load "/projects/rapaport/CVA/mkb3.CVA/paths/paths")


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

;builds the outside knowledge "all plants grow from all cracks"
(describe
;forall x and forall y
(assert forall($x $y)
&ant
    ;if x is a plant
    ((build member *x class (build lex "plant"))
    ;if y is a crack
    (build member *y class (build lex "crack")))
cq
    ;then x grows from y
    (build agent *x act (build action (build lex "grow from") object *y)))
```

```
)

;builds the inference rule that if w is unknown, and x is
;like w, and x performs action y, then w performs action y
(describe
;forall w,x,y, and z
(assert forall ($w $x $y $z)
;if object1 is like object 2
&ant    ((build object1 *x rel (build lex "is like") object2 *w)
        ;and w is a member of the class z
        (build member *w class *z)
        ;and z is unknown
        (build member *z class (build lex "unknown"))
        ;and x does the act y
        (build agent *x act *y))
cq
        ;the unknown word performs act y
        (build agent  *w act *y))
)

;builds the inference rule that if x is unknown, and x performs action y,
;and z performs action y, then x is a subclass of z.
```

```
(describe
;forall x, y, z, q, and c
(assert forall ($x $y $z $q $c)
&ant(
        ;if x is part of the class q
        (build member *x class *q)
        ;if q is part of the class of unknowns
        (build member *q class (build lex "unknown"))
        ;and x performs y
        (build agent *x act *y)
        ;and z performs y
        (build agent *z act *y)
        ;and z is a member of some class c
        (build member *z class *c)
    )
cq
        ;then q is a subclass of c
        (build subclass *q superclass *c)
)


)
```

```
; Cassie READS THE PASSAGE:
; ==========================
; (put annotated SNePSUL code of the passage here)

;Passage: "But now I'm up, and little flames are growing like sedums from the
;cracks in today's log wall, and I still have a little while before I have to
;drive Phoebe to school."

;builds the part "all flames are like all sedums"
(describe
      ;forall x and all y
      (add forall($x $y)
      ;if x is a flame and y is a sedum
      &ant ((build member *x class (build lex "flame"))
            (build member *y class (build lex "sedum")) )
      ;then x is like y.
      cq (build object1 *x rel (build lex "is like") object2 *y))
)
;sedum is unknown
(describe (add member (build lex "sedum") class (build lex "unknown")))
```

```
;builds the part "flames grow from cracks"
(describe
	;forall x and all y
	(add forall($x $y)
	;if x is a flame and y is a crack
	&ant ((build member *x class (build lex "flame"))
		(build member *y class (build lex "crack")))
	;then x grows from y
	cq (build agent *x act (build action (build lex "grow from") object *y)))
)

;something is a sedum
(describe (add member #asedum class (build lex "sedum")))
;something is a flame
(describe (add member #aflame class (build lex "flame")))
;something is a plant
(describe (add member #aplant class (build lex "plant")))
;something is a crack
(describe (add member #acrack class (build lex "crack")))

;Have Cassie read in the passage including the word, which should
; trigger forward inference and process the definition.
```

;Then, ask what the word means.

^(defineNoun "sedum")

10 Demo Run

timberlake {~} > mlisp

International Allegro CL Enterprise Edition

8.2 [Linux (x86)] (Jul 9, 2010 16:05)

This development copy of Allegro CL is licensed to:

[4549] University at Buffalo

;; Optimization settings: safety 1, space 1, speed 1, debug 2.

;; For a complete description of all compiler switches given the

;; current optimization settings evaluate (explain-compiler-settings).

;;---

;; Current reader case mode: :case-sensitive-lower

cl-user(1):

cl-user(1): :ld /projects/snwiz/bin/sneps

; Loading /projects/snwiz/bin/sneps.lisp

;;; Installing streamc patch, version 2.

Loading system SNePS...10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

SNePS-2.7 [PL:2 2011/04/19 17:07:58] loaded.

Type `(sneps)' or `(snepslog)' to get started.

cl-user(2): (sneps)

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

Copyright (C) 1984--2010 by Research Foundation of

State University of New York. SNePS comes with ABSOLUTELY NO WARRANTY!

Type `(copyright)' for detailed copyright information.

Type `(demo)' for a list of example applications.

5/12/2011 11:39:49

* (demo "bethanyg-SEDUM-demo.txt")

File /home/eedue/bethanyg/bethanyg-SEDUM-demo.txt is now the source of input.

CPU time : 0.01

