Contextual Vocabulary Acquisition:

Computing the meaning of "drenched"

Amol Kothari

17th December 2004

CSE740

Abstract:

The goal of the project is to model the cognitive processes that occur in a human mind, while computing the meaning of an unknown word from the context in which it is used. My aim in specific was to build a representation of such processes, that occur while determining the meaning of the word "drenched" from the context "Suda got drenched during the rainstorm because she had left her umbrella at her office." This representation was built in the Semantic Network Processing System (SNePS). After the representation was complete, our cognitive agent Cassie was asked to come up with a definition for the unknown word "drenched" given the representation of the context. To aid Cassie in this process, she was also provided with a representation of the background knowledge, which a human would require in deducing the meaning of "drenched". Cassie came up with exactly the same definition for "drenched" that humans came up with.

Introduction to Contextual Vocabulary Acquisition (CVA):

Many times during the course of everyday reading, we encounter a word whose meaning is not known to us. In order to comprehend the meaning of the unknown word and hence theSNEPS Project Report passage, we might do either of theses:

1. Ignore the word and proceed with the reading process. If the word is never encountered again then there is no problem, but inevitably the word will be encountered again in another context, where its meaning is even more crucial in understanding the passage.
2. Consult a dictionary and determine the meaning of the unknown word. Though this method seems straight forward it has certain disadvantages. Firstly a dictionary might not always be at hand to help in determining the meaning of the unknown word. It leads to an interruption in the reading process. Also a dictionary might not always prove helpful in understanding the meaning of the word. They are often misleading.
3. Do Contextual Vocabulary Acquisition, which is what is done by most people either consciously or subconsciously. The advantages that this method has is that there is no need for a dictionary to be at hand, and hence no interruption in the reading process. Also further encounters with the word lead to refinement of the definition of the unknown word rather than leading to more confusion.

Contextual Vocabulary Acquisition is thus the process of figuring out the meaning of an unknown word by using clues provided by the context in which the word appears. The meaning obtained for the unknown word, using this method might not be exactly like the dictionary like definition of the word, but what we need is just a definition that will enable us to read further on, after completely understanding the context. This preliminary definition of the word might not be complete, in the sense that it might not have all the components of the meaning of the word, but only those that are directly or indirectly indicated by the context. When the word is encountered in another context, we might learn more about the meaning of the unknown word, hence the definition is refined after each subsequent exposure to the word.

SNePS:

SNePS is an acronym for Semantic Network Processing System. It is a system that takes as input SNePS User Language commands and constructs a corresponding semantic network. SNePS is used to build a semantic network representation of the passage containing the unknown word, as well as a semantic representation of the background knowledge that is required by a human reader, to correctly determine the meaning of the unknown word.

Cassie:

Cassie is the name of the cognitive agent which deduces the meaning of the unknown word, using the semantic network representation developed in SNePS. Cassie consists of the SNePS knowledge representation and reasoning system. To determine the meaning of an unknown word, it first locates the word in the Semantic Network and then looks for possible associations that the word might have with other concepts in the network. After an exhaustive search of the network it comes up with a possible structure for the word. This structure consists of possible properties like class inclusions, synonyms, actions, ownership and properties.

My role in the project:

My role in the project is to represent the following passage in SNePS along with the background knowledge that a human might need to infer the meaning of the unknown word "drenched" from the context.

"Suda got drenched during the rainstorm because she had left her umbrella at her office."

In the above sentence we assume that the meaning of the word "drenched" is unknown. The source of the passage is:

http://learning.ricr.ac.th/Efcass/chapter3_5.htm

The website above advocates the learning from context method. In fact the passage provided above was one of the many examples that were given to emphasize the fact that meaning of unknown words can be inferred from context. It was thus interesting to determine whether the meaning of the word "drenched" could indeed be determined from the passage by a human and by Cassie. If it could, then what was the thought process that went behind doing so.

Verbal Protocols:

To follow the thought process that actually goes behind determining the meaning of the word "drenched" from the given context, I carried out Think Aloud Protocols on a few subjects. The subjects were provided with the same context mentioned earlier, but with the word "drenched" replaced by the word "gressed", and asked to determine the meaning of "gressed". The word "drenched" was replaced by the word "gressed" due to the fact that most people having average or even below average verbal ability would know the meaning of the word "drenched" before hand.

In all 3 subjects were chosen for carrying out the verbal protocols. The results obtained are as follows:

Subject 1:
When asked to compute the meaning of the word "gressed" from the context the subject immediately responded with the meaning "wet". When pressed further the subject said that since "Suda had forgotten her umbrella at her office", she did not have an umbrella with her. Also at the same time there was a rainstorm. Since she did not have an umbrella with her and at the same time it started raining, Suda got wet. But Suda also got "gressed" at the same time. Therefore the words "gressed" and "wet" meant the same thing. Hence the final conclusion made by subject 1 was that "gressed" means "wet".

Subject 2:
Subject 2 responded in the same manner as Subject 1. But in addition to the meaning "wet" the subjects also came up with the meaning of "gressed" as "upset". The inference here was that if a person gets wet then the person also gets upset. Therefore the final conclusion made by subject 2 was that the word "gressed" could mean either of "wet" and "upset".

Subject 3:
Subject 3 responded with only one meaning for the word "gressed", that of being "upset". The thought process in getting to this meaning was along the same lines as that made by subject 2.

After carrying out the verbal protocols, there were two possible meanings of the word "drenched":

1. **"wet"**
2. **"upset"**

My task in this project was to model the thought process that went on in the minds of the subjects for the verbal protocol and to represent this thought process as a Semantic network using Semantic Network Processing System (SNePS). The semantic network would then be examined by Cassie, who would try and come up with a possible definition for "drenched".

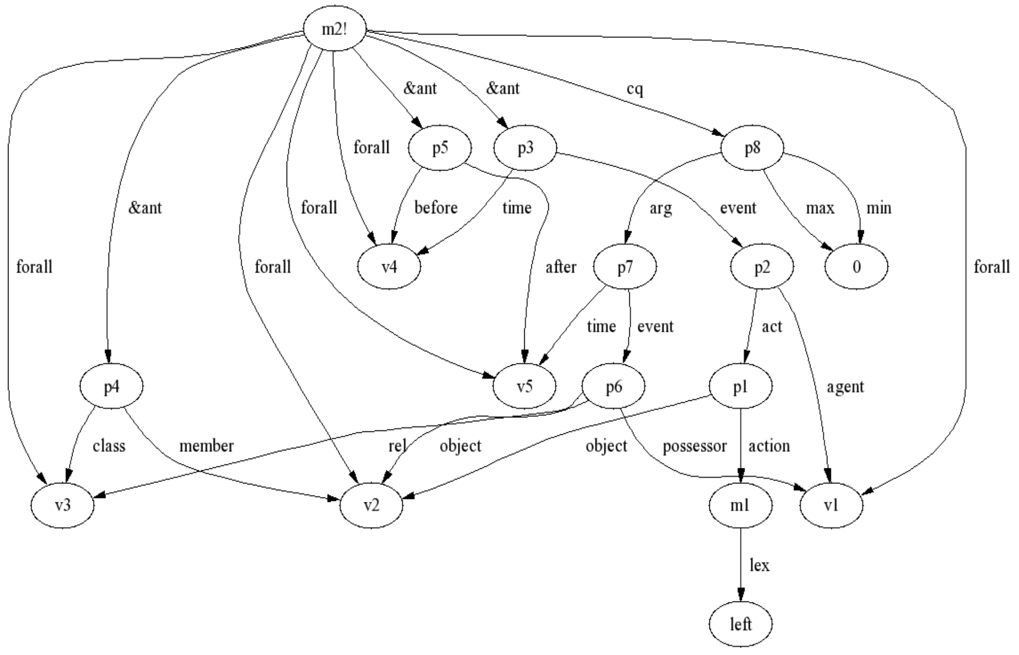
Me final task was to compare the meanings inferred by Cassie for the given representation with the meanings inferred by the human protocols.

