Case Studies of SNePS

Stuart C. Shapiro
Department of Computer Science
and Center for Cognitive Science
State University of New York at Buffalo
226 Bell Hall
Buffalo, NY 14260
shapiro@cs.buffalo.edu

Abstract

SNePS, the Semantic Network Processing System, has been designed to be a system for representing the beliefs of a natural-language-using intelligent system (a "cognitive agent"). This paper expands on this motivation, discusses some of the system features that derived from this motivation, and presents four case studies of interactions with SNePS demonstrating some of these features. The features demonstrated in the case studies are: non-standard connectives; the use of recursive rules; the Unique Variable Binding Rule, that says that two variables in a rule cannot be instantiated to the same term; and discussing sentences and propositions in natural language.

1 System Description

SNePS, the Semantic Network Processing System [9, 15, 17], has been designed to be a system for representing the beliefs of a natural-language-using intelligent system (a "cognitive agent"). It has always been the intention that a SNePS-based "knowledge base" would ultimately be built, not by a programmer or knowledge engineer entering representations of knowledge in some formal language or data entry system, but by a human informing it using a natural language (NL) (generally supposed to be English), or by the system reading books or articles that had been prepared for human readers. Because of this motivation, the criteria for the development of SNePS have included: it should be able to represent anything and everything expressible in NL; it should be able to represent generic, as well as specific information; it should be able to use the generic and the specific information to reason and infer information implied by what it has been told; it cannot count on any particular order among the pieces of information it is given; it must continue to act reasonably even if the information it is given includes circular definitions, recursive rules, and inconsistent information.

More concretely, SNePS is a propositional semantic network Knowledge Representation/Reasoning (KRR) System. Like other semantic networks, the SNePS formalism consists of nodes and labelled, directed arcs. Unlike other semantic networks, propositions are exclusively represented by nodes. Nodes with no arcs emanating from them are called *base* nodes, and are taken to represent distinct entities. Nodes with arcs emanating from them are called molecular nodes, and are *structurally* defined by the network structure reachable by following arcs in the forward directions. No two nodes have identical such structures. The semantics of a node (what it represents) is structurally determined by the structure descending from it, and is *assertionally* determined by the network structure it is connected with by arcs that point into it. It is noteworthy that the structural information associated with a node cannot change, but that the assertional information associated with it can change. Nodes represent all the entities the cognitive agent being modelled (constructed) can think about, including individuals, classes, properties, events, actions, propositions, rules, *etc.* All nodes are also terms of the SNePS formalism. This means that they can be in "argument" positions relative to other nodes. *I.e.,* SNePS can represent propositions about propositions, and rules about rules without limit.

We have developed SNePS notations for a generalization of Predicate Logic. Some nodes are distinguished as variables, and there are structures for non-standard, network-oriented propositional connectives and quantifiers. Variables can range over all nodes. Thus, the language is first-order if you remember that all nodes are terms of the formalism, but looks higher order (in fact, naive), if you concentrate on the fact that nodes are used to represent propositions. SNIP, the SNePS Inference Package, can interpret rules, and forms the "inference engine" of the SNePS KRR system.

SNePS 2.1, the current version of SNePS, contains SNeBR, the SNePS Belief Revision System [2], as part of it. Thus, when using SNePS, one is always working in a "Current Context" (CC), a set of hypothesis-propositions, and in a "Current Belief Space," the set of believed propositions derived from subsets of the CC. All standard network structure is definitional, in the sense that if a node n is constructed in the network so as to represent the proposition P, that does not, in itself, make P "believed by" the system. This status is determined by a relationship between a proposition and the assumptions under which it was derived (its *origin set*). If the origin set of a proposition node is a subset of the CC, then the proposition represented by the node is currently believed by the system.

Other aspects of SNePS will be discussed as they arise in the case studies below.

