ACQUIRING MEANINGS OF ADJECTIVES FROM CONTEXT

Ananthakrishnan Ugrasenan Santha

29th April 2003

CSE 740: Seminar : Contextual Vocabulary Acquisition

Abstract

This paper discusses the issues involved in having a cognitive agent arrive at the meaning of an unknown adjective from its context. The paper reports what background knowledge and rules were made use of and what was the result obtained. There is also a discussion on how the existing adjective definition algorithm might be improved.

1. Introduction

This paper describes an attempt to have a computational cognitive agent deduce the meaning of an adjective from context, and examines some of the issues involved. The work was part of the NSF funded Contextual Vocabulary Acquisition (CVA) project at the University at Buffalo (Rapaport & Ehrlich 2000) (Rapaport & Kibby 2002). This work picks up from where Adam Lammert (Lammert 2002) left off.

The CVA project uses CASSIE (hereafter Cassie), the Cognitive Agent of the SNePS[1] System (Shapiro & Rapaport 1987) (Shapiro & Rapaport 1995). The contexts and the relevant background is "told" to Cassie through SNePSUL – the SNePS User Language. Later, when asked, Cassie tries to deduce the meaning from the contexts given and the background knowledge, which includes appropriate rules of inference.

2. The Adjective and the Passage

The adjective this work was based on is "taciturn". This is the same word that Lammert (Lammert 2002), and before him, Kazuhiro Kawachi (Kawachi 2001) and Christopher Garver (Garver 2002) worked on. But most of the previous work was based on this adjective appearing in contexts containing *unlike*. This adjective is in the list of words of the CVA project – in fact this is the only adjective in the list. There is also a set of CVA Think-Aloud Protocols compiled by K. Wieland which was used to find out what background information and rules are used by human readers to do the same task.

The passage used for this study is one from the list of CVA contexts for the

1 Semantic Network Processing System

adjective:

> In those early years, I who was very young, along with many others who were not found myself awed and tongue-tied in his presence. This reticence in turn affected George, who, misunderstanding it, acted withdrawn and taciturn, confirming our expectations. I felt he deemed us not quite up to his advanced level of reason and knowledge a fact that was certainly true but was not, I believe, a correct assessment of his reactions. Looking back this now seems very mixed, but it certainly seemed real at the time and continued to affect George's relations with others for years to come.
> [George Gaylord Simpson, Biographical Memoirs Volume 60. National Academy Press., p. 332- 333.]

The second sentence is the one that contains the unknown adjective. A basic assumption made throughout the CVA project is that except for the unknown word, the cognitive agent (reader) knows all the other words in the passage. Assuming this, the first two sentences were rephrased to:

> "My manner was reticent towards George. This caused George to act withdrawn and taciturn".

The third and fourth sentences were not represented because they were not thought of as influencing the meaning formation (in the right direction) based on the CVA think-aloud protocols.

3. Background Knowledge

The choice of background knowledge is crucial to the task. Some of it are things that human beings will infer without even being conscious of it. For example the fact that there is a person by name of George. But Cassie will have to be explicitly told this. The following two assertions are of this nature.

```
Assertion 1
;;; There is an object with proper name George.
(describe (assert object #George
                    proper-name (build lex "George")))

Assertion 2
;;; George and I are persons.
(describe (assert member (#I *George)
                    class (build lex "person")))
```

The assertions that follow have to do with the assumption that Cassie knows everything in the passage except the unknown word. This means Cassie knows (or has to

be told) the meaning of *reticent* and *withdrawn* and also the fact that they are manners. Someone who knows the meaning of both will know that they are synonyms. This was represented by saying that both of them are synonymous with *untalkative.*[2] One thing to be noted in the representations is that the *words* are told to be synonyms while the *concepts* they represent are told to be manners. While a human being would make this inference automatically, Cassie will have to be explicitly told this.

```
Assertion 3
;;; The words 'reticent' and 'withdrawn' are
;;; synonyms of 'untalkative'
(describe (assert object ("reticent" "withdrawn")
                         rel (build lex "synonym")
                         object "untalkative"))

Assertion 4
;;; The concepts denoted by 'reticent' and 'withdrawn'
;;; are manners.
(describe (assert member ((build lex "reticent")
                                    (build lex "withdrawn"))
                         class (build lex "manner")))
```

4. Background Rules

The main strategy is to use the fact that it was the speaker's reticent manner that caused George to act withdrawn and taciturn. A simplified paraphrase of the main rule (rule 7) used is as follows:

> If A acting in manner X causes B to act Y and Z
> and if X and Y are 'similar'
> and Z is unknown,
> then Z is 'similar' to X and Y.

The idea is to use this to have Cassie come up with *untalkative, withdrawn,* and *reticent* as possible synonyms of *taciturn.* For this to happen, we will have to have supporting rules that connect 'similarity' and 'possible synonymity'. The following *forall* statements are the rules used:

---

2 Untalkative, which is certainly in *my* vocabulary does not find entries in the online versions of the Merriam-Webster dictionary, www.m-w.com or the Collins Cobuild dictionary, http://www.linguistics.ruhr-uni-bochum.de:8099/. However www.dictionary.com defines it as an adjective which means “temperamentally disinclined to talk”. Reticent is cited as a synonym.

Rule 1

```
;;; If word1 and word2 are synonyms
;;; and something has the property denoted by word1
;;; then it has the property denoted by word2
(describe (assert forall ($thing $word1 $word2)
        &ant (build object *word1
                        rel (build lex "synonym")
                        object *word2)
        &ant (build object *thing
                                property (build lex *word1))
        cq (build object *thing
                                property (build lex *word2))))
```

Rule 2

```
;;; If word1 and word2 are synonyms
;;; then the concept denoted by word1
;;; is similar to the concept denoted by word2
(describe (assert forall ($word1$word2 $syn)
        ant (build object *word1
                        rel (build lex "synonym")
                        object *word2)
        cq (build object (build lex *word1)
                        rel (build lex "similar")
                        object (build lex word2))))
```

Rule 3

```
;;; If word1 and word2 have a common synonym syn
;;; then the concept denoted by word1
;;; is similar to the concept denoted by word2
(describe (assert forall ($word1 $word2 $syn)
        &ant (build object *word1
                        rel (build lex "synonym")
                        object *syn)
        &ant (build object *word2
                        rel (build lex "synonym")
                        object *syn)
        cq (build object (build lex *word1)
                        rel (build lex "similar")
                        object (build lex *word2))))
```

Rule 4

```
;;; If the concept denoted by word1
;;; is similar to the concept denoted by word2
```

```
;;; then word1 is a possible synonym of word2
(describe (assert forall ($word1 $word2)
        ant (build object(build lex *word1)
                        rel (build lex "similar")
                        object (build lex *word2))
        cq (build object *word1
                        rel (build lex "possible_synonym")
                        object *word2)))

Rule 5
;;; If word1 is a synonym of word2,
;;; then word1 is a possible_synonym of word2
(describe (assert forall ($word1 $word2)
        ant (build object *word1
                        rel (build lex "synonym")
                        object *word2)
        cq (build object *word1
                        rel (build lex "possible_synonym")
                        object *word2)))

Rule 6
;;; If obj1 is similar to obj2,
;;; and obj1 is a member of the class cl
;;; then obj2 is a member of the class cl
(describe (assert forall ($obj1 $obj2 $cl)
        &ant (build object *obj1
                        rel (build lex "similar")
                        object *obj2)
        &ant (build member *obj1
                        class *cl)
        cq (build member *obj2
                        class *cl)))

Rule 7
;;; If some person prsnA's manner mnnrA, which has property propA
;;;     caused some person prsnB to have
;;;     properties propB1 and propB2
;;; and if propA and propB1 are similar,
;;; then propB2 is similar to propB1 and propA
(describe (assert forall ($prsnA $prsnB $mnnrA $propA $propB1 $propB2)
        &ant (build member *prsnA class (build lex "person"))
        &ant (build possessor *prsnA
                        rel (build lex "manner")
```

```
                object *mnnrA)
        &ant (build object *mnnrA property *propA)
        &ant (build member *prsnB class (build lex "person"))
        &ant (build cause *mnnrA
                effect (build object *prsnB
                                                property (*propB1 *propB2)))
        &ant (build object *propA
                rel (build lex "similar")
                object *propB1)
        cq (build object *propB2
                rel (build lex "similar")
                object (*propB1 *propA))))
```

5. Representing the Passage

Now we get to the passage. The passage to be represented is:

"My manner was reticent towards George. This caused George to act withdrawn and taciturn".

The first sentence can be split into two for ease of representation as follows:

- My manner was reticent.
- This manner (reticence) was directed at George.

```
Sentence 1
; "My manner was reticent."
;;;     possessor: I
;;;     rel: (Node with lex "manner")
;;;     object: (BaseNode with property "reticent")
(describe (assert possessor *I
                rel (build lex "manner")
                object #mymanner))
(describe (assert object *mymanner
                property (build lex "reticent")))

; "This manner (reticence) was directed at George."
;;;     object: (my reticence)
;;;     direction: George
(describe (assert object *mymanner
                direction *George))
```

The second sentence was represented as below:

```
Sentence 2
;; "This (reticent manner) caused George to act withdrawn and taciturn."
(describe (assert cause *mymanner
                effect (assert object *George
                             property ((build lex "withdrawn")
                                       (build lex taciturn)))))
```

6. Modifying the Algorithm

The existing algorithm was Lammert's *unlike* algorithm (Lammert 2002). It was left the same as it was except for the following minor changes. The 'possible_equiv' in the adj_info structure that holds the definition of the adjective was replaced with 'possible_synonyms'. Also, the corresponding function, findPossEquiv was replaced with a function findPossibleSynonyms which looked for words that are the possible synonyms of the given adjective.