Representation of Background Knowledge:

The background knowledge consists of the following rules, which were either consciously or intuitively used by the verbal protocols in computing the meaning of “gressed”. A list of all new case frames that have been used in the representation are provided in Appendix

1. **“If a person y leaves an object x then the person y does not have object x.”**

   Need for the rule:
   The rule is needed to assert the fact that if a person y leaves some object x at a time t1, then at some time t2 after the time t1, the person y does not have the object x. This rule represents the concept that humans have of “leaving” an object. It is represented as follows:



The following nodes in the network correspond to:

v1 -> person y

v2 -> object x

p2 -> the event of leaving the object x performed by person y

v4 -> the time at which p2 occurs

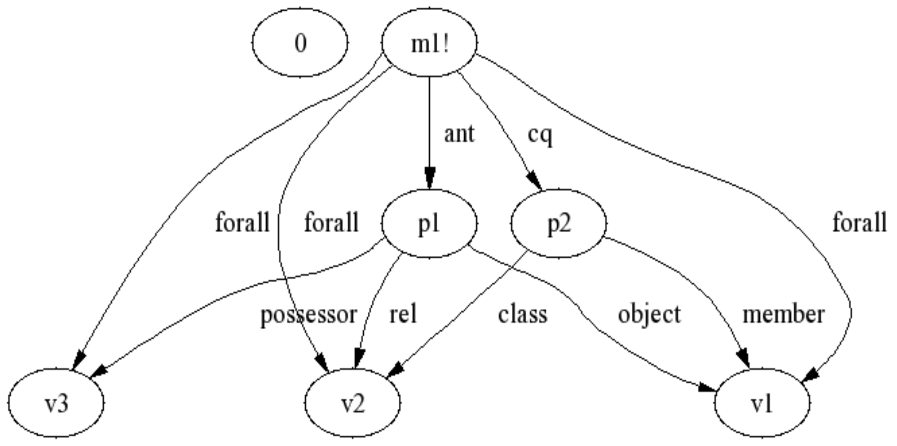
p7 -> the event of person y not having object x

v5 -> the time at which p7 occurs

The antecedent for the above rule states that the event p2 occurs at a time t1 (v4). To represent the above I have used an event-time case frame [Appendix A-4]. The event p2 is represented by the agent-act-action-object case frame. The consequent for the above rule is the event p7 that occurs at time t2 (v5). To represent the event p2 I have used the negation of the object-rel-possessor case frame.

2. **“If x is y’s z then x is a member if the class z”.**

   Need for the rule:
   This rule is needed to represent the concept that humans have of possessing an object. For e.g.:
   Given the sentence “Fido is John’s dog” most humans sub consciously infer that Fido is a member of the class of dogs.
   The rule is represented as follows:



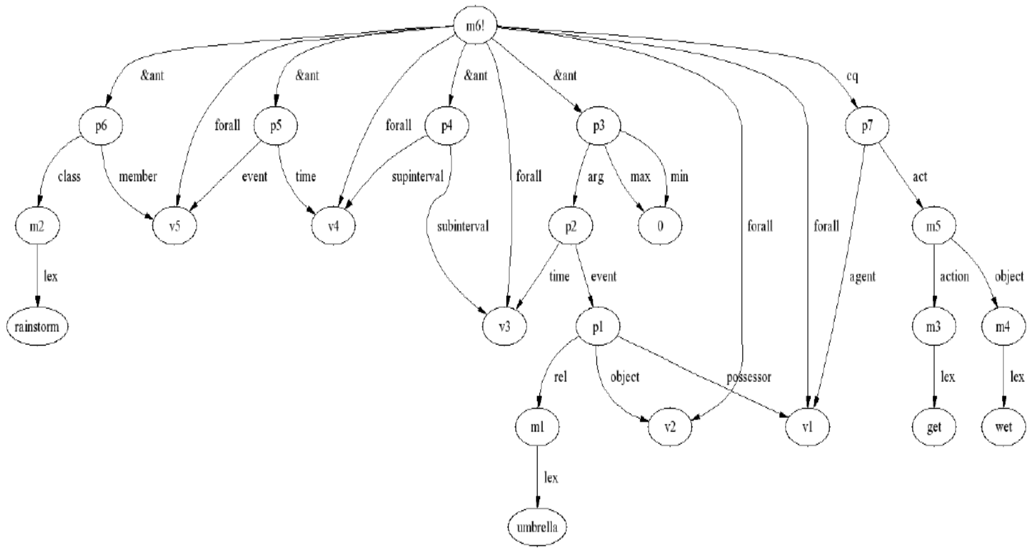
   The following nodes in the network correspond to:
   V1 -> object x
   V2 -> relationship z
   V3 -> possessor y

   The antecedent for the rule has an object-rel-possessor case frame that represents the fact that x is y’s z.
   The consequent uses the member-class case frame to represent the fact that x is a member of the class z.

3. **"If x does not have an umbrella at a time t, and there is a rainstorm at that time t, then x gets wet."**

   Need for the rule:

   The basis for this rule are the Verbal Protocols that were carried out as explained earlier. The rule is represented using the following network:



   The following nodes in the network correspond to:

   V5 -> A member of the class of rainstorms

   V4 -> the time at which V5 occurs

   P2 -> the event that x does not have an umbrella

   V3 -> the time at which event p2 occurs

   V2 -> A member of the class of umbrellas

   V1 -> the node representing the person x

   P7 -> the proposition that person x gets wet

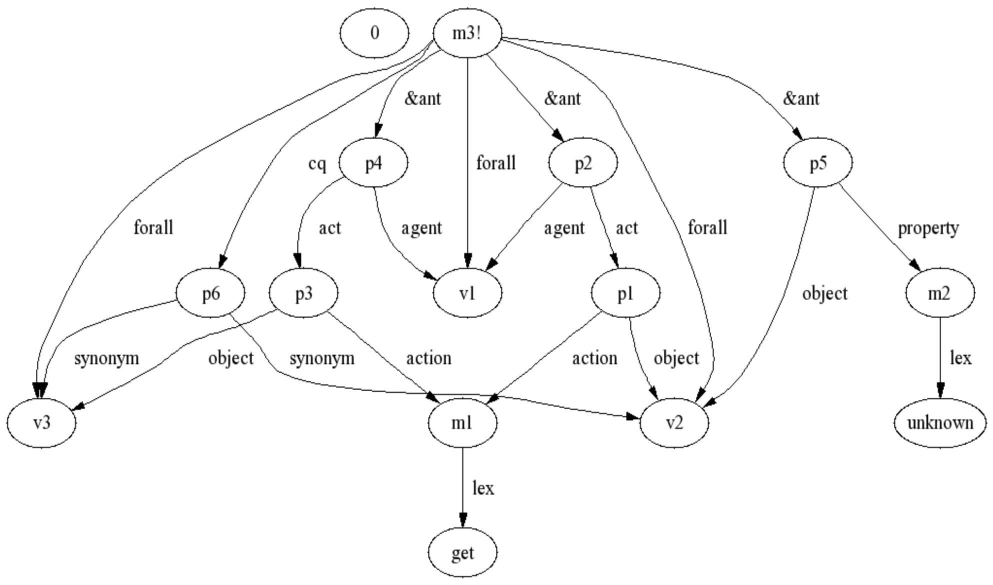
There are 4 antecedents for the rule. They are as follows:

a. The event p2 occurs. This is represented in a similar way as in Rule 2.
b. The rainstorm V5 is a member of the class of rainstorms.
c. The event of the rainstorm V5 occurs at a time V4.
d. The time V3 is a sub-interval of the time V4. This representation makes use of the subinterval – supinterval case frame [Appendix A-3]

The consequent is the proposition P7, represented by the agent-act-action-object case frame.

4. **"If a person x gets y'ed at some time t, and that person also gets z'ed at the same time t, and z is unknown, then z and y are synonyms."**

   Need for the rule:
   The rule was again designed on the basis of the Verbal Protocols , wherein the subjects had the general notion that if is some person performs an act y at a time t, and the person also performs an act z at the same time t, then y and z are the same, or they are synonyms. The rule is represented by the following network:

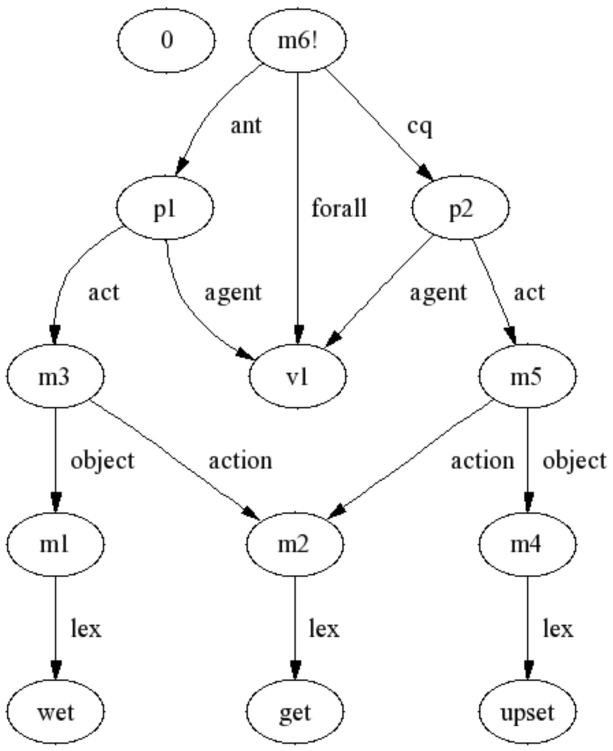


The following nodes in the network correspond to:
V1 -> the person x
V2 -> z
V3 -> y
P4 -> the proposition that the person x gets y'ed
P2 -> the proposition that the person x gets z'ed

The antecedent for the rule sis the proposition P4
The consequent for the rule sis the proposition P2

5. "If x gets wet then x gets upset"

This rule was a direct consequence of the inference followed by Verbal Protocol Subjects 2 and 3. According to the subjects if a person gets wet then the person as a result of being wet gets upset. The rule is represented by the following network:



V1 -> the node representing the person x

P1 -> the proposition that the agent v1 gets wet

P2 -> the proposition that the agent v1 gets upset

The antecedent for the rule is the proposition P1

The consequent for the rule is the proposition P2

Representation of the passage:

The given passage is

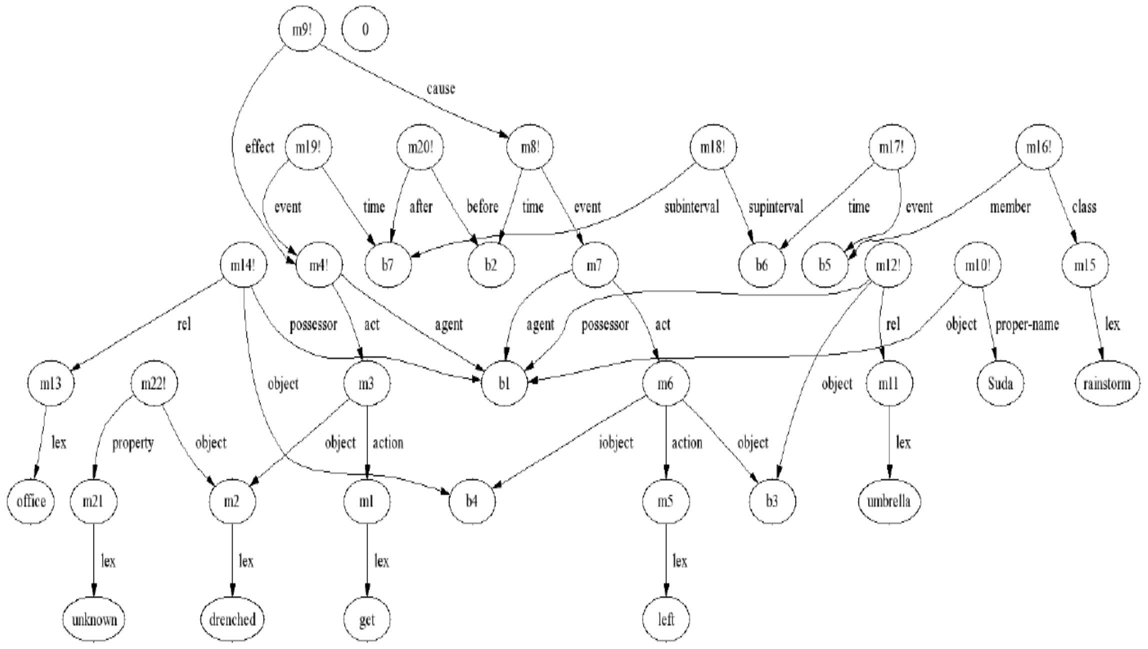
“Suda got drenched during the rainstorm because she had left her umbrella at her office.”

The passage can be broken down into two parts the cause and the effect.

The cause is the proposition “Suda left her umbrella at her office.”

The effect is the proposition “Suda got drenched during the rainstorm.”