2 Case Studies

2.1 Non-Standard Connectives

Since SNePS is a network formalism, my students and I invented a set of non-standard propositional connectives that take sets of arguments, and are generalizations of the standard propositional connectives and quantifiers [8, 10, 17]. The usefulness of these connectives is shown in the following interaction, which shows SNIP solving a logic puzzle taken from [18, pp. 55–58]. SNePSUL, the SNePS User's Language [17] is used for input, and each input is glossed in the comment before it with an English version. The SNePS prompt is "*". CPU time is reported in seconds. This transcript, and all transcripts in this paper, have been edited to fit on the short lines specified by the format of this paper.

In particular, this interaction contains two uses of the *numerical quantifier* and several uses of the *andor* connective. The numerical quantifier ${}_k\exists_i^j(x)R(x) : P(x)$ means that of the k individuals that satisfy R, at least i and at most j also satisfy P. It is represented by a node built with the schema (build pevb x emin i emax j etot k &ant R cq P). The andor, $\bigvee\!\!\!\bigwedge_i^j\{P_1,\ldots,P_n\}$ means that at least i and at most j of the n propositions are true. It is represented by a node built with the schema (build min i max j arg ($P_1,\ldots,P_n$)).

```
* ;; Declare the arc labels to be used.
(define member class employee job)
(MEMBER CLASS EMPLOYEE JOB)
 CPU time : 0.06

* ;; Roberta, Thelma, Steve, and Pete are people.
(describe
 (assert member (Roberta Thelma Steve Pete)
         class person))
(M1! (CLASS PERSON)
     (MEMBER PETE ROBERTA STEVE THELMA))
(M1!)
 CPU time : 0.22

* ;; Chef, guard, nurse, telephone operator,
  ;; police officer, teacher, actor, and boxer
  ;; are jobs.
(describe
 (assert
  member (chef guard nurse telephone\ operator
                police\ officer teacher actor boxer)
  class job))
 (M2! (CLASS JOB)
      (MEMBER ACTOR BOXER CHEF GUARD NURSE
              POLICE OFFICER TEACHER
              TELEPHONE OPERATOR))
(M2!)
 CPU time : 0.44

* ;; Every person has two jobs
  ;; Uses the numerical quantifier:
  ;;       Of 8 jobs exactly 2 are held by *person
(describe
 (assert
  forall $person
  ant (build member *person class person)
  cq (build pevb $job emin 2 emax 2 etot 8
            &ant (build member *job class job)
            cq (build employee *person job *job))))
(M3! (FORALL V1)
     (ANT (P1 (CLASS PERSON) (MEMBER V1)))
     (CQ (P4 (PEVB V2) (EMIN 2) (EMAX 2) (ETOT 8)
             (&ANT (P2 (CLASS JOB) (MEMBER V2)))
             (CQ (P3 (EMPLOYEE V1) (JOB V2))))))
(M3!)
 CPU time : 0.33

* ;; Every job is held by exactly one person.
(describe
 (assert
  forall *job
  ant (build member *job class job)
  cq (build pevb *person emin 1 emax 1 etot 4
      &ant (build member *person class person)
      cq (build employee *person job *job))))
(M4! (FORALL V2)
     (ANT (P2 (CLASS JOB) (MEMBER V2)))
     (CQ (P5 (PEVB V1) (EMIN 1) (EMAX 1) (ETOT 4)
             (&ANT (P1 (CLASS PERSON) (MEMBER V1)))