```
* ;
;=============================================================================
==
; FILENAME:sedum-demo.txt
; 2_22_11:                03-21-11
; PROGRAMMER:    bethanyg


; 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 "bethanyg-SEDUM-demo.txt" :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

*
;using the noun algorithm
^(
--> load "/projects/rapaport/CVA/STN2/defun_noun.cl")
; Loading /projects/rapaport/CVA/STN2/defun_noun.cl
t



 CPU time : 0.05

*
```

; 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)

#<The snip package>

CPU time : 0.00

```
*
 ;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)
#<The sneps package>



 CPU time : 0.00


*
; load all pre-defined relations:
```

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



 CPU time : 0.01

* ;(intext "/projects/rapaport/CVA/STN2/demos/rels")



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


 CPU time : 0.00
```

```
*

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

;builds the outside knowledge "all plants grow from all cracks"
(describe
;forall x and forall y
(assert forall($x $y)
&ant
        ;if x is a plant
        ((build member *x class (build lex "plant"))
        ;if y is a crack
        (build member *y class (build lex "crack")))
cq
        ;then x grows from y
        (build agent *x act (build action (build lex "grow from") object *y)))
)

(m4! (forall v2 v1)
 (&ant (p2 (class (m2 (lex crack))) (member v2))
  (p1 (class (m1 (lex plant))) (member v1)))
```

```
 (cq
  (p4 (act (p3 (action (m3 (lex grow from))) (object v2))) (agent v1))))

(m4!)

 CPU time : 0.00

*
;builds the inference rule that if w is unknown, and x is
;like w, and x performs action y, then w performs action y
(describe
;forall w,x,y, and z
(assert forall ($w $x $y $z)
;if object1 is like object 2
&ant    ((build object1 *x rel (build lex "is like") object2 *w)
          ;and w is a member of the class z
          (build member *w class *z)
          ;and z is unknown
          (build member *z class (build lex "unknown"))
          ;and x does the act y
          (build agent *x act *y))
cq
```

```
        ;the unknown word performs act y
        (build agent  *w act *y))
)

(m7! (forall v6 v5 v4 v3)
 (&ant (p8 (act v5) (agent v4))
  (p7 (class (m6 (lex unknown))) (member v6))
  (p6 (class v6) (member v3))
  (p5 (object1 v4) (object2 v3) (rel (m5 (lex is like)))))
 (cq (p9 (act v5) (agent v3))))

(m7!)

 CPU time : 0.00

*
;builds the inference rule that if x is unknown, and x performs action y,
;and z performs action y, then x is a subclass of z.

(describe
;forall x, y, z, q, and c
(assert forall ($x $y $z $q $c)
```

```
&ant(
        ;if x is part of the class q
        (build member *x class *q)
        ;if q is part of the class of unknowns
        (build member *q class (build lex "unknown"))
        ;and x performs y
        (build agent *x act *y)
        ;and z perfroms y
        (build agent *z act *y)
        ;and z is a member of some class c
        (build member *z class *c)
   )
cq
        ;then q is a subclass of c
        (build subclass *q superclass *c)
)

)

(m8! (forall v11 v10 v9 v8 v7)
 (&ant (p14 (class v11) (member v9)) (p13 (act v8) (agent v9))
  (p12 (act v8) (agent v7))
```

```
 (p11 (class (m6 (lex unknown))) (member v10))
 (p10 (class v10) (member v7)))
(cq (p15 (subclass v10) (superclass v11))))

(m8!)

 CPU time : 0.00

*
; Cassie READS THE PASSAGE:
; =========================
; (put annotated SNePSUL code of the passage here)

;Passage: "But now I'm up, and little flames are growing like sedums from the
;cracks in today's log wall, and I still have a little while before I have to
;drive Phoebe to school."

;builds the part "all flames are like all sedums"
(describe
        ;forall x and all y
        (add forall($x $y)
        ;if x is a flame and y is a sedum
```

```
        &ant ((build member *x class (build lex "flame"))
                (build member *y class (build lex "sedum")) )
        ;then x is like y.
        cq (build object1 *x rel (build lex "is like") object2 *y))
)

(m11! (forall v13 v12)
 (&ant (p17 (class (m10 (lex sedum))) (member v13))
  (p16 (class (m9 (lex flame))) (member v12)))
 (cq (p18 (object1 v12) (object2 v13) (rel (m5 (lex is like))))))

(m11!)

 CPU time : 0.00

* ;sedum is unknown
(describe (add member (build lex "sedum") class (build lex "unknown")))

(m12! (class (m6 (lex unknown))) (member (m10 (lex sedum))))

(m12!)
```

```
 CPU time : 0.04


*

;builds the part "flames grow from cracks"

(describe

        ;forall x and all y

        (add forall($x $y)

        ;if x is a flame and y is a crack

        &ant ((build member *x class (build lex "flame"))

                (build member *y class (build lex "crack")))

        ;then x grows from y

        cq (build agent *x act (build action (build lex "grow from") object *y)))

)


(m21! (forall v15 v14)

 (&ant (p43 (class (m2 (lex crack))) (member v15))

  (p42 (class (m9 (lex flame))) (member v14)))

 (cq

  (p45 (act (p44 (action (m3 (lex grow from))) (object v15)))

   (agent v14))))


(m21!)
```