The sentence can be represented by the following network:



M4 -> The event that Suda gets “drenched”

B7 -> The time at which M4 occurs

M12 -> The node asserting that B3 is Suda’s umbrella

M14 -> The node asserting that B4 is Suda’s office

B5 -> A member of the class of rainstorms

B6 -> The time at which B5 occurs

B2 -> The time at which event M7 occurs

B1 -> The concept representing “Suda”

B3 -> Suda’s umbrella

B4 -> Suda’s office

M7 -> The event that Suda leaves her umbrella at her office

The cause is the event M7 occurring at B2.
To represent the propositions M12 and M14 we use the object- rel - possessor case frames.
The effect is the event M4.

Additional facts that are represented:

- M18 -> The time b7 at which the event occurs, is a subinterval of a time b6 (subinterval – supinterval case frame [Appendix A-3])
- M20 -> The time B2 occurs before time B7 (before-after case frame [Appendix A-2])
- M22 -> The word “drenched” has the property of being unknown
- M9 -> Finally the cause-effect case frame [Appendix A-1] connects the proposition represented by cause and the proposition represented by the effect.

Result provided by Cassie:

After the passage and the background were represented, Cassie was asked to define "drenched" using the verb algorithm, but the verb algorithm provided no output. Then Cassie was asked to define "drenched" using the noun algorithm.

Cassie then came up with the word "wet" as a possible synonym for the word "drenched". But it did not come up with "upset" as a possible synonym for "drenched". Then Cassie was explicitly asked whether "drenched" and "upset" were synonyms. She replied positively. Hence due to some glitch in the inference engine, Cassie did not initially provide both "upset" and "wet" as synonyms, but only the word "upset". In the end though, we got the 2 possible synonyms for "drenched" as

1. "wet"
2. "upset"

Future Work:

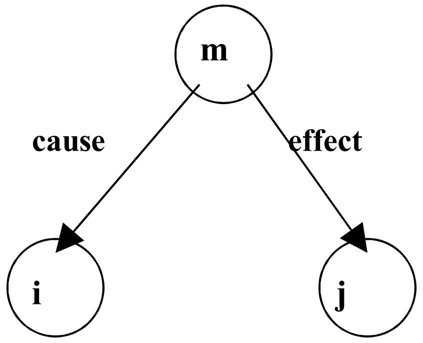
In the long run one of the most important things to do is to improve the verb algorithm, so that it can pick up the meaning of verbs like "drenched". The current verb algorithm was unable to do so, but the noun algorithm was successful in defining "drenched" as having possible synonyms "wet" and "upset". Hence the verb algorithm should be modified so that in addition to analyzing the structure of the sentence for defining the verb, it should also pick up possible synonyms for the word.

Another important thing to look into is the working of the SNePS inference package. The inference package was unable to deduce the fact that "drenched" and "upset" were synonyms implicitly. It was only after explicitly asking it whether "drenched" and "upset" were synonyms , does the inference package trigger rule 5 , and deduce the result. Hence such glitches have to be eliminated from SNePS.

Appendix A:

New Case Frames used:

1. Cause - Effect Case Frame:

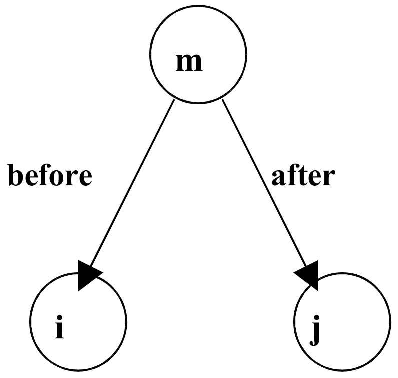


Semantics:

[[m]] is the proposition that proposition [[i]] causes proposition [[j]]

e.g.: (assert cause #earthquake effect #damage)

2. Before - After Case Frame:

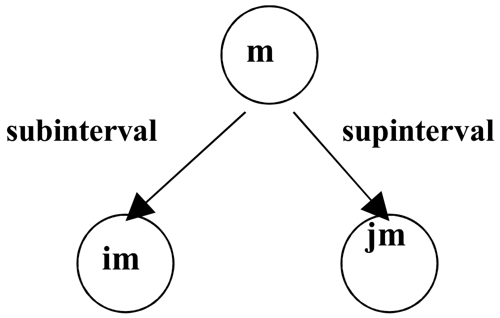


Semantics:

[[m]] is the proposition that time [[i]] occurs before [[j]]

e.g.: (assert before #t1 after #t2)

3. Subinterval - Supinterval Case Frame

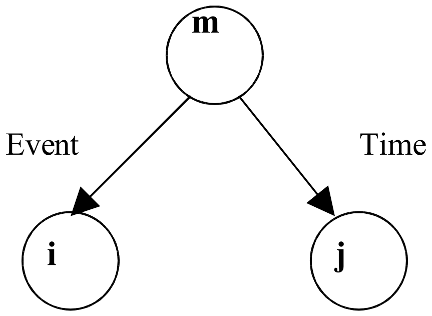


Semantics:

[[m]] is the proposition that time [[i]] is a subinterval of time [[j]]

e.g.: (assert subinterval #t1 supinterval #t2)

4. Event - Time Case Frame



Semantics:

[[m]] is the proposition that event [[i]] occurs at time [[j]]

e.g.: (assert event #t1 time #t2)

Appendix B :

Annotated Demo :

```
* ; =======================================================================
; FILENAME: drenched.demo
; DATE: 7th November 2004
; PROGRAMMER:  Amol Kothari

; this template version:   template.demo.2003.11.17.txt
; Lines beginning with a semi-colon are comments.
; Lines beginning with "^" are Lisp commands.
; All other lines are SNePS commands.
; To use this file: run SNePS; at the SNePS prompt (*), type:
;(demo "drenched.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)

; Load the appropriate definition algorithm:
 ^(
--> load "/projects/rapaport/CVA/STN2/defun_noun.cl")
; Loading /projects/rapaport/CVA/STN2/defun_noun.cl
t


 CPU time : 0.19

*

; Clear the SNePS network:
(resetnet t)
```

```
Net reset

 CPU time : 0.01

*
; 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

*
CPU time : 0.00

*
; 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 THE NEW RELATIONS
;=========================================================================


(define  subinterval supinterval iobject)

(subinterval supinterval iobject)

 CPU time : 0.00

*


;==========================================================================
; ENTER THE BACKGROUND KNOWLEDGE
;==========================================================================


;--------------------------------------------------------------------------------------------------
;Rule 1: forall x,y &z :
;
;                  " if x leaves y then x does not have y "
;--------------------------------------------------------------------------------------------------
(assert forall $X forall $Y forall $Z forall $t1 forall $t2
&ant (build time *t1 event(build act  ( build object *Y action (build lex "left") ) agent *X))
&ant (build member *Y class *Z)
&ant (build before *t1 after *t2)
 cq  (build min 0 max 0 arg(build time *t2 event (build possessor *X object *Y rel *Z) ) ))

(m2!)

 CPU time : 0.00