             (CQ (P3 (EMPLOYEE V1) (JOB V2))))))
(M4!)
 CPU time : 0.28

* ;; Roberta and Thelma are female.
(describe (assert member (Roberta Thelma)
                  class female))
(M5! (CLASS FEMALE) (MEMBER ROBERTA THELMA))
(M5!)
 CPU time : 0.11

* ;; Steve and Pete are male.
 (describe (assert member (Steve Pete) class male))
(M6! (CLASS MALE) (MEMBER PETE STEVE))
(M6!)
 CPU time : 0.12

* ;; No woman is a nurse, actor,
  ;;    or telephone operator.
  ;; Uses ANDOR to express NOR.
  ;; (At least 0, and at most 0
  ;;  of the arguments are true.)
(describe
 (assert
  forall $woman
  ant (build member *woman class female)
  cq (build min 0 max 0
            arg ((build employee *woman job nurse)
                 (build employee *woman job actor)
                 (build employee *woman
                   job telephone\ operator)))))
(M7! (FORALL V3)
     (ANT (P6 (CLASS FEMALE) (MEMBER V3)))
     (CQ (P10 (MIN 0) (MAX 0)
              (ARG (P7 (EMPLOYEE V3) (JOB NURSE))
                (P8 (EMPLOYEE V3) (JOB ACTOR))
                (P9 (EMPLOYEE V3)
                    (JOB TELEPHONE OPERATOR))))))
(M7!)
 CPU time : 0.35

* ;; No male is a chef.
(describe
 (assert forall $man
         ant (build member *man class male)
         cq (build min 0 max 0
              arg (build employee *man job chef))))
(M8! (FORALL V4)
     (ANT (P11 (CLASS MALE) (MEMBER V4)))
     (CQ (P13 (MIN 0) (MAX 0)
          (ARG (P12 (EMPLOYEE V4) (JOB CHEF))))))
(M8!)
 CPU time : 0.25

* ;; Roberta is not a boxer.
(describe
 (assert min 0 max 0
         arg (build employee Roberta job boxer)))
(M10! (MIN 0) (MAX 0)
      (ARG (M9 (EMPLOYEE ROBERTA) (JOB BOXER))))
(M10!)
 CPU time : 0.16

* ;; Pete is neither the nurse, the police officer,
  ;;      nor the teacher.
(describe
 (assert min 0 max 0
         arg ((build employee Pete job nurse)
              (build employee Pete
                     job police\ officer)
              (build employee Pete job teacher))))
```

```
(M14! (MIN 0) (MAX 0)
      (ARG (M11 (EMPLOYEE PETE) (JOB NURSE))
           (M12 (EMPLOYEE PETE)
                (JOB POLICE OFFICER))
           (M13 (EMPLOYEE PETE) (JOB TEACHER))))
(M14!)
 CPU time : 0.27

* ;; Roberta is neither the chef,
  ;; nor the police officer.
(describe
 (assert min 0 max 0
         arg ((build employee Roberta job chef)
              (build employee Roberta
                     job police\ officer))))
(M17! (MIN 0) (MAX 0)
      (ARG (M15 (EMPLOYEE ROBERTA) (JOB CHEF))
           (M16 (EMPLOYEE ROBERTA)
                (JOB POLICE OFFICER))))
(M17!)
 CPU time : 0.23

* ;; No one is both the chef and the police officer
  ;;    (because they were seen together.)
  ;; Uses ANDOR to express NAND.
(describe
 (assert
  forall *person
  ant (build member *person class person)
  cq (build min 0 max 1
            arg ((build employee *person job chef)
                 (build employee *person
                        job police\ officer)))))
(M18! (FORALL V1)
      (ANT (P1 (CLASS PERSON) (MEMBER V1)))
      (CQ (P16 (MIN 0) (MAX 1)
               (ARG (P14 (EMPLOYEE V1)