Note that because the background rules were asserted and not added, for the rules to fire, they have to be triggered with deduce statements.

On calling the adjective definition algorithm, we get the following definition.

```
;;;Definition of taciturn.
^(
--> define_adjective "taciturn");
#S(adj_info :adjective "taciturn" :unlike_props nil :class (manner)
        :type_nouns (person)
        :possible_synonyms
        (taciturn untalkative withdrawn reticent))
```

There are a couple of points to be noted about the definition. Firstly, the adjective itself, taciturn, was returned as a possible synonym. This can be taken care of by adding a final refine step to the algorithm which checks for such redundancies. Secondly, the only reason that type_nouns does not contain manner along with person is that Cassie was not explicitly asked that.

7. Possible Future Work

The most important area that needs work is the algorithm itself. While the noun definition algorithm and to a lesser extent the verb definition algorithm have received a a

lot of attention, vary little work has been done on the adjective definition algorithm. As mentioned in the previous section, one immediate and reasonably next step would be to add a refine step which examines the definition for possible redundancies and gets rid of those. This step will be especially important in the context of some of the other possible improvements we are going to discuss.

One of the reasons why adjectives receive comparatively lesser attention than nouns and verbs is that they are far more difficult to define from the context. Granger classifies adjectives as the most difficult to derive meanings of from context (Granger 1977). This difficulty is shared by humans also. One possible way to deal with this difficulty is to figure out some way to get as much information about the unknown adjective as is available from each context. For instance consider the following context:

> A terrible, beetle-browed, mastiff-mouthed, yellow-skinned, broad-bottomed, grim-taciturn individual; with a pair of dull-cruel-looking black eyes, and as much Parliamentary intellect and silent-rage in him ... as I have ever seen in any man.
> --Thomas Carlyle, Letter, June 24, 1824, to his brother. New Letters of Thomas Carlyle (1904).
> http://www.bartleby.com/66/95/10495.html

From this passage all we can infer about taciturn is that it is something that describes a certain individual and that it is highly likely to be a very undesirable quality. We cannot even be sure if it is a physical or personality trait, unless we stretch the fact that it appears alongside grim and hence likely to be a personality trait. These facts about taciturn, while nowhere near a dictionary-like definition or coming up with synonyms, are nevertheless valuable when taken together, especially if this passage is all we have to work with.

In order to make use of the maximum possible and losing the minimum possible, the adjective definition structure will have to have fields that can hold all this information. For example, a field to hold the fact whether the adjective has a positive or negative connotation, another one to hold possible actions that might act as triggers etc. The entries in these less important fields can be screened out in the refine step if we have better stuff like synonyms.

8. Other Interesting Issues

As mentioned before, one of the fundamental assumptions we make in the CVA project is that *the* unknown word is the only unknown word in the passage. But this is frequently not the case. Consider the original passage. Restating the first two sentences:

> In those early years, I who was very young, along with many others who were not found

myself awed and tongue-tied in his presence. This reticence in turn affected George, who, misunderstanding it, acted withdrawn and taciturn, confirming our expectations.

Any person who does not know the meaning of the word reticence also, in which case our rule will not work. But we can safely assume that most people would know the meaning of *tongue-tied, awed* and *withdrawn.* In this case most people would not find it difficult to infer that reticent means either tongue-tied or awed. And since reticence caused another person to be withdrawn, which is similar to tongue-tied, reticence is most likely similar to withdrawn and tongue-tied. After this inference our rule will deduce all these to be similar to taciturn.

This means that this assumption that there be only one unknown word per passage is not a *real* constraint, but just one that is there for convenience. Provided the meanings are derivable from context, the number of unknown words per passage creates little difference other than, perhaps, make the rules somewhat more in number.

9. Conclusion

The amount of work done with adjectives in the CVA project is small compared to nouns or verbs. A lot more adjectives in a lot more contexts will have to be examined to arrive at a good adjective definition structure as well as a good, general algorithm.

10. References

1. 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.

2. Rapaport, William J., & Kibby, Michael W. (2002), "ROLE: Contextual Vocabulary Acquisition: From Algorithm to Curriculum".

3. Shapiro, Stuart C., & Rapaport, William J. (1987), "SNePS Considered as a Fully Intensional Propositional Semantic Network", in Nick Cercone & Gordon McCalla (eds.), The Knowledge Frontier: Essays in the Representation of Knowledge (New York: Springer-Verlag): 262-315.

4. Shapiro, Stuart C. and Rapaport, William J. (1995), "An Introduction to a Computational Reader of Narrative", in Judith Felson Duchan, Gail A. Bruder, &

Lynne E. Hewitt (eds.), Deixis in Narrative: A Cognitive Science Perspective (Hillsdale, NJ: Lawrence Erlbaum Associates): 79-105.