*
```

```
;---------------------------------------------------------------------------------------------------
;Rule 2: forall x,y,t1,t2 &rainstorm :
;
;   " if x does not have an umbrella & there is a rainstorm then x get wet"
;---------------------------------------------------------------------------------------------------

(assert forall $X forall $Y forall $t1 forall $t2 forall $rainstorm
&ant (build min 0 max 0 arg (build time *t1 event (build object *Y rel (build lex "umbrella") possessor *X)))
&ant ( build subinterval *t1 supinterval *t2 )
&ant ( build time *t2 event *rainstorm )
&ant ( build member *rainstorm class (build lex "rainstorm"))
cq   ( build agent *X act (build action (build lex "get") object (build lex "wet"))) )

(m8!)

 CPU time : 0.01

*



;-------------------------------------------------------------------------------------------------
;Rule 3: forall x :
;
;  " if x gets wet then x gets upset"
;-------------------------------------------------------------------------------------------------

(assert forall $X
&ant ( build act (build object (build lex "wet")  action (build lex "get")= get_lex)  agent *X )
cq   ( build act (build object (build lex "upset")action  *get_lex              )  agent *X ))

(m11!)

 CPU time : 0.00

*
```

```
;---------------------------------------------------------------------------------------------------
;Rule 4: forall x,y &z :
;
;  " if x is z's y (obj rel pos) then x is a member of class y "
;---------------------------------------------------------------------------------------------------

(assert forall $X forall $Y forall $Z
                                                                                              &ant
( build object *X rel *Y possessor *Z)
                                                                                              cq
( build member *X class *Y) )

(m12!)

 CPU time : 0.00

*

;---------------------------------------------------------------------------------------------------
;Rule 5: forall x,y &z :
;
;  " if x gets y at time t & x gets z at the same time t,& z is unknown, then z means y "
;---------------------------------------------------------------------------------------------------

(assert forall $X forall $Y forall $Z
&ant (build act (build object *Y action (build lex "get")=get_lex1) agent *X )
&ant (build act (build object *Z action *get_lex1            ) agent *X )
&ant (build object *Y property (build lex "unknown"))
cq   (build synonym *Y synonym *Z) )

(m14!)

 CPU time : 0.01

*
```

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

;===========================================================================;PA
SSAGE
;
;" Suda got drenched during the rainstorm becuase she had left her umbrella at her office. "
;===========================================================================




;-----------------------------------------------------------------------------------------------
; The main passage structure
;
;  The Cause and Effect relationship
;         Cause is  "Suda left her umbrella at her office"
;  Effect is " Suda got drenched"
;-----------------------------------------------------------------------------------------------

( add
effect (assert agent #Suda act ( build action (build lex "get") object (build lex "drenched"))  )=
Suda_got_drenched

cause (assert time #t4 event (build agent *Suda act (build action (build lex "left") object #Suda_umbrella
iobject #Suda_office ) )) )

(m21!)

 CPU time : 0.00

*
```

```
;-----------------------------------------------------------------------------------------------
;  The Base node Suda has the proper-name Suda
;-----------------------------------------------------------------------------------------------

(add object *Suda proper-name "Suda")

(m22!)

 CPU time : 0.01

*

;-----------------------------------------------------------------------------------------------
;  The object Suda_umbrella is Suda's umbrella
;-----------------------------------------------------------------------------------------------

(add object *Suda_umbrella rel (build lex "umbrella") possessor *Suda)

Since
((m12! (forall v14 v13 v12) (ant (p18 (object v12) (possessor v14) (rel v13)))
  (cq (p19 (class v13) (member v12)))))
and
((p18 (object (v12 <-- b3)) (possessor (v14 <-- b1)) (rel (v13 <-- m3))))
I infer
((p19 (class (v13 <-- m3)) (member (v12 <-- b3))))

I wonder if
((p5 (after v5) (before v4)))
holds within the BS defined by context default-defaultct

I wonder if
((p3
  (event
   (p2 (act (p1 (action (m1 (lex (left)))) (object (v2 <-- b3)))) (agent v1)))
  (time v4)))
holds within the BS defined by context default-defaultct

I know
((m20!
  (event
   (m19 (act (m18 (action (m1 (lex (left)))) (iobject (b4)) (object (b3))))
    (agent (b1))))
  (time (b2))))

(m27! m24! m23!)

 CPU time : 0.04

*




;-----------------------------------------------------------------------------------------------
;  The object Suda_office is Suda's office
;-----------------------------------------------------------------------------------------------

(add object *Suda_office rel (build lex "office") possessor *Suda)

Since
((m12! (forall v14 v13 v12) (ant (p18 (object v12) (possessor v14) (rel v13)))
  (cq (p19 (class v13) (member v12)))))
and
((p18 (object (v12 <-- b4)) (possessor (v14 <-- b1)) (rel (v13 <-- m28))))
I infer
((p19 (class (v13 <-- m28)) (member (v12 <-- b4))))

I wonder if
((p3
  (event
   (p2 (act (p1 (action (m1 (lex (left)))) (object (v2 <-- b4)))) (agent v1)))
  (time v4)))
holds within the BS defined by context default-defaultct

(m30! m29!)

 CPU time : 0.10

*
```

```
;-----------------------------------------------------------------------------------------------
; Rainstorm1 is a member of the class of rainstorms
;-----------------------------------------------------------------------------------------------

(add member #rainstorm1 class (build lex "rainstorm") )

I wonder if
((p3
  (event
   (p2 (act (p1 (action (m1 (lex (left)))) (object (v2 <-- b5)))) (agent v1)))
  (time v4)))
holds within the BS defined by context default-defaultct

I wonder if
((p13 (event (v10 <-- b5)) (time v9)))
holds within the BS defined by context default-defaultct

I wonder if
((p12 (subinterval v8) (supinterval v9)))
holds within the BS defined by context default-defaultct

I wonder if
((p11 (min 0) (max 0)
  (arg
   (p10 (event (p9 (object v7) (possessor v6) (rel (m3 (lex (umbrella))))))
    (time v8)))))
holds within the BS defined by context default-defaultct

I wonder if
((p10 (event (p9 (object v7) (possessor v6) (rel (m3 (lex (umbrella))))))
  (time v8)))
holds within the BS defined by context default-defaultct

I wonder if
((p4 (class (v3 <-- m3)) (member v2)))
holds within the BS defined by context default-defaultct
```

```
I wonder if
((p3 (event (p2 (act (p1 (action (m1 (lex (left)))) (object v2))) (agent v1)))
  (time v4)))
holds within the BS defined by context default-defaultct

I know
((m24! (class (m3 (lex (umbrella)))) (member (b3))))

I know
((m20!
  (event
   (m19 (act (m18 (action (m1 (lex (left)))) (iobject (b4)) (object (b3))))
    (agent (b1))))
  (time (b2))))

I know
((m27!
  (event
   (m26 (act (m25 (action (m1 (lex (left)))) (object (b3)))) (agent (b1))))
  (time (b2))))

I wonder if
((p18 (object v12) (possessor v14) (rel (v13 <-- m3))))
holds within the BS defined by context default-defaultct

Since
((m12! (forall v14 v13 v12) (ant (p18 (object v12) (possessor v14) (rel v13)))
  (cq (p19 (class v13) (member v12)))))
and
((p18 (object (v12 <-- b3)) (possessor (v14 <-- b1)) (rel (v13 <-- m3))))
I infer
((p19 (class (v13 <-- m3)) (member (v12 <-- b3))))

Since
((m12! (forall v14 v13 v12) (ant (p18 (object v12) (possessor v14) (rel v13)))
  (cq (p19 (class v13) (member v12)))))
and
((p18 (object (v12 <-- b3)) (possessor (v14 <-- b1)) (rel (v13 <-- m3))))
I infer
```

```
((p19 (class (v13 <-- m3)) (member (v12 <-- b3))))

Since
((m12! (forall v14 v13 v12) (ant (p18 (object v12) (possessor v14) (rel v13)))
  (cq (p19 (class v13) (member v12)))))
and
((p18 (object (v12 <-- b3)) (possessor (v14 <-- b1)) (rel (v13 <-- m3))))
I infer
((p19 (class (v13 <-- m3)) (member (v12 <-- b3))))

I know
((m23! (object (b3)) (possessor (b1)) (rel (m3 (lex (umbrella))))))

(m32! m23!)