CPU time : 0.00

*

;something is a sedum

(describe (add member #asedum class (build lex "sedum")))

(m22! (class (m10 (lex sedum))) (member b1))

(m22!)

CPU time : 0.01

* ;something is a flame

(describe (add member #aflame class (build lex "flame")))

(m32! (object1 b2) (object2 b1) (rel (m5 (lex is like))))

(m31! (class (m9 (lex flame))) (member b2))

(m32! m31!)

CPU time : 0.01

* ;something is a plant

(describe (add member #aplant class (build lex "plant")))

(m33! (class (m1 (lex plant))) (member b3))

(m33!)

CPU time : 0.00

* ;something is a crack

(describe (add member #acrack class (build lex "crack")))

(m40! (subclass (m10 (lex sedum))) (superclass (m1 (lex plant))))

(m39! (subclass (m10)) (superclass (m9 (lex flame))))

(m38! (act (m35 (action (m3 (lex grow from))) (object b4))) (agent b1))

(m37! (act (m35)) (agent b2))

(m36! (act (m35)) (agent b3))

(m34! (class (m2 (lex crack))) (member b4))

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

CPU time : 0.02

*

;Have Cassie read in the passage including the word, which should

; trigger forward inference and process the definition.

;Then, ask what the word means.

^(

--> defineNoun "sedum")

Definition of sedum:

Class Inclusions: plant, flame,

Possible Actions: grow from crack,

nil

CPU time : 0.02

*

End of /home/eedue/bethanyg/bethanyg-SEDUM-demo.txt demonstration.

CPU time: 0.16

11 References

Adams, Marilyn Jager (2010-2011), "Advancing Our Students' Language and Literacy: The Challenge of Complex Texts", *American Educator* 34(4) (Winter): 3–11, 53.

Rapaport, William J., & Ehrlich, Karen (2000), "A Computational Theory of Vocabulary Acquisition" in Lucja M. Iwanska & Stuart C. Shapiro (eds.), *Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language* (Menlo Park, CA/Cambridge, MA: AAAI Press/MIT Press): 347-375.