5. Lammert, Adam (2002) "Defining Adjectives Through Contextual Vocabulary Acquisition"

6. Kawachi, Kazuhiro (2001) "Vocabulary Acquisition from the Context Containing *unlike*." 2001.

7. Garver, Christopher (2002) "Adjective Representation in Contextual Vocabulary Acquisition"

8. CVA Passages for taciturn
   http://www.cse.buffalo.edu/~rapaport/CVA/Taciturn/taciturn.txt

9. Wieland, K. (2002) CVA think-aloud protocols for taciturn (vedosarn)
   http://www.cse.buffalo.edu/~rapaport/CVA/

10. Granger, Richard H. (1977), "Foul-Up: a Program that Figures Out Meanings of Words from Context", Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI-77, MIT) (Los Altos, CA: William Kaufmann): 67-68.

```
Script started on Tue Apr 29 10:57:52 2003
pollux {~/cva_cse740/Taciturn} > acl
International Allegro CL Enterprise Edition
6.2 [Solaris] (Aug 15, 2002 14:24)
Copyright (C) 1985-2002, Franz Inc., Berkeley, CA, USA.  All Rights Reserved.

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

;; Optimization settings: safety 1, space 1, speed 1, debug 2.
;; For a complete description of all compiler switches given the
;; current optimization settings evaluate (explain-compiler-settings).
;;---
;; Current reader case mode: :case-sensitive-lower
cl-user(1): :ld /projects/snwiz/bin/sneps
; Loading /projects/snwiz/bin/sneps.lisp
Loading system SNePS...10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
SNePS-2.6 [PL:0a 2002/09/30 22:37:46] loaded.
Type `(sneps)' or `(snepslog)' to get started.
cl-user(2): :ld adjective_defn.lisp
; Loading /home/csgrad/aku2/cva_cse740/Taciturn/adjective_defn.lisp
cl-user(3): (sneps)


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

Copyright (C) 1984--2002 by Research Foundation of
State University of New York. SNePS comes with ABSOLUTELY NO WARRANTY!
Type `(copyright)' for detailed copyright information.
Type `(demo)' for a list of example applications.

   4/29/2003 10:58:52

* (demo "taciturn.demo")

File /home/csgrad/aku2/cva_cse740/Taciturn/taciturn.demo is now the source of input.


 CPU time : 0.02

* ;;;   The Taciturn Demo
;;;     CSE 740: Seminar: Contextual Vocabulary Acquisition
;;;     Ananthakrishnan Ugrasenan Santha

;;; Passage:
;;;"My manner was reticent towards George. This caused George
;;; to act withdrawn and taciturn".

;;; Reset Network
(resetnet t)

Net reset


 CPU time : 0.01

*
;;; Define relations:
;;; lex,object,proper-name,class,member,rel
;;; possessor,direction,property,cause,effect,
(define lex object proper-name class member rel
        possessor direction property cause effect)
#effect is already defined.

(lex object proper-name class member rel possessor direction property
 cause effect)

 CPU time : 0.01

*
;;*********************************************;;
;;*****       Background Knowledge        *****;;
;;*********************************************;;

;;; There is an object with propername George.
(describe (assert object #George
                proper-name (build lex "George")))

(m2! (object b1) (proper-name (m1 (lex George))))
```

```
(m2!)

 CPU time : 0.00

*
;;; George and I are persons.
(describe (assert member (#I *George)
                class (build lex "person")))

(m4! (class (m3 (lex person))) (member b2 b1))

(m4!)

 CPU time : 0.00

*
;;; The words 'reticent' and 'withdrawn' are
;;; synonyms of 'untalkative'
(describe (assert object ("reticent" "withdrawn")
                rel (build lex "synonym")
                object "untalkative"))

(m6! (object untalkative withdrawn reticent) (rel (m5 (lex synonym))))

(m6!)

 CPU time : 0.01

*
;;; The concepts denoted by 'reticent' and 'withdrawn'
;;; are manners.
(describe (assert member ((build lex "reticent")
                          (build lex "withdrawn"))
                class (build lex "manner")))
(m10! (class (m9 (lex manner)))
 (member (m8 (lex withdrawn)) (m7 (lex reticent))))

(m10!)

 CPU time : 0.00

*

;;*******************************************;;
;;*****       Background Rules            *****;;
;;*******************************************;;

;;; If word1 and word2 are synonyms
;;; and something has the property denoted by word1
;;; then it has the property denoted by word2
(describe (assert forall ($thing $word1 $word2)
        &ant (build object *word1
                rel (build lex "synonym")
                object *word2)
        &ant (build object *thing
                        property (build lex *word1))
        cq (build object *thing
                        property (build lex *word2))))

(m11! (forall v3 v2 v1)
 (&ant (p3 (object v1) (property (p2 (lex v2))))
  (p1 (object v3 v2) (rel (m5 (lex synonym)))))
 (cq (p5 (object v1) (property (p4 (lex v3))))))

(m11!)

 CPU time : 0.00

*
;;; If word1 and word2 are synonyms
;;; then the concept denoted by word1
;;; is similar to the concept denoted by word2
(describe (assert forall ($word1$word2 $syn)
        ant (build object *word1
                rel (build lex "synonym")
                object *word2)
        cq (build object (build lex *word1)
                rel (build lex "similar")
                object (build lex word2))))
```

```
(m14! (forall v6 v5 v4)
 (ant (p6 (object v5 v4) (rel (m5 (lex synonym)))))
 (cq
  (p8 (object (m13 (lex word2)) (p7 (lex v4)))
   (rel (m12 (lex similar))))))

(m14!)

 CPU time : 0.01

*
;;; If word1 and word2 have a common synonym syn
;;; then the concept denoted by word1
;;; is similar to the concept denoted by word2
(describe (assert forall ($word1 $word2 $syn)
        &ant (build object *word1
                rel (build lex "synonym")
                object *syn)
        &ant (build object *word2
                rel (build lex "synonym")
                object *syn)
        cq (build object (build lex *word1)
                rel (build lex "similar")
                object (build lex *word2))))

(m15! (forall v9 v8 v7)
 (&ant (p10 (object v9 v8) (rel (m5 (lex synonym))))
  (p9 (object v9 v7) (rel (m5))))
 (cq
  (p13 (object (p12 (lex v8)) (p11 (lex v7)))
   (rel (m12 (lex similar))))))

(m15!)

 CPU time : 0.02

*
;;; If the concept denoted by word1
;;; is similar to the concept denoted by word2
;;; then word1 is a possible synonym of word2
(describe (assert forall ($word1 $word2)
        ant (build object(build lex *word1)
                rel (build lex "similar")
                object (build lex *word2))
        cq (build object *word1
                rel (build lex "possible_synonym")
                object *word2)))

(m17! (forall v11 v10)
 (ant
  (p16 (object (p15 (lex v11)) (p14 (lex v10)))
   (rel (m12 (lex similar)))))
 (cq (p17 (object v11 v10) (rel (m16 (lex possible_synonym))))))

(m17!)

 CPU time : 0.01

*
;;; If word1 is a synonym of word2,
;;; then word1 is a possible_synonym of word2
(describe (assert forall ($word1 $word2)
        ant (build object *word1
                rel (build lex "synonym")
                object *word2)
        cq (build object *word1
                rel (build lex "possible_synonym")
                object *word2)))

(m18! (forall v13 v12)
 (ant (p18 (object v13 v12) (rel (m5 (lex synonym)))))
 (cq (p19 (object v13 v12) (rel (m16 (lex possible_synonym))))))

(m18!)

 CPU time : 0.08

*
;;; If obj1 is similar to obj2,
;;; and obj1 is a member of the class cl
```

```
;;; then obj2 is a member of the class cl
(describe (assert forall ($obj1 $obj2 $cl)
        &ant (build object *obj1
                rel (build lex "similar")
                object *obj2)
        &ant (build member *obj1
                class *cl)
        cq (build member *obj2
                class *cl)))

(m19! (forall v16 v15 v14)
 (&ant (p21 (class v16) (member v14))
  (p20 (object v15 v14) (rel (m12 (lex similar)))))
 (cq (p22 (class v16) (member v15))))

(m19!)

 CPU time : 0.01

*
;;; If some person prsnA's manner mnnrA, which has property propA
;;;     caused some person prsnB to have
;;;     properties propB1 and propB2
;;; and if propA and propB1 are similar,
;;; then propB2 is similar to propB1 and propA
(describe (assert forall ($prsnA $prsnB $mnnrA $propA $propB1 $propB2)
        &ant (build member *prsnA class (build lex "person"))
        &ant (build possessor *prsnA
                rel (build lex "manner")
                object *mnnrA)
        &ant (build object *mnnrA property *propA)
        &ant (build member *prsnB class (build lex "person"))
        &ant (build cause *mnnrA
                effect (build object *prsnB
                             property (*propB1 *propB2)))
        &ant (build object *propA
                rel (build lex "similar")
                object *propB1)
        cq (build object *propB2
                rel (build lex "similar")
                object (*propB1 *propA))))

(m20! (forall v22 v21 v20 v19 v18 v17)
 (&ant (p29 (object v21 v20) (rel (m12 (lex similar))))
  (p28 (cause v19) (effect (p27 (object v18) (property v22 v21))))
  (p26 (class (m3 (lex person))) (member v18))
  (p25 (object v19) (property v20))
  (p24 (object v19) (possessor v17) (rel (m9 (lex manner))))
  (p23 (class (m3)) (member v17)))
 (cq (p30 (object v22 v21 v20) (rel (m12)))))

(m20!)

 CPU time : 0.02

*
;;*******************************************;;
;;*****           Passage                 *****;;
;;*******************************************;;

;; "My manner was reticent towards George."
;; Split into: 1. My manner was reticent.
;;      and: 2. This reticence was directed at George.

;1. "My manner was reticent."
;;;     possessor: I
;;;     rel: (Node with lex "manner")
;;;     object: (BaseNode with property "reticent")
(describe (assert possessor *I
                rel (build lex "manner")
                object #mymanner))

(m21! (object b3) (possessor b2) (rel (m9 (lex manner))))

(m21!)

 CPU time : 0.01

* (describe (assert object *mymanner
                property (build lex "reticent")))
```

```
(m22! (object b3) (property (m7 (lex reticent))))

(m22!)

 CPU time : 0.00

*
;2. "This manner (reticence) was directed at George."
;;;     object: (my reticence)
;;;     direction: George
(describe (assert object *mymanner
                direction *George))

(m23! (direction b1) (object b3))

(m23!)

 CPU time : 0.00

*
;;; "This (reticent manner) caused George to act withdrawn and taciturn."
(describe (assert cause *mymanner
                effect (assert object *George
                               property ((build lex "withdrawn")
                                         (build lex taciturn)))))

(m26! (cause b3)
 (effect
  (m25! (object b1)
   (property (m24 (lex taciturn)) (m8 (lex withdrawn))))))

(m26!)

 CPU time : 0.01

*
;;*********************************************;;
;;*****           Deductions              *****;;
;;*********************************************;;

;;; Is taciturn a possible synonym of untalkative?
(describe (deduce object "taciturn"
                rel (find lex "possible_synonym")
                object "untalkative"))

I wonder if
((m27 (object (taciturn) (untalkative))
  (rel (m16 (lex (possible_synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p16
  (object (p15 (lex (v11 <-- taciturn)))
   (p14 (lex (v10 <-- untalkative))))
  (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p16
  (object (p15 (lex (v11 <-- untalkative)))
   (p14 (lex (v10 <-- taciturn))))
  (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p18 (object (v13 <-- taciturn) (v12 <-- untalkative))
  (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p18 (object (v13 <-- untalkative) (v12 <-- taciturn))
  (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p10 (object v9 (v8 <-- untalkative)) (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
```

```
((p10 (object v9 (v8 <-- taciturn)) (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p9 (object v9 (v7 <-- taciturn)) (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p9 (object v9 (v7 <-- untalkative)) (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p29 (object (v21 <-- m24) v20) (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p29 (object v21 (v20 <-- m24)) (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p29 (object (v21 <-- m28) v20) (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p29 (object (v21 <-- m28) (v20 <-- m24)) (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p29 (object v21 (v20 <-- m28)) (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p29 (object (v21 <-- m24) (v20 <-- m28)) (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p28 (cause v19)
  (effect (p27 (object v18) (property (v22 <-- m28) (v21 <-- m24))))))
holds within the BS defined by context default-defaultct

I wonder if
((p28 (cause v19)
  (effect (p27 (object v18) (property (v22 <-- m28) v21)))))
holds within the BS defined by context default-defaultct

I wonder if
((p28 (cause v19)
  (effect (p27 (object v18) (property (v22 <-- m24) (v21 <-- m28))))))
holds within the BS defined by context default-defaultct

I wonder if
((p28 (cause v19)
  (effect (p27 (object v18) (property v22 (v21 <-- m28))))))
holds within the BS defined by context default-defaultct

I wonder if
((p28 (cause v19)
  (effect (p27 (object v18) (property (v22 <-- m24) v21)))))
holds within the BS defined by context default-defaultct

I wonder if
((p28 (cause v19)
  (effect (p27 (object v18) (property v22 (v21 <-- m24))))))
holds within the BS defined by context default-defaultct

I wonder if
((p26 (class (m3 (lex (person)))) (member v18)))
holds within the BS defined by context default-defaultct

I wonder if
((p25 (object v19) (property v20)))
holds within the BS defined by context default-defaultct

I wonder if
((p25 (object v19) (property (v20 <-- m24))))
holds within the BS defined by context default-defaultct

I wonder if
((p25 (object v19) (property (v20 <-- m28))))
holds within the BS defined by context default-defaultct
```

```
I wonder if
((p24 (object v19) (possessor v17) (rel (m9 (lex (manner))))))
holds within the BS defined by context default-defaultct

I wonder if
((p23 (class (m3 (lex (person)))) (member v17)))
holds within the BS defined by context default-defaultct

I know
((m6! (object (untalkative) (withdrawn) (reticent))
  (rel (m5 (lex (synonym))))))

I know
((m26! (cause (b3))
  (effect
   (m25! (object (b1))
    (property (m24 (lex (taciturn))) (m8 (lex (withdrawn))))))))

I know
((m4! (class (m3 (lex (person)))) (member (b2) (b1))))

I know
((m22! (object (b3)) (property (m7 (lex (reticent))))))

I know
((m25! (object (b1))
  (property (m24 (lex (taciturn))) (m8 (lex (withdrawn))))))

I know
((m21! (object (b3)) (possessor (b2)) (rel (m9 (lex (manner))))))

I wonder if
((p6 (object v5 (v4 <-- untalkative)) (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p6 (object v5 (v4 <-- taciturn)) (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p10 (object v9 v8) (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

Since
((m14! (forall v6 v5 v4)
  (ant (p6 (object v5 v4) (rel (m5 (lex (synonym))))))
  (cq
   (p8 (object (m13 (lex (word2))) (p7 (lex v4)))
    (rel (m12 (lex (similar))))))))
and
((p6 (object (v5 <-- reticent) (v4 <-- untalkative))
  (rel (m5 (lex (synonym))))))
I infer
((p8 (object (m13 (lex (word2))) (p7 (lex (v4 <-- untalkative))))
  (rel (m12 (lex (similar))))))

Since
((m14! (forall v6 v5 v4)
  (ant (p6 (object v5 v4) (rel (m5 (lex (synonym))))))
  (cq
   (p8 (object (m13 (lex (word2))) (p7 (lex v4)))
    (rel (m12 (lex (similar))))))))
and
((p6 (object (v5 <-- withdrawn) (v4 <-- untalkative))
  (rel (m5 (lex (synonym))))))
I infer
((p8 (object (m13 (lex (word2))) (p7 (lex (v4 <-- untalkative))))
  (rel (m12 (lex (similar))))))

I wonder if
((p9 (object v9 v7) (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p29 (object v21 v20) (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p28 (cause v19) (effect (p27 (object v18) (property v22 v21)))))
```