                         (JOB CHEF))
                    (P15 (EMPLOYEE V1)
                         (JOB POLICE OFFICER))))))
(M18!)
 CPU time : 0.27

* ;; Infer at least the 8 jobs
(describe (deduce (8 0) employee *person job *job))
(M10! (MIN 0) (MAX 0)
      (ARG (M9 (EMPLOYEE ROBERTA) (JOB BOXER))))
(M100! (EMPLOYEE PETE) (JOB ACTOR))
(M101! (EMPLOYEE PETE) (JOB TELEPHONE OPERATOR))
(M19! (MIN 0) (MAX 0)
      (ARG (M15 (EMPLOYEE ROBERTA) (JOB CHEF))))
(M20! (MIN 0) (MAX 0)
      (ARG (M16 (EMPLOYEE ROBERTA)
                (JOB POLICE OFFICER))))
(M21! (MIN 0) (MAX 0)
      (ARG (M11 (EMPLOYEE PETE) (JOB NURSE))))
(M22! (MIN 0) (MAX 0)
      (ARG (M12 (EMPLOYEE PETE)
                (JOB POLICE OFFICER))))
(M23! (MIN 0) (MAX 0)
      (ARG (M13 (EMPLOYEE PETE) (JOB TEACHER))))
(M50! (EMPLOYEE THELMA) (JOB CHEF))
(M55! (EMPLOYEE STEVE) (JOB POLICE OFFICER))
(M67! (MIN 0) (MAX 0)
      (ARG (M59 (EMPLOYEE ROBERTA) (JOB NURSE))))
(M68! (MIN 0) (MAX 0)
      (ARG (M60 (EMPLOYEE ROBERTA) (JOB ACTOR))))
(M69! (MIN 0) (MAX 0)
      (ARG (M61 (EMPLOYEE ROBERTA)
                (JOB TELEPHONE OPERATOR))))
(M70! (MIN 0) (MAX 0)
      (ARG (M63 (EMPLOYEE THELMA) (JOB NURSE))))
(M71! (MIN 0) (MAX 0)
      (ARG (M64 (EMPLOYEE THELMA) (JOB ACTOR))))
(M72! (MIN 0) (MAX 0)
      (ARG (M65 (EMPLOYEE THELMA)
                (JOB TELEPHONE OPERATOR))))
(M73! (EMPLOYEE ROBERTA) (JOB GUARD))
(M74! (EMPLOYEE ROBERTA) (JOB TEACHER))
(M75! (EMPLOYEE STEVE) (JOB NURSE))
(M77! (MIN 0) (MAX 0)
      (ARG (M76 (EMPLOYEE PETE) (JOB GUARD))))
(M79! (MIN 0) (MAX 0)
      (ARG (M78 (EMPLOYEE THELMA) (JOB GUARD))))
(M81! (MIN 0) (MAX 0)
      (ARG (M80 (EMPLOYEE STEVE) (JOB GUARD))))
(M83! (MIN 0) (MAX 0)
      (ARG (M82 (EMPLOYEE THELMA) (JOB TEACHER))))
(M85! (MIN 0) (MAX 0)
      (ARG (M84 (EMPLOYEE STEVE) (JOB TEACHER))))
(M88! (MIN 0) (MAX 0)
      (ARG (M48 (EMPLOYEE PETE) (JOB CHEF))))
(M89! (MIN 0) (MAX 0)
      (ARG (M54 (EMPLOYEE STEVE) (JOB CHEF))))
(M90! (MIN 0) (MAX 0)
      (ARG (M51 (EMPLOYEE THELMA)
                (JOB POLICE OFFICER))))
(M92! (MIN 0) (MAX 0)
      (ARG (M91 (EMPLOYEE STEVE) (JOB ACTOR))))
(M94! (MIN 0) (MAX 0)
      (ARG (M93 (EMPLOYEE STEVE)
                (JOB TELEPHONE OPERATOR))))
(M95! (EMPLOYEE THELMA) (JOB BOXER))
(M97! (MIN 0) (MAX 0)
      (ARG (M96 (EMPLOYEE PETE) (JOB BOXER))))
(M99! (MIN 0) (MAX 0)
      (ARG (M98 (EMPLOYEE STEVE) (JOB BOXER))))
(M10! M100! M101! M19! M20! M21! M22! M23! M50!
      M55! M67! M68! M69! M70! M71! M72! M73! M74!
      M75! M77! M79! M81! M83! M85! M88! M89! M90!
      M92! M94! M95! M97! M99!)

 CPU time : 71.48

* ;; List the 8 job-holding propositions.
(describe (findassert employee ?p job ?j))
(M100! (EMPLOYEE PETE) (JOB ACTOR))
(M101! (EMPLOYEE PETE) (JOB TELEPHONE OPERATOR))
(M50! (EMPLOYEE THELMA) (JOB CHEF))
(M55! (EMPLOYEE STEVE) (JOB POLICE OFFICER))
(M73! (EMPLOYEE ROBERTA) (JOB GUARD))
(M74! (EMPLOYEE ROBERTA) (JOB TEACHER))
(M75! (EMPLOYEE STEVE) (JOB NURSE))
(M95! (EMPLOYEE THELMA) (JOB BOXER))
(M100! M101! M50! M55! M73! M74! M75! M95!)
 CPU time : 0.45
```