 CPU time : 0.08

*



;-----------------------------------------------------------------------------------------------
; The event rainstorm1 occurs at time t1
;-----------------------------------------------------------------------------------------------

(add time #t1 event *rainstorm1)

(m34!)

 CPU time : 0.01

*

;(show *nodes)
;------------------------------------------------------------------------------------------------
; t1 is a superinterval of the time interval t2
;------------------------------------------------------------------------------------------------

(add supinterval *t1 subinterval #t2)
```

```
(m35!)

 CPU time : 0.01

*



;-----------------------------------------------------------------------------------------------
; The event of Suda getting drenched occurs at time t2
;-----------------------------------------------------------------------------------------------

(add time *t2 event *Suda_got_drenched)

I wonder if
((p14 (class (m4 (lex (rainstorm)))) (member (v10 <-- m17!))))
holds within the BS defined by context default-defaultct

I wonder if
((p12 (subinterval v8) (supinterval (v9 <-- b7))))
holds within the BS defined by context default-defaultct

I wonder if
((p18 (object (v12 <-- m17!)) (possessor v14) (rel (v13 <-- m4))))
holds within the BS defined by context default-defaultct

(m36!)

 CPU time : 0.03

*



;-----------------------------------------------------------------------------------------------
; time t2 occurs after time t4
;-----------------------------------------------------------------------------------------------

(add before *t4 after *t2)

Since
```

```
((m2! (forall (v5 <-- b7) (v4 <-- b2) (v3 <-- m3) (v2 <-- b3) (v1 <-- b1))
  (&ant (p5 (after (v5 <-- b7)) (before (v4 <-- b2)))
   (p4 (class (v3 <-- m3)) (member (v2 <-- b3)))
   (p3
    (event
     (p2 (act (p1 (action (m1 (lex (left)))) (object (v2 <-- b3))))
      (agent (v1 <-- b1))))
    (time (v4 <-- b2))))
  (cq
   (p8 (min 0) (max 0)
    (arg
     (p7
      (event
       (p6 (object (v2 <-- b3)) (possessor (v1 <-- b1)) (rel (v3 <-- m3))))
      (time (v5 <-- b7))))))))
and
((p5 (after (v5 <-- b7)) (before (v4 <-- b2))))
and
((p4 (class (v3 <-- m3)) (member (v2 <-- b3))))
and
((p3
  (event
   (p2 (act (p1 (action (m1 (lex (left)))) (object (v2 <-- b3))))
    (agent (v1 <-- b1))))
  (time (v4 <-- b2))))
I infer
((p8 (min 0) (max 0)
  (arg
   (p7
    (event
     (p6 (object (v2 <-- b3)) (possessor (v1 <-- b1)) (rel (v3 <-- m3))))
    (time (v5 <-- b7))))))

I know it is not the case that
((p7
  (event (p6 (object (v2 <-- b3)) (possessor (v1 <-- b1)) (rel (v3 <-- m3))))
  (time (v5 <-- b7))))

Since it is not the case that
```

```
((p10
  (event
   (p9 (object (v7 <-- b3)) (possessor (v6 <-- b1))
    (rel (m3 (lex (umbrella))))))
  (time (v8 <-- b7))))
I infer
((p11 (min 0) (max 0)
  (arg
   (p10
    (event
     (p9 (object (v7 <-- b3)) (possessor (v6 <-- b1))
      (rel (m3 (lex (umbrella))))))
    (time (v8 <-- b7))))))

Since
((m11! (forall v11)
  (ant
   (p16 (act (m7 (action (m5 (lex (get)))) (object (m6 (lex (wet))))))
    (agent v11)))
  (cq
   (p17 (act (m10 (action (m5 (lex (get)))) (object (m9 (lex (upset))))))
    (agent v11)))))
and
((p16 (act (m7 (action (m5 (lex (get)))) (object (m6 (lex (wet))))))
  (agent (v11 <-- b1))))
I infer
((p17 (act (m10 (action (m5 (lex (get)))) (object (m9 (lex (upset))))))
  (agent (v11 <-- b1))))

I wonder if
((p24 (object (v16 <-- m6)) (property (m13 (lex (unknown))))))
holds within the BS defined by context default-defaultct

I wonder if
((p24 (object v16) (property (m13 (lex (unknown))))))
holds within the BS defined by context default-defaultct

I wonder if
((p23 (act (p22 (action (m5 (lex (get)))) (object v17))) (agent (v15 <-- b1))))
```

```
holds within the BS defined by context default-defaultct

I wonder if
((p21 (act (p20 (action (m5 (lex (get)))) (object v16))) (agent (v15 <-- b1))))
holds within the BS defined by context default-defaultct

I know
((m17! (act (m16 (action (m5 (lex (get)))) (object (m15 (lex (drenched))))))
  (agent (b1))))

I know
((m42! (act (m7 (action (m5 (lex (get)))) (object (m6 (lex (wet))))))
  (agent (b1))))

I know
((m43! (act (m10 (action (m5 (lex (get)))) (object (m9 (lex (upset))))))
  (agent (b1))))

(m43! m42! m41! m39! m17!)

 CPU time : 0.09

*



;-----------------------------------------------------------------------------------------------
; The word drenched has the property of being unknown
;-----------------------------------------------------------------------------------------------

(add object (build lex "drenched") property (build lex "unknown") )

(m46! m45!)