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

I wonder if
((p21 (class (v16 <-- m3)) (member v14)))
holds within the BS defined by context default-defaultct

I wonder if
((p20 (object v15 v14) (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p3 (object v1) (property (p2 (lex v2)))))
holds within the BS defined by context default-defaultct

I wonder if
((p1 (object (v3 <-- untalkative) v2) (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p1 (object (v3 <-- taciturn) v2) (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p1 (object v3 v2) (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I know
((m6! (object (untalkative) (withdrawn) (reticent))
  (rel (m5 (lex (synonym))))))

Since
((m15!
  (forall (v9 <-- withdrawn) (v8 <-- reticent) (v7 <-- untalkative))
  (&ant
   (p10 (object (v9 <-- withdrawn) (v8 <-- reticent))
    (rel (m5 (lex (synonym)))))
   (p9 (object (v9 <-- withdrawn) (v7 <-- untalkative))
    (rel (m5 (lex (synonym))))))
  (cq
   (p13
    (object (p12 (lex (v8 <-- reticent)))
     (p11 (lex (v7 <-- untalkative))))
    (rel (m12 (lex (similar))))))))
and
((p10 (object (v9 <-- withdrawn) (v8 <-- reticent))
  (rel (m5 (lex (synonym))))))
and
((p9 (object (v9 <-- withdrawn) (v7 <-- untalkative))
  (rel (m5 (lex (synonym))))))
I infer
((p13
  (object (p12 (lex (v8 <-- reticent)))
   (p11 (lex (v7 <-- untalkative))))
  (rel (m12 (lex (similar))))))

Since
((m15!
  (forall (v9 <-- reticent) (v8 <-- untalkative) (v7 <-- withdrawn))
  (&ant
   (p10 (object (v9 <-- reticent) (v8 <-- untalkative))
    (rel (m5 (lex (synonym)))))
   (p9 (object (v9 <-- reticent) (v7 <-- withdrawn))
    (rel (m5 (lex (synonym))))))
  (cq
   (p13
    (object (p12 (lex (v8 <-- untalkative)))
     (p11 (lex (v7 <-- withdrawn))))
    (rel (m12 (lex (similar))))))))
and
((p10 (object (v9 <-- reticent) (v8 <-- untalkative))
  (rel (m5 (lex (synonym))))))
and
((p9 (object (v9 <-- reticent) (v7 <-- withdrawn))
  (rel (m5 (lex (synonym))))))
I infer
((p13
  (object (p12 (lex (v8 <-- untalkative)))
   (p11 (lex (v7 <-- withdrawn))))
  (rel (m12 (lex (similar))))))
```

```
Since
((m15!
  (forall (v9 <-- withdrawn) (v8 <-- untalkative) (v7 <-- reticent))
  (&ant
   (p10 (object (v9 <-- withdrawn) (v8 <-- untalkative))
    (rel (m5 (lex (synonym)))))
   (p9 (object (v9 <-- withdrawn) (v7 <-- reticent))
    (rel (m5 (lex (synonym))))))
  (cq
   (p13
    (object (p12 (lex (v8 <-- untalkative)))
     (p11 (lex (v7 <-- reticent))))
    (rel (m12 (lex (similar))))))))
and
((p10 (object (v9 <-- withdrawn) (v8 <-- untalkative))
  (rel (m5 (lex (synonym))))))
and
((p9 (object (v9 <-- withdrawn) (v7 <-- reticent))
  (rel (m5 (lex (synonym))))))
I infer
((p13
  (object (p12 (lex (v8 <-- untalkative)))
   (p11 (lex (v7 <-- reticent))))
  (rel (m12 (lex (similar))))))

Since
((m15!
  (forall (v9 <-- reticent) (v8 <-- withdrawn) (v7 <-- untalkative))
  (&ant
   (p10 (object (v9 <-- reticent) (v8 <-- withdrawn))
    (rel (m5 (lex (synonym)))))
   (p9 (object (v9 <-- reticent) (v7 <-- untalkative))
    (rel (m5 (lex (synonym))))))
  (cq
   (p13
    (object (p12 (lex (v8 <-- withdrawn)))
     (p11 (lex (v7 <-- untalkative))))
    (rel (m12 (lex (similar))))))))
and
((p10 (object (v9 <-- reticent) (v8 <-- withdrawn))
  (rel (m5 (lex (synonym))))))
and
((p9 (object (v9 <-- reticent) (v7 <-- untalkative))
  (rel (m5 (lex (synonym))))))
I infer
((p13
  (object (p12 (lex (v8 <-- withdrawn)))
   (p11 (lex (v7 <-- untalkative))))
  (rel (m12 (lex (similar))))))

I know
((m33! (object (untalkative) (withdrawn)) (rel (m5 (lex (synonym))))))

I know
((m34! (object (untalkative) (reticent)) (rel (m5 (lex (synonym))))))

I know
((m32! (object (m28 (lex (untalkative))) (m13 (lex (word2))))
  (rel (m12 (lex (similar))))))

I know
((m26! (cause (b3))
  (effect
   (m25! (object (b1))
    (property (m24 (lex (taciturn))) (m8 (lex (withdrawn)))))))))

I know
((m4! (class (m3 (lex (person)))) (member (b2) (b1))))

I know
((m35! (class (m3 (lex (person)))) (member (b2))))

I know
((m36! (class (m3 (lex (person)))) (member (b1))))

I know
((m22! (object (b3)) (property (m7 (lex (reticent))))))

Since
((m11! (forall (v3 <-- untalkative) (v2 <-- reticent) (v1 <-- b3))
```

```
  (&ant
   (p3 (object (v1 <-- b3)) (property (p2 (lex (v2 <-- reticent)))))
   (p1 (object (v3 <-- untalkative) (v2 <-- reticent))
    (rel (m5 (lex (synonym))))))
  (cq
   (p5 (object (v1 <-- b3))
    (property (p4 (lex (v3 <-- untalkative))))))))
and
((p3 (object (v1 <-- b3)) (property (p2 (lex (v2 <-- reticent))))))
and
((p1 (object (v3 <-- untalkative) (v2 <-- reticent))
  (rel (m5 (lex (synonym))))))
I infer
((p5 (object (v1 <-- b3)) (property (p4 (lex (v3 <-- untalkative))))))

Since
((m11! (forall (v3 <-- withdrawn) (v2 <-- reticent) (v1 <-- b3))
  (&ant
   (p3 (object (v1 <-- b3)) (property (p2 (lex (v2 <-- reticent)))))
   (p1 (object (v3 <-- withdrawn) (v2 <-- reticent))
    (rel (m5 (lex (synonym))))))
  (cq
   (p5 (object (v1 <-- b3)) (property (p4 (lex (v3 <-- withdrawn))))))))
and
((p3 (object (v1 <-- b3)) (property (p2 (lex (v2 <-- reticent))))))
and
((p1 (object (v3 <-- withdrawn) (v2 <-- reticent))
  (rel (m5 (lex (synonym))))))
I infer
((p5 (object (v1 <-- b3)) (property (p4 (lex (v3 <-- withdrawn))))))

Since
((m11! (forall (v3 <-- untalkative) (v2 <-- reticent) (v1 <-- b3))
  (&ant
   (p3 (object (v1 <-- b3)) (property (p2 (lex (v2 <-- reticent)))))
   (p1 (object (v3 <-- untalkative) (v2 <-- reticent))
    (rel (m5 (lex (synonym))))))
  (cq
   (p5 (object (v1 <-- b3))
    (property (p4 (lex (v3 <-- untalkative))))))))
and
((p3 (object (v1 <-- b3)) (property (p2 (lex (v2 <-- reticent))))))
and
((p1 (object (v3 <-- untalkative) (v2 <-- reticent))
  (rel (m5 (lex (synonym))))))
I infer
((p5 (object (v1 <-- b3)) (property (p4 (lex (v3 <-- untalkative))))))

Since
((m20!
  (forall (v22 <-- m24) (v21 <-- m8) (v20 <-- m28) (v19 <-- b3)
   (v18 <-- b1) (v17 <-- b2))
  (&ant
   (p29 (object (v21 <-- m8) (v20 <-- m28))
    (rel (m12 (lex (similar)))))
   (p28 (cause (v19 <-- b3))
    (effect
     (p27 (object (v18 <-- b1))
      (property (v22 <-- m24) (v21 <-- m8)))))
   (p26 (class (m3 (lex (person)))) (member (v18 <-- b1)))
   (p25 (object (v19 <-- b3)) (property (v20 <-- m28)))
   (p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
    (rel (m9 (lex (manner)))))
   (p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
  (cq
   (p30 (object (v22 <-- m24) (v21 <-- m8) (v20 <-- m28))
    (rel (m12 (lex (similar))))))))
and
((p29 (object (v21 <-- m8) (v20 <-- m28)) (rel (m12 (lex (similar))))))
and
((p28 (cause (v19 <-- b3))
  (effect
   (p27 (object (v18 <-- b1)) (property (v22 <-- m24) (v21 <-- m8))))))
and
((p26 (class (m3 (lex (person)))) (member (v18 <-- b1))))
and
((p25 (object (v19 <-- b3)) (property (v20 <-- m28))))
and
((p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
  (rel (m9 (lex (manner))))))
```

```
and
((p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
I infer
((p30 (object (v22 <-- m24) (v21 <-- m8) (v20 <-- m28))
  (rel (m12 (lex (similar))))))

Since
((m20!
  (forall (v22 <-- m24) (v21 <-- m8) (v20 <-- m28) (v19 <-- b3)
   (v18 <-- b1) (v17 <-- b2))
  (&ant
   (p29 (object (v21 <-- m8) (v20 <-- m28))
    (rel (m12 (lex (similar)))))
   (p28 (cause (v19 <-- b3))
    (effect
     (p27 (object (v18 <-- b1))
      (property (v22 <-- m24) (v21 <-- m8)))))
   (p26 (class (m3 (lex (person)))) (member (v18 <-- b1)))
   (p25 (object (v19 <-- b3)) (property (v20 <-- m28)))
   (p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
    (rel (m9 (lex (manner)))))
   (p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
  (cq
   (p30 (object (v22 <-- m24) (v21 <-- m8) (v20 <-- m28))
    (rel (m12 (lex (similar))))))))
and
((p29 (object (v21 <-- m8) (v20 <-- m28)) (rel (m12 (lex (similar))))))
and
((p28 (cause (v19 <-- b3))
  (effect
   (p27 (object (v18 <-- b1)) (property (v22 <-- m24) (v21 <-- m8))))))
and
((p26 (class (m3 (lex (person)))) (member (v18 <-- b1))))
and
((p25 (object (v19 <-- b3)) (property (v20 <-- m28))))
and
((p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
  (rel (m9 (lex (manner))))))
and
((p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
I infer
((p30 (object (v22 <-- m24) (v21 <-- m8) (v20 <-- m28))
  (rel (m12 (lex (similar))))))

Since
((m20!
  (forall (v22 <-- m8) (v21 <-- m24) (v20 <-- m28) (v19 <-- b3)
   (v18 <-- b1) (v17 <-- b2))
  (&ant
   (p29 (object (v21 <-- m24) (v20 <-- m28))
    (rel (m12 (lex (similar)))))
   (p28 (cause (v19 <-- b3))
    (effect
     (p27 (object (v18 <-- b1))
      (property (v22 <-- m8) (v21 <-- m24)))))
   (p26 (class (m3 (lex (person)))) (member (v18 <-- b1)))
   (p25 (object (v19 <-- b3)) (property (v20 <-- m28)))
   (p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
    (rel (m9 (lex (manner)))))
   (p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
  (cq
   (p30 (object (v22 <-- m8) (v21 <-- m24) (v20 <-- m28))
    (rel (m12 (lex (similar))))))))
and
((p29 (object (v21 <-- m24) (v20 <-- m28)) (rel (m12 (lex (similar))))))
and
((p28 (cause (v19 <-- b3))
  (effect
   (p27 (object (v18 <-- b1)) (property (v22 <-- m8) (v21 <-- m24))))))
and
((p26 (class (m3 (lex (person)))) (member (v18 <-- b1))))
and
((p25 (object (v19 <-- b3)) (property (v20 <-- m28))))
and
((p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
  (rel (m9 (lex (manner))))))
and
((p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
I infer
((p30 (object (v22 <-- m8) (v21 <-- m24) (v20 <-- m28))
```

```
  (rel (m12 (lex (similar))))))

Since
((m20!
  (forall (v22 <-- m8) (v21 <-- m24) (v20 <-- m28) (v19 <-- b3)
   (v18 <-- b1) (v17 <-- b2))
  (&ant
   (p29 (object (v21 <-- m24) (v20 <-- m28))
    (rel (m12 (lex (similar)))))
   (p28 (cause (v19 <-- b3))
    (effect
     (p27 (object (v18 <-- b1))
      (property (v22 <-- m8) (v21 <-- m24)))))
   (p26 (class (m3 (lex (person)))) (member (v18 <-- b1)))
   (p25 (object (v19 <-- b3)) (property (v20 <-- m28)))
   (p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
    (rel (m9 (lex (manner)))))
   (p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
  (cq
   (p30 (object (v22 <-- m8) (v21 <-- m24) (v20 <-- m28))
    (rel (m12 (lex (similar))))))))
and
((p29 (object (v21 <-- m24) (v20 <-- m28)) (rel (m12 (lex (similar))))))
and
((p28 (cause (v19 <-- b3))
  (effect
   (p27 (object (v18 <-- b1)) (property (v22 <-- m8) (v21 <-- m24))))))
and
((p26 (class (m3 (lex (person)))) (member (v18 <-- b1))))
and
((p25 (object (v19 <-- b3)) (property (v20 <-- m28))))
and
((p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
  (rel (m9 (lex (manner))))))
and
((p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
I infer
((p30 (object (v22 <-- m8) (v21 <-- m24) (v20 <-- m28))
  (rel (m12 (lex (similar))))))

Since
((m17! (forall v11 v10)
  (ant
   (p16 (object (p15 (lex v11)) (p14 (lex v10)))
    (rel (m12 (lex (similar))))))
  (cq (p17 (object v11 v10) (rel (m16 (lex (possible_synonym))))))))
and
((p16
  (object (p15 (lex (v11 <-- taciturn)))
   (p14 (lex (v10 <-- untalkative))))
  (rel (m12 (lex (similar))))))
I infer
((p17 (object (v11 <-- taciturn) (v10 <-- untalkative))
  (rel (m16 (lex (possible_synonym))))))

Since
((m17! (forall v11 v10)
  (ant
   (p16 (object (p15 (lex v11)) (p14 (lex v10)))
    (rel (m12 (lex (similar))))))
  (cq (p17 (object v11 v10) (rel (m16 (lex (possible_synonym))))))))
and
((p16
  (object (p15 (lex (v11 <-- untalkative)))
   (p14 (lex (v10 <-- taciturn))))
  (rel (m12 (lex (similar))))))
I infer
((p17 (object (v11 <-- untalkative) (v10 <-- taciturn))
  (rel (m16 (lex (possible_synonym))))))

I know
((m25! (object (b1))
  (property (m24 (lex (taciturn))) (m8 (lex (withdrawn))))))

Since
((m11! (forall (v3 <-- reticent) (v2 <-- withdrawn) (v1 <-- b1))
  (&ant
   (p3 (object (v1 <-- b1)) (property (p2 (lex (v2 <-- withdrawn)))))
   (p1 (object (v3 <-- reticent) (v2 <-- withdrawn))
    (rel (m5 (lex (synonym))))))
```

```
  (cq
   (p5 (object (v1 <-- b1)) (property (p4 (lex (v3 <-- reticent))))))))
and
((p3 (object (v1 <-- b1)) (property (p2 (lex (v2 <-- withdrawn))))))
and
((p1 (object (v3 <-- reticent) (v2 <-- withdrawn))
  (rel (m5 (lex (synonym))))))
I infer
((p5 (object (v1 <-- b1)) (property (p4 (lex (v3 <-- reticent))))))

Since
((m11! (forall (v3 <-- untalkative) (v2 <-- withdrawn) (v1 <-- b1))
  (&ant
   (p3 (object (v1 <-- b1)) (property (p2 (lex (v2 <-- withdrawn)))))
   (p1 (object (v3 <-- untalkative) (v2 <-- withdrawn))
    (rel (m5 (lex (synonym))))))
  (cq
   (p5 (object (v1 <-- b1))
    (property (p4 (lex (v3 <-- untalkative))))))))
and
((p3 (object (v1 <-- b1)) (property (p2 (lex (v2 <-- withdrawn))))))
and
((p1 (object (v3 <-- untalkative) (v2 <-- withdrawn))
  (rel (m5 (lex (synonym))))))
I infer
((p5 (object (v1 <-- b1)) (property (p4 (lex (v3 <-- untalkative))))))