 CPU time : 0.02

*

;-----------------------------------------------------------------------------------------------
;  Ask Cassie explicitly wether 'drenched' and 'upset' are synonyms
```

```
;-----------------------------------------------------------------------------------------------

(deduce synonym (build lex "drenched") synonym (build lex "upset"))

I wonder if
((m47 (synonym (m15 (lex (drenched))) (m9 (lex (upset))))))
holds within the BS defined by context default-defaultct

I wonder if
((p24 (object (v16 <-- m9)) (property (m13 (lex (unknown))))))
holds within the BS defined by context default-defaultct

I wonder if
((p23 (act (p22 (action (m5 (lex (get)))) (object (v17 <-- m15))))
  (agent v15)))
holds within the BS defined by context default-defaultct

I wonder if
((p23 (act (p22 (action (m5 (lex (get)))) (object (v17 <-- m9)))) (agent v15)))
holds within the BS defined by context default-defaultct

I wonder if
((p21 (act (p20 (action (m5 (lex (get)))) (object (v16 <-- m9)))) (agent v15)))
holds within the BS defined by context default-defaultct

I wonder if
((p21 (act (p20 (action (m5 (lex (get)))) (object (v16 <-- m15))))
  (agent v15)))
holds within the BS defined by context default-defaultct

Since
((m14! (forall (v17 <-- m9) (v16 <-- m15) (v15 <-- b1))
  (&ant (p24 (object (v16 <-- m15)) (property (m13 (lex (unknown)))))
   (p23 (act (p22 (action (m5 (lex (get)))) (object (v17 <-- m9))))
    (agent (v15 <-- b1)))
   (p21 (act (p20 (action (m5 (lex (get)))) (object (v16 <-- m15))))
    (agent (v15 <-- b1))))
  (cq (p25 (synonym (v17 <-- m9) (v16 <-- m15))))))
and
```

```
((p24 (object (v16 <-- m15)) (property (m13 (lex (unknown))))))
and
((p23 (act (p22 (action (m5 (lex (get)))) (object (v17 <-- m9))))
  (agent (v15 <-- b1))))
and
((p21 (act (p20 (action (m5 (lex (get)))) (object (v16 <-- m15))))
  (agent (v15 <-- b1))))
I infer
((p25 (synonym (v17 <-- m9) (v16 <-- m15))))

I know
((m17! (act (m16 (action (m5 (lex (get)))) (object (m15 (lex (drenched))))))
  (agent (b1))))

I know
((m43! (act (m10 (action (m5 (lex (get)))) (object (m9 (lex (upset))))))
  (agent (b1))))

I wonder if
((p16 (act (m7 (action (m5 (lex (get)))) (object (m6 (lex (wet))))))
  (agent v11)))
holds within the BS defined by context default-defaultct

Since
((m11! (forall v11)
  (ant
   (p16 (act (m7 (action (m5 (lex (get)))) (object (m6 (lex (wet))))))
    (agent v11)))
  (cq
   (p17 (act (m10 (action (m5 (lex (get)))) (object (m9 (lex (upset))))))
    (agent v11)))))
and
((p16 (act (m7 (action (m5 (lex (get)))) (object (m6 (lex (wet))))))
  (agent (v11 <-- b1))))
I infer
((p17 (act (m10 (action (m5 (lex (get)))) (object (m9 (lex (upset))))))
  (agent (v11 <-- b1))))

I know
```

```
((m42! (act (m7 (action (m5 (lex (get)))) (object (m6 (lex (wet))))))
  (agent (b1))))

I wonder if
((p14 (class (m4 (lex (rainstorm)))) (member v10)))
holds within the BS defined by context default-defaultct

I wonder if
((p13 (event v10) (time v9)))
holds within the BS defined by context default-defaultct

Since
((m8! (forall (v10 <-- b5) (v9 <-- b6) (v8 <-- b7) (v7 <-- b3) (v6 <-- b1))
  (&ant (p14 (class (m4 (lex (rainstorm)))) (member (v10 <-- b5)))
   (p13 (event (v10 <-- b5)) (time (v9 <-- b6)))
   (p12 (subinterval (v8 <-- b7)) (supinterval (v9 <-- b6)))
   (p11 (min 0) (max 0)
    (arg
     (p10
      (event
       (p9 (object (v7 <-- b3)) (possessor (v6 <-- b1))
        (rel (m3 (lex (umbrella))))))
      (time (v8 <-- b7))))))
  (cq
   (p15 (act (m7 (action (m5 (lex (get)))) (object (m6 (lex (wet))))))
    (agent (v6 <-- b1))))))
and
((p14 (class (m4 (lex (rainstorm)))) (member (v10 <-- b5))))
and
((p13 (event (v10 <-- b5)) (time (v9 <-- b6))))
and
((p12 (subinterval (v8 <-- b7)) (supinterval (v9 <-- b6))))
and
((p11 (min 0) (max 0)
  (arg
   (p10
    (event
     (p9 (object (v7 <-- b3)) (possessor (v6 <-- b1))
      (rel (m3 (lex (umbrella))))))
    (time (v8 <-- b7))))))
I infer
((p15 (act (m7 (action (m5 (lex (get)))) (object (m6 (lex (wet))))))
  (agent (v6 <-- b1))))

I know
((m32! (class (m4 (lex (rainstorm)))) (member (b5))))

I know
((m20!
  (event
   (m19 (act (m18 (action (m1 (lex (left)))) (iobject (b4)) (object (b3))))
    (agent (b1))))
  (time (b2))))

I know
((m27!
  (event
   (m26 (act (m25 (action (m1 (lex (left)))) (object (b3)))) (agent (b1))))
  (time (b2))))

I know
((m34! (event (b5)) (time (b6))))

I know
((m36!
  (event
   (m17! (act (m16 (action (m5 (lex (get)))) (object (m15 (lex (drenched))))))
    (agent (b1))))
  (time (b7))))

I know it is not the case that
((m40
  (event (m23! (object (b3)) (possessor (b1)) (rel (m3 (lex (umbrella))))))
  (time (b7))))

I wonder if
((p18 (object v12) (possessor v14) (rel (v13 <-- m4))))
holds within the BS defined by context default-defaultct
```

```
I wonder if
((p4 (class v3) (member v2)))
holds within the BS defined by context default-defaultct

Since
((m2! (forall (v5 <-- b7) (v4 <-- b2) (v3 <-- m3) (v2 <-- b3) (v1 <-- b1))
  (&ant (p5 (after (v5 <-- b7)) (before (v4 <-- b2)))
   (p4 (class (v3 <-- m3)) (member (v2 <-- b3)))
   (p3
    (event
     (p2 (act (p1 (action (m1 (lex (left)))) (object (v2 <-- b3))))
      (agent (v1 <-- b1))))
    (time (v4 <-- b2))))
  (cq
   (p8 (min 0) (max 0)
    (arg
     (p7
      (event
       (p6 (object (v2 <-- b3)) (possessor (v1 <-- b1)) (rel (v3 <-- m3))))
      (time (v5 <-- b7))))))))
and
((p5 (after (v5 <-- b7)) (before (v4 <-- b2))))
and
((p4 (class (v3 <-- m3)) (member (v2 <-- b3))))
and
((p3
  (event
   (p2 (act (p1 (action (m1 (lex (left)))) (object (v2 <-- b3))))
    (agent (v1 <-- b1))))
  (time (v4 <-- b2))))
I infer
((p8 (min 0) (max 0)
  (arg
   (p7
    (event
     (p6 (object (v2 <-- b3)) (possessor (v1 <-- b1)) (rel (v3 <-- m3))))
    (time (v5 <-- b7))))))
```

```
I know
((m24! (class (m3 (lex (umbrella)))) (member (b3))))

I know
((m30! (class (m28 (lex (office)))) (member (b4))))