Since
((m11! (forall (v3 <-- untalkative) (v2 <-- withdrawn) (v1 <-- b1))
  (&ant
   (p3 (object (v1 <-- b1)) (property (p2 (lex (v2 <-- withdrawn)))))
   (p1 (object (v3 <-- untalkative) (v2 <-- withdrawn))
    (rel (m5 (lex (synonym))))))
  (cq
   (p5 (object (v1 <-- b1))
    (property (p4 (lex (v3 <-- untalkative))))))))
and
((p3 (object (v1 <-- b1)) (property (p2 (lex (v2 <-- withdrawn))))))
and
((p1 (object (v3 <-- untalkative) (v2 <-- withdrawn))
  (rel (m5 (lex (synonym))))))
I infer
((p5 (object (v1 <-- b1)) (property (p4 (lex (v3 <-- untalkative))))))

I know
((m37! (object (b1)) (property (m24 (lex (taciturn))))))

I know
((m38! (object (b1)) (property (m8 (lex (withdrawn))))))

I wonder if
((p6 (object v5 v4) (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

Since
((m14! (forall v6 v5 v4)
  (ant (p6 (object v5 v4) (rel (m5 (lex (synonym))))))
  (cq
   (p8 (object (m13 (lex (word2))) (p7 (lex v4)))
    (rel (m12 (lex (similar))))))))
and
((p6 (object (v5 <-- withdrawn) (v4 <-- untalkative))
  (rel (m5 (lex (synonym))))))
I infer
((p8 (object (m13 (lex (word2))) (p7 (lex (v4 <-- untalkative))))
  (rel (m12 (lex (similar))))))

Since
((m14! (forall v6 v5 v4)
  (ant (p6 (object v5 v4) (rel (m5 (lex (synonym))))))
  (cq
   (p8 (object (m13 (lex (word2))) (p7 (lex v4)))
    (rel (m12 (lex (similar))))))))
and
((p6 (object (v5 <-- reticent) (v4 <-- untalkative))
  (rel (m5 (lex (synonym))))))
I infer
((p8 (object (m13 (lex (word2))) (p7 (lex (v4 <-- untalkative))))
  (rel (m12 (lex (similar))))))
```

```
Since
((m14! (forall v6 v5 v4)
  (ant (p6 (object v5 v4) (rel (m5 (lex (synonym))))))
  (cq
   (p8 (object (m13 (lex (word2))) (p7 (lex v4)))
    (rel (m12 (lex (similar))))))))
and
((p6 (object (v5 <-- reticent) (v4 <-- withdrawn))
  (rel (m5 (lex (synonym))))))
I infer
((p8 (object (m13 (lex (word2))) (p7 (lex (v4 <-- withdrawn))))
  (rel (m12 (lex (similar))))))

Since
((m14! (forall v6 v5 v4)
  (ant (p6 (object v5 v4) (rel (m5 (lex (synonym))))))
  (cq
   (p8 (object (m13 (lex (word2))) (p7 (lex v4)))
    (rel (m12 (lex (similar))))))))
and
((p6 (object (v5 <-- untalkative) (v4 <-- reticent))
  (rel (m5 (lex (synonym))))))
I infer
((p8 (object (m13 (lex (word2))) (p7 (lex (v4 <-- reticent))))
  (rel (m12 (lex (similar))))))

Since
((m14! (forall v6 v5 v4)
  (ant (p6 (object v5 v4) (rel (m5 (lex (synonym))))))
  (cq
   (p8 (object (m13 (lex (word2))) (p7 (lex v4)))
    (rel (m12 (lex (similar))))))))
and
((p6 (object (v5 <-- withdrawn) (v4 <-- reticent))
  (rel (m5 (lex (synonym))))))
I infer
((p8 (object (m13 (lex (word2))) (p7 (lex (v4 <-- reticent))))
  (rel (m12 (lex (similar))))))

Since
((m14! (forall v6 v5 v4)
  (ant (p6 (object v5 v4) (rel (m5 (lex (synonym))))))
  (cq
   (p8 (object (m13 (lex (word2))) (p7 (lex v4)))
    (rel (m12 (lex (similar))))))))
and
((p6 (object (v5 <-- untalkative) (v4 <-- withdrawn))
  (rel (m5 (lex (synonym))))))
I infer
((p8 (object (m13 (lex (word2))) (p7 (lex (v4 <-- withdrawn))))
  (rel (m12 (lex (similar))))))

Since
((m15!
  (forall (v9 <-- reticent) (v8 <-- withdrawn) (v7 <-- untalkative))
  (&ant
   (p10 (object (v9 <-- reticent) (v8 <-- withdrawn))
    (rel (m5 (lex (synonym)))))
   (p9 (object (v9 <-- reticent) (v7 <-- untalkative))
    (rel (m5 (lex (synonym))))))
  (cq
   (p13
    (object (p12 (lex (v8 <-- withdrawn)))
     (p11 (lex (v7 <-- untalkative))))
    (rel (m12 (lex (similar))))))))
and
((p10 (object (v9 <-- reticent) (v8 <-- withdrawn))
  (rel (m5 (lex (synonym))))))
and
((p9 (object (v9 <-- reticent) (v7 <-- untalkative))
  (rel (m5 (lex (synonym))))))
I infer
((p13
  (object (p12 (lex (v8 <-- withdrawn)))
   (p11 (lex (v7 <-- untalkative))))
  (rel (m12 (lex (similar))))))

Since
((m15!
  (forall (v9 <-- reticent) (v8 <-- untalkative) (v7 <-- withdrawn))
```

```
  (&ant
   (p10 (object (v9 <-- reticent) (v8 <-- untalkative))
    (rel (m5 (lex (synonym)))))
   (p9 (object (v9 <-- reticent) (v7 <-- withdrawn))
    (rel (m5 (lex (synonym))))))
  (cq
   (p13
    (object (p12 (lex (v8 <-- untalkative)))
     (p11 (lex (v7 <-- withdrawn))))
    (rel (m12 (lex (similar)))))))
and
((p10 (object (v9 <-- reticent) (v8 <-- untalkative))
  (rel (m5 (lex (synonym))))))
and
((p9 (object (v9 <-- reticent) (v7 <-- withdrawn))
  (rel (m5 (lex (synonym))))))
I infer
((p13
  (object (p12 (lex (v8 <-- untalkative)))
   (p11 (lex (v7 <-- withdrawn))))
  (rel (m12 (lex (similar))))))

Since
((m15!
  (forall (v9 <-- withdrawn) (v8 <-- untalkative) (v7 <-- reticent))
  (&ant
   (p10 (object (v9 <-- withdrawn) (v8 <-- untalkative))
    (rel (m5 (lex (synonym)))))
   (p9 (object (v9 <-- withdrawn) (v7 <-- reticent))
    (rel (m5 (lex (synonym))))))
  (cq
   (p13
    (object (p12 (lex (v8 <-- untalkative)))
     (p11 (lex (v7 <-- reticent))))
    (rel (m12 (lex (similar)))))))
and
((p10 (object (v9 <-- withdrawn) (v8 <-- untalkative))
  (rel (m5 (lex (synonym))))))
and
((p9 (object (v9 <-- withdrawn) (v7 <-- reticent))
  (rel (m5 (lex (synonym))))))
I infer
((p13
  (object (p12 (lex (v8 <-- untalkative)))
   (p11 (lex (v7 <-- reticent))))
  (rel (m12 (lex (similar))))))

Since
((m15!
  (forall (v9 <-- untalkative) (v8 <-- reticent) (v7 <-- withdrawn))
  (&ant
   (p10 (object (v9 <-- untalkative) (v8 <-- reticent))
    (rel (m5 (lex (synonym)))))
   (p9 (object (v9 <-- untalkative) (v7 <-- withdrawn))
    (rel (m5 (lex (synonym))))))
  (cq
   (p13
    (object (p12 (lex (v8 <-- reticent)))
     (p11 (lex (v7 <-- withdrawn))))
    (rel (m12 (lex (similar)))))))
and
((p10 (object (v9 <-- untalkative) (v8 <-- reticent))
  (rel (m5 (lex (synonym))))))
and
((p9 (object (v9 <-- untalkative) (v7 <-- withdrawn))
  (rel (m5 (lex (synonym))))))
I infer
((p13
  (object (p12 (lex (v8 <-- reticent))) (p11 (lex (v7 <-- withdrawn))))
  (rel (m12 (lex (similar))))))

Since
((m15!
  (forall (v9 <-- untalkative) (v8 <-- withdrawn) (v7 <-- reticent))
  (&ant
   (p10 (object (v9 <-- untalkative) (v8 <-- withdrawn))
    (rel (m5 (lex (synonym)))))
   (p9 (object (v9 <-- untalkative) (v7 <-- reticent))
    (rel (m5 (lex (synonym))))))
  (cq
```

```
   (p13
    (object (p12 (lex (v8 <-- withdrawn)))
     (p11 (lex (v7 <-- reticent))))
    (rel (m12 (lex (similar))))))))
and
((p10 (object (v9 <-- untalkative) (v8 <-- withdrawn))
  (rel (m5 (lex (synonym))))))
and
((p9 (object (v9 <-- untalkative) (v7 <-- reticent))
  (rel (m5 (lex (synonym))))))
I infer
((p13
  (object (p12 (lex (v8 <-- withdrawn))) (p11 (lex (v7 <-- reticent))))
  (rel (m12 (lex (similar))))))

Since
((m15!
  (forall (v9 <-- withdrawn) (v8 <-- reticent) (v7 <-- untalkative))
  (&ant
   (p10 (object (v9 <-- withdrawn) (v8 <-- reticent))
    (rel (m5 (lex (synonym)))))
   (p9 (object (v9 <-- withdrawn) (v7 <-- untalkative))
    (rel (m5 (lex (synonym))))))
  (cq
   (p13
    (object (p12 (lex (v8 <-- reticent)))
     (p11 (lex (v7 <-- untalkative))))
    (rel (m12 (lex (similar))))))))
and
((p10 (object (v9 <-- withdrawn) (v8 <-- reticent))
  (rel (m5 (lex (synonym))))))
and
((p9 (object (v9 <-- withdrawn) (v7 <-- untalkative))
  (rel (m5 (lex (synonym))))))
I infer
((p13
  (object (p12 (lex (v8 <-- reticent)))
   (p11 (lex (v7 <-- untalkative))))
  (rel (m12 (lex (similar))))))

Since
((m20!
  (forall (v22 <-- m24) (v21 <-- m8) (v20 <-- m7) (v19 <-- b3)
   (v18 <-- b1) (v17 <-- b2))
  (&ant
   (p29 (object (v21 <-- m8) (v20 <-- m7)) (rel (m12 (lex (similar)))))
   (p28 (cause (v19 <-- b3))
    (effect
     (p27 (object (v18 <-- b1))
      (property (v22 <-- m24) (v21 <-- m8)))))
   (p26 (class (m3 (lex (person)))) (member (v18 <-- b1)))
   (p25 (object (v19 <-- b3)) (property (v20 <-- m7)))
   (p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
    (rel (m9 (lex (manner)))))
   (p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
  (cq
   (p30 (object (v22 <-- m24) (v21 <-- m8) (v20 <-- m7))
    (rel (m12 (lex (similar))))))))
and
((p29 (object (v21 <-- m8) (v20 <-- m7)) (rel (m12 (lex (similar))))))
and
((p28 (cause (v19 <-- b3))
  (effect
   (p27 (object (v18 <-- b1)) (property (v22 <-- m24) (v21 <-- m8))))))
and
((p26 (class (m3 (lex (person)))) (member (v18 <-- b1))))
and
((p25 (object (v19 <-- b3)) (property (v20 <-- m7))))
and
((p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
  (rel (m9 (lex (manner))))))
and
((p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
I infer
((p30 (object (v22 <-- m24) (v21 <-- m8) (v20 <-- m7))
  (rel (m12 (lex (similar))))))

Since
((m20!
  (forall (v22 <-- m8) (v21 <-- m24) (v20 <-- m7) (v19 <-- b3)
```

```
   (v18 <-- b1) (v17 <-- b2))
  (&ant
   (p29 (object (v21 <-- m24) (v20 <-- m7))
    (rel (m12 (lex (similar)))))
   (p28 (cause (v19 <-- b3))
    (effect
     (p27 (object (v18 <-- b1))
      (property (v22 <-- m8) (v21 <-- m24)))))
   (p26 (class (m3 (lex (person)))) (member (v18 <-- b1)))
   (p25 (object (v19 <-- b3)) (property (v20 <-- m7)))
   (p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
    (rel (m9 (lex (manner)))))
   (p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
  (cq
   (p30 (object (v22 <-- m8) (v21 <-- m24) (v20 <-- m7))
    (rel (m12 (lex (similar))))))))
and
((p29 (object (v21 <-- m24) (v20 <-- m7)) (rel (m12 (lex (similar))))))
and
((p28 (cause (v19 <-- b3))
  (effect
   (p27 (object (v18 <-- b1)) (property (v22 <-- m8) (v21 <-- m24))))))
and
((p26 (class (m3 (lex (person)))) (member (v18 <-- b1))))
and
((p25 (object (v19 <-- b3)) (property (v20 <-- m7))))
and
((p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
  (rel (m9 (lex (manner))))))
and
((p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
I infer
((p30 (object (v22 <-- m8) (v21 <-- m24) (v20 <-- m7))
  (rel (m12 (lex (similar))))))

Since
((m20!
  (forall (v22 <-- m24) (v21 <-- m8) (v20 <-- m28) (v19 <-- b3)
   (v18 <-- b1) (v17 <-- b2))
  (&ant
   (p29 (object (v21 <-- m8) (v20 <-- m28))
    (rel (m12 (lex (similar)))))
   (p28 (cause (v19 <-- b3))
    (effect
     (p27 (object (v18 <-- b1))
      (property (v22 <-- m24) (v21 <-- m8)))))
   (p26 (class (m3 (lex (person)))) (member (v18 <-- b1)))
   (p25 (object (v19 <-- b3)) (property (v20 <-- m28)))
   (p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
    (rel (m9 (lex (manner)))))
   (p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
  (cq
   (p30 (object (v22 <-- m24) (v21 <-- m8) (v20 <-- m28))
    (rel (m12 (lex (similar))))))))
and
((p29 (object (v21 <-- m8) (v20 <-- m28)) (rel (m12 (lex (similar))))))
and
((p28 (cause (v19 <-- b3))
  (effect
   (p27 (object (v18 <-- b1)) (property (v22 <-- m24) (v21 <-- m8))))))
and
((p26 (class (m3 (lex (person)))) (member (v18 <-- b1))))
and
((p25 (object (v19 <-- b3)) (property (v20 <-- m28))))
and
((p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
  (rel (m9 (lex (manner))))))
and
((p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
I infer
((p30 (object (v22 <-- m24) (v21 <-- m8) (v20 <-- m28))
  (rel (m12 (lex (similar))))))

Since
((m20!
  (forall (v22 <-- m8) (v21 <-- m24) (v20 <-- m28) (v19 <-- b3)
   (v18 <-- b1) (v17 <-- b2))
  (&ant
   (p29 (object (v21 <-- m24) (v20 <-- m28))
    (rel (m12 (lex (similar)))))
```

```
   (p28 (cause (v19 <-- b3))
    (effect
     (p27 (object (v18 <-- b1))
      (property (v22 <-- m8) (v21 <-- m24)))))
   (p26 (class (m3 (lex (person)))) (member (v18 <-- b1)))
   (p25 (object (v19 <-- b3)) (property (v20 <-- m28)))
   (p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
    (rel (m9 (lex (manner)))))
   (p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
  (cq
   (p30 (object (v22 <-- m8) (v21 <-- m24) (v20 <-- m28))
    (rel (m12 (lex (similar))))))))
and
((p29 (object (v21 <-- m24) (v20 <-- m28)) (rel (m12 (lex (similar))))))
and
((p28 (cause (v19 <-- b3))
  (effect
   (p27 (object (v18 <-- b1)) (property (v22 <-- m8) (v21 <-- m24))))))
and
((p26 (class (m3 (lex (person)))) (member (v18 <-- b1))))
and
((p25 (object (v19 <-- b3)) (property (v20 <-- m28))))
and
((p24 (object (v19 <-- b3)) (possessor (v17 <-- b2))
  (rel (m9 (lex (manner))))))
and
((p23 (class (m3 (lex (person)))) (member (v17 <-- b2))))
I infer
((p30 (object (v22 <-- m8) (v21 <-- m24) (v20 <-- m28))
  (rel (m12 (lex (similar))))))

I know
((m6! (object (untalkative) (withdrawn) (reticent))
  (rel (m5 (lex (synonym))))))

I know
((m33! (object (untalkative) (withdrawn)) (rel (m5 (lex (synonym))))))

I know
((m34! (object (untalkative) (reticent)) (rel (m5 (lex (synonym))))))

I know
((m39! (object (withdrawn) (reticent)) (rel (m5 (lex (synonym))))))

(m27! (object taciturn untalkative) (rel (m16 (lex possible_synonym))))

(m27!)

 CPU time : 1.47

*
;;; Is taciturn a possible synonym of withdrawn?
(describe (deduce object "taciturn"
                  rel (find lex "possible_synonym")
                  object "withdrawn"))

I wonder if
((m53 (object (taciturn) (withdrawn))
  (rel (m16 (lex (possible_synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p16
  (object (p15 (lex (v11 <-- taciturn)))
   (p14 (lex (v10 <-- withdrawn))))
  (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p16
  (object (p15 (lex (v11 <-- withdrawn)))
   (p14 (lex (v10 <-- taciturn))))
  (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p18 (object (v13 <-- taciturn) (v12 <-- withdrawn))
  (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct
```

```
I wonder if
((p18 (object (v13 <-- withdrawn) (v12 <-- taciturn))
  (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I know
((m45! (object (m24 (lex (taciturn))) (m8 (lex (withdrawn))))
  (rel (m12 (lex (similar))))))

Since
((m17! (forall v11 v10)
  (ant
   (p16 (object (p15 (lex v11)) (p14 (lex v10)))
    (rel (m12 (lex (similar))))))
  (cq (p17 (object v11 v10) (rel (m16 (lex (possible_synonym))))))))
and
((p16
  (object (p15 (lex (v11 <-- withdrawn)))
   (p14 (lex (v10 <-- taciturn))))
  (rel (m12 (lex (similar))))))
I infer
((p17 (object (v11 <-- withdrawn) (v10 <-- taciturn))
  (rel (m16 (lex (possible_synonym))))))

Since
((m17! (forall v11 v10)
  (ant
   (p16 (object (p15 (lex v11)) (p14 (lex v10)))
    (rel (m12 (lex (similar))))))
  (cq (p17 (object v11 v10) (rel (m16 (lex (possible_synonym))))))))
and
((p16
  (object (p15 (lex (v11 <-- taciturn)))
   (p14 (lex (v10 <-- withdrawn))))
  (rel (m12 (lex (similar))))))
I infer
((p17 (object (v11 <-- taciturn) (v10 <-- withdrawn))
  (rel (m16 (lex (possible_synonym))))))

(m53! (object taciturn withdrawn) (rel (m16 (lex possible_synonym))))

(m53!)