I know
((m32! (class (m4 (lex (rainstorm)))) (member (b5))))

I wonder if
((p18 (object v12) (possessor v14) (rel v13)))
holds within the BS defined by context default-defaultct

Since
((m12! (forall v14 v13 v12) (ant (p18 (object v12) (possessor v14) (rel v13)))
  (cq (p19 (class v13) (member v12)))))
and
((p18 (object (v12 <-- b4)) (possessor (v14 <-- b1)) (rel (v13 <-- m28))))
I infer
((p19 (class (v13 <-- m28)) (member (v12 <-- b4))))

Since
((m12! (forall v14 v13 v12) (ant (p18 (object v12) (possessor v14) (rel v13)))
  (cq (p19 (class v13) (member v12)))))
and
((p18 (object (v12 <-- b3)) (possessor (v14 <-- b1)) (rel (v13 <-- m3))))
I infer
((p19 (class (v13 <-- m3)) (member (v12 <-- b3))))

Since
((m12! (forall v14 v13 v12) (ant (p18 (object v12) (possessor v14) (rel v13)))
  (cq (p19 (class v13) (member v12)))))
and
((p18 (object (v12 <-- b4)) (possessor (v14 <-- b1)) (rel (v13 <-- m28))))
I infer
((p19 (class (v13 <-- m28)) (member (v12 <-- b4))))

Since
((m12! (forall v14 v13 v12) (ant (p18 (object v12) (possessor v14) (rel v13)))
  (cq (p19 (class v13) (member v12)))))
and
((p18 (object (v12 <-- b4)) (possessor (v14 <-- b1)) (rel (v13 <-- m28))))
I infer
((p19 (class (v13 <-- m28)) (member (v12 <-- b4))))

Since
((m12! (forall v14 v13 v12) (ant (p18 (object v12) (possessor v14) (rel v13)))
  (cq (p19 (class v13) (member v12)))))
and
((p18 (object (v12 <-- b3)) (possessor (v14 <-- b1)) (rel (v13 <-- m3))))
I infer
((p19 (class (v13 <-- m3)) (member (v12 <-- b3))))

Since
((m12! (forall v14 v13 v12) (ant (p18 (object v12) (possessor v14) (rel v13)))
  (cq (p19 (class v13) (member v12)))))
and
((p18 (object (v12 <-- b3)) (possessor (v14 <-- b1)) (rel (v13 <-- m3))))
I infer
((p19 (class (v13 <-- m3)) (member (v12 <-- b3))))

Since
((m12! (forall v14 v13 v12) (ant (p18 (object v12) (possessor v14) (rel v13)))
  (cq (p19 (class v13) (member v12)))))
and
((p18 (object (v12 <-- b3)) (possessor (v14 <-- b1)) (rel (v13 <-- m3))))
I infer
((p19 (class (v13 <-- m3)) (member (v12 <-- b3))))

I know
((m23! (object (b3)) (possessor (b1)) (rel (m3 (lex (umbrella))))))

I know
((m29! (object (b4)) (possessor (b1)) (rel (m28 (lex (office))))))

(m47!)

 CPU time : 0.27
```

```
*

; Ask Cassie what "WORD" means:
^(
--> defineNoun "drenched")
 Definition of drenched:
 Synonyms: wet, upset,
nil



 CPU time : 0.04

*

;-------------------------------------------------------------------------------------------------
;Display all the nodes
;-------------------------------------------------------------------------------------------------
(describe *nodes)
(m47! (synonym (m15 (lex drenched)) (m9 (lex upset))))
(m46! (synonym (m15) (m6 (lex wet))))
(m45! (object (m15)) (property (m13 (lex unknown))))
(m43! (act (m10 (action (m5 (lex get))) (object (m9)))) (agent b1))
(m42! (act (m7 (action (m5)) (object (m6)))) (agent b1))
(m41! (min 0) (max 0)
 (arg
  (m40 (event (m23! (object b3) (possessor b1) (rel (m3 (lex umbrella)))))
   (time b7))))
(m39! (after b7) (before b2))
(m36! (event (m17! (act (m16 (action (m5)) (object (m15)))) (agent b1)))
 (time b7))
(m35! (subinterval b7) (supinterval b6))
(m34! (event b5) (time b6))
(m32! (class (m4 (lex rainstorm))) (member b5))
(m30! (class (m28 (lex office))) (member b4))
(m29! (object b4) (possessor b1) (rel (m28)))
(m27!
 (event (m26 (act (m25 (action (m1 (lex left))) (object b3))) (agent b1)))
 (time b2))
(m24! (class (m3)) (member b3))
(m22! (object b1) (proper-name Suda))
(m21!
 (cause
  (m20!
   (event (m19 (act (m18 (action (m1)) (iobject b4) (object b3))) (agent b1)))
   (time b2)))
 (effect (m17!)))
(m14! (forall v17 v16 v15)
 (&ant (p24 (object v16) (property (m13)))
  (p23 (act (p22 (action (m5)) (object v17))) (agent v15))
  (p21 (act (p20 (action (m5)) (object v16))) (agent v15)))
 (cq (p25 (synonym v17 v16))))
(m12! (forall v14 v13 v12) (ant (p18 (object v12) (possessor v14) (rel v13)))
 (cq (p19 (class v13) (member v12))))
(m11! (forall v11) (ant (p16 (act (m7)) (agent v11)))
 (cq (p17 (act (m10)) (agent v11))))
(m8! (forall v10 v9 v8 v7 v6)
 (&ant (p14 (class (m4)) (member v10)) (p13 (event v10) (time v9))
  (p12 (subinterval v8) (supinterval v9))
  (p11 (min 0) (max 0)
   (arg (p10 (event (p9 (object v7) (possessor v6) (rel (m3)))) (time v8)))))
 (cq (p15 (act (m7)) (agent v6))))
(m2! (forall v5 v4 v3 v2 v1)
 (&ant (p5 (after v5) (before v4)) (p4 (class v3) (member v2))
  (p3 (event (p2 (act (p1 (action (m1)) (object v2))) (agent v1))) (time v4)))
 (cq
  (p8 (min 0) (max 0)
   (arg (p7 (event (p6 (object v2) (possessor v1) (rel v3))) (time v5))))))

(m47! m46! m45! m43! m42! m41! m40 m39! m36! m35! b7 m34! b6 m32! b5 m30! m29!
 m28 office m27! m26 m25 m24! m23! m22! Suda m21! m20! m19 m18 b4 b3 b2 m17!
 m16 m15 drenched b1 m14! p25 p24 m13 unknown p23 p22 p21 p20 v17 v16 v15 m12!
 p19 p18 v14 v13 v12 m11! p17 m10 m9 upset p16 v11 m8! p15 m7 m6 wet m5 get
 p14 m4 rainstorm p13 p12 p11 p10 p9 m3 umbrella v10 v9 v8 v7 v6 m2! p8 p7 p6
 0 p5 p4 p3 p2 p1 m1 left v5 v4 v3 v2 v1)
 CPU time : 0.02

*

End of /home/csgrad/kothari4/seminar/project/drenched.demo demonstration.


 CPU time : 1.01
```