 CPU time : 0.11

*
;;; Is taciturn a possible synonym of reticent?
(describe (deduce object "taciturn"
                  rel (find lex "possible_synonym")
                  object "reticent"))

I wonder if
((m55 (object (taciturn) (reticent))
  (rel (m16 (lex (possible_synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p16
  (object (p15 (lex (v11 <-- taciturn)))
   (p14 (lex (v10 <-- reticent))))
  (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p16
  (object (p15 (lex (v11 <-- reticent)))
   (p14 (lex (v10 <-- taciturn))))
  (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I wonder if
((p18 (object (v13 <-- taciturn) (v12 <-- reticent))
  (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct

I wonder if
((p18 (object (v13 <-- reticent) (v12 <-- taciturn))
  (rel (m5 (lex (synonym))))))
holds within the BS defined by context default-defaultct
```

```
I know
((m52! (object (m24 (lex (taciturn))) (m7 (lex (reticent))))
  (rel (m12 (lex (similar))))))

Since
((m17! (forall v11 v10)
  (ant
   (p16 (object (p15 (lex v11)) (p14 (lex v10)))
    (rel (m12 (lex (similar))))))
  (cq (p17 (object v11 v10) (rel (m16 (lex (possible_synonym))))))))
and
((p16
  (object (p15 (lex (v11 <-- reticent)))
   (p14 (lex (v10 <-- taciturn))))
  (rel (m12 (lex (similar))))))
I infer
((p17 (object (v11 <-- reticent) (v10 <-- taciturn))
  (rel (m16 (lex (possible_synonym))))))

Since
((m17! (forall v11 v10)
  (ant
   (p16 (object (p15 (lex v11)) (p14 (lex v10)))
    (rel (m12 (lex (similar))))))
  (cq (p17 (object v11 v10) (rel (m16 (lex (possible_synonym))))))))
and
((p16
  (object (p15 (lex (v11 <-- taciturn)))
   (p14 (lex (v10 <-- reticent))))
  (rel (m12 (lex (similar))))))
I infer
((p17 (object (v11 <-- taciturn) (v10 <-- reticent))
  (rel (m16 (lex (possible_synonym))))))

(m55! (object taciturn reticent) (rel (m16 (lex possible_synonym))))

(m55!)

 CPU time : 0.11

*
;;; Is taciturn a manner?
(deduce member (find lex "taciturn") class (find lex "manner"))

I wonder if
((m57 (class (m9 (lex (manner))))) (member (m24 (lex (taciturn))))))
holds within the BS defined by context default-defaultct

I wonder if
((p21 (class (v16 <-- m9)) (member v14)))
holds within the BS defined by context default-defaultct

I wonder if
((p20 (object (v15 <-- m24) v14) (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

I know
((m10! (class (m9 (lex (manner))))
  (member (m8 (lex (withdrawn))) (m7 (lex (reticent))))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m24) (v14 <-- m8))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m8)))
   (p20 (object (v15 <-- m24) (v14 <-- m8))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m24))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m8))))
and
((p20 (object (v15 <-- m24) (v14 <-- m8)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m24))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m24) (v14 <-- m7))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m7)))
   (p20 (object (v15 <-- m24) (v14 <-- m7))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m24))))))
and
```

```
((p21 (class (v16 <-- m9)) (member (v14 <-- m7))))
and
((p20 (object (v15 <-- m24) (v14 <-- m7)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m24))))

I know
((m29! (object (m28 (lex (untalkative))) (m24 (lex (taciturn))))
  (rel (m12 (lex (similar))))))

I know
((m44!
  (object (m28 (lex (untalkative))) (m24 (lex (taciturn)))
   (m8 (lex (withdrawn))))
  (rel (m12 (lex (similar))))))

I know
((m45! (object (m24 (lex (taciturn))) (m8 (lex (withdrawn))))
  (rel (m12 (lex (similar))))))

I know
((m51!
  (object (m24 (lex (taciturn))) (m8 (lex (withdrawn)))
   (m7 (lex (reticent))))
  (rel (m12 (lex (similar))))))

I know
((m52! (object (m24 (lex (taciturn))) (m7 (lex (reticent))))
  (rel (m12 (lex (similar))))))

I wonder if
((p20 (object v15 (v14 <-- m24)) (rel (m12 (lex (similar))))))
holds within the BS defined by context default-defaultct

Since
((m19! (forall (v16 <-- m9) (v15 <-- m28) (v14 <-- m7))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m7)))
   (p20 (object (v15 <-- m28) (v14 <-- m7))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m28))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m7))))
and
((p20 (object (v15 <-- m28) (v14 <-- m7)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m28))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m13) (v14 <-- m7))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m7)))
   (p20 (object (v15 <-- m13) (v14 <-- m7))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m13))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m7))))
and
((p20 (object (v15 <-- m13) (v14 <-- m7)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m13))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m8) (v14 <-- m7))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m7)))
   (p20 (object (v15 <-- m8) (v14 <-- m7))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m8))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m7))))
and
((p20 (object (v15 <-- m8) (v14 <-- m7)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m8))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m24) (v14 <-- m7))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m7)))
   (p20 (object (v15 <-- m24) (v14 <-- m7))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m24))))))
and
```

```
((p21 (class (v16 <-- m9)) (member (v14 <-- m7))))
and
((p20 (object (v15 <-- m24) (v14 <-- m7)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m24))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m7) (v14 <-- m8))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m8)))
   (p20 (object (v15 <-- m7) (v14 <-- m8))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m7))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m8))))
and
((p20 (object (v15 <-- m7) (v14 <-- m8)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m7))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m13) (v14 <-- m8))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m8)))
   (p20 (object (v15 <-- m13) (v14 <-- m8))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m13))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m8))))
and
((p20 (object (v15 <-- m13) (v14 <-- m8)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m13))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m24) (v14 <-- m8))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m8)))
   (p20 (object (v15 <-- m24) (v14 <-- m8))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m24))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m8))))
and
((p20 (object (v15 <-- m24) (v14 <-- m8)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m24))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m28) (v14 <-- m8))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m8)))
   (p20 (object (v15 <-- m28) (v14 <-- m8))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m28))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m8))))
and
((p20 (object (v15 <-- m28) (v14 <-- m8)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m28))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m7) (v14 <-- m28))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m28)))
   (p20 (object (v15 <-- m7) (v14 <-- m28))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m7))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m28))))
and
((p20 (object (v15 <-- m7) (v14 <-- m28)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m7))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m8) (v14 <-- m28))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m28)))
   (p20 (object (v15 <-- m8) (v14 <-- m28))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m8))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m28))))
and
```

```
((p20 (object (v15 <-- m8) (v14 <-- m28)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m8))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m13) (v14 <-- m28))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m28)))
   (p20 (object (v15 <-- m13) (v14 <-- m28))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m13))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m28))))
and
((p20 (object (v15 <-- m13) (v14 <-- m28)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m13))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m24) (v14 <-- m28))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m28)))
   (p20 (object (v15 <-- m24) (v14 <-- m28))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m24))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m28))))
and
((p20 (object (v15 <-- m24) (v14 <-- m28)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m24))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m28) (v14 <-- m13))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m13)))
   (p20 (object (v15 <-- m28) (v14 <-- m13))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m28))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m13))))
and
((p20 (object (v15 <-- m28) (v14 <-- m13)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m28))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m7) (v14 <-- m13))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m13)))
   (p20 (object (v15 <-- m7) (v14 <-- m13))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m7))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m13))))
and
((p20 (object (v15 <-- m7) (v14 <-- m13)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m7))))

Since
((m19! (forall (v16 <-- m9) (v15 <-- m8) (v14 <-- m13))
  (&ant (p21 (class (v16 <-- m9)) (member (v14 <-- m13)))
   (p20 (object (v15 <-- m8) (v14 <-- m13))
    (rel (m12 (lex (similar))))))
  (cq (p22 (class (v16 <-- m9)) (member (v15 <-- m8))))))
and
((p21 (class (v16 <-- m9)) (member (v14 <-- m13))))
and
((p20 (object (v15 <-- m8) (v14 <-- m13)) (rel (m12 (lex (similar))))))
I infer
((p22 (class (v16 <-- m9)) (member (v15 <-- m8))))

I know
((m29! (object (m28 (lex (untalkative))) (m24 (lex (taciturn))))
  (rel (m12 (lex (similar))))))

I know
((m44!
  (object (m28 (lex (untalkative))) (m24 (lex (taciturn)))
   (m8 (lex (withdrawn))))
  (rel (m12 (lex (similar))))))

I know
```

```
((m45! (object (m24 (lex (taciturn))) (m8 (lex (withdrawn))))
  (rel (m12 (lex (similar))))))

I know
((m51!
  (object (m24 (lex (taciturn))) (m8 (lex (withdrawn)))
   (m7 (lex (reticent))))
  (rel (m12 (lex (similar))))))

I know
((m52! (object (m24 (lex (taciturn))) (m7 (lex (reticent))))
  (rel (m12 (lex (similar))))))

(m57!)

 CPU time : 0.18

*
;;*******************************************;;
;;*****            Definition             *****;;
;;*******************************************;;

^(
--> define_adjective "taciturn");
#S(adj_info :adjective "taciturn" :unlike_props nil :class (manner)
            :type_nouns (person)
            :possible_synonyms
            (taciturn untalkative withdrawn reticent))


 CPU time : 0.00

*

End of /home/csgrad/aku2/cva_cse740/Taciturn/taciturn.demo demonstration.


 CPU time : 2.13

* (lisp)
"End of SNePS"
cl-user(4): :exit
; Exiting Lisp
pollux {~/cva_cse740/Taciturn} > ^D##exit

script done on Tue Apr 29 10:59:27 2003
```