2.2 Recursive Rules

If information comes from human informants, and from written material intended for human consumption, we cannot guarantee that rules will not be expressed in recursive, or even circular ways. The initial design of SNIP took into account being able to operate in the presence of recursive rules [5, 13].

The following interaction shows the use of a recursive rule. The interaction is carried out in SNePSLOG, a logic-programming-like interface to SNePS [3, 4, 14]. User input is on the lines beginning with ":", which is the SNePSLOG prompt. System output is on the other lines.

Notice that the first input is expressed as a second-order rule. This is a natural way to express many rules, and SNePS and SNePSLOG encourage it. Predicates, however, are represented as SNePS nodes, which are terms of the SNePS formalism. This rule also uses the SNePS connective *and-entailment*. $\{A_1, \ldots, A_n\}$ &⇒ $\{C_1, \ldots, C_m\}$ means that the conjunction of the antecedents implies the conjunction of the consequents.

```
: all(r) (transitive(r)
         => all(x, y, z) ({r(x,y), r(y,z)}
                          &=> {r(x,z)})).
  all(R)(TRANSITIVE(R)
        => (all(X,Y,Z)({R(X,Y),R(Y,Z)}
                        &=> {R(X,Z)})))
 CPU time : 0.23

: transitive(bigger).
  TRANSITIVE(BIGGER)
 CPU time : 0.06

: bigger(elephant, horse).
  BIGGER(ELEPHANT,HORSE)
 CPU time : 0.07

: bigger(horse, sheep).
  BIGGER(HORSE,SHEEP)
 CPU time : 0.06

: bigger(sheep, dog).
  BIGGER(SHEEP,DOG)
 CPU time : 0.07

: bigger(dog, cat).
  BIGGER(DOG,CAT)
 CPU time : 0.05

: bigger(cat, mouse).
  BIGGER(CAT,MOUSE)
 CPU time : 0.06

: bigger(mouse, spider).
  BIGGER(MOUSE,SPIDER)
 CPU time : 0.06

: bigger(spider, fly).
  BIGGER(SPIDER,FLY)
 CPU time : 0.06
```

In most system, a recursive rule will lead to an infinite loop for some query. Usually, this depends on whether the rule is left- or right-recursive, and on where the open variables are in the query. Our rule above, of course, is both left- and right-recursive, and below we demonstrate queries that have all possible combinations of constants and variables.

```
: bigger(dog, mouse)?
  BIGGER(DOG,MOUSE)
 CPU time : 40.90

: bigger(?x, sheep)?
  BIGGER(ELEPHANT,SHEEP)
  BIGGER(HORSE,SHEEP)
 CPU time : 14.13

: bigger(mouse, ?x)?
  BIGGER(MOUSE,FLY)
  BIGGER(MOUSE,SPIDER)
 CPU time : 14.72


: bigger(?x, ?y)?
  BIGGER(DOG,MOUSE)
  BIGGER(DOG,SPIDER)
  BIGGER(DOG,FLY)
  BIGGER(SHEEP,MOUSE)
  BIGGER(HORSE,MOUSE)
  BIGGER(ELEPHANT,MOUSE)
  BIGGER(MOUSE,FLY)
  BIGGER(SHEEP,SPIDER)
  BIGGER(HORSE,DOG)
  BIGGER(ELEPHANT,SHEEP)
  BIGGER(SHEEP,CAT)
  BIGGER(SHEEP,FLY)
  BIGGER(CAT,SPIDER)
  BIGGER(ELEPHANT,SPIDER)
  BIGGER(HORSE,SPIDER)
  BIGGER(ELEPHANT,CAT)
  BIGGER(ELEPHANT,FLY)
  BIGGER(CAT,FLY)
  BIGGER(HORSE,FLY)
  BIGGER(ELEPHANT,HORSE)
  BIGGER(ELEPHANT,DOG)
  BIGGER(HORSE,CAT)
  BIGGER(HORSE,SHEEP)
  BIGGER(SHEEP,DOG)
  BIGGER(DOG,CAT)
  BIGGER(CAT,MOUSE)
  BIGGER(MOUSE,SPIDER)
  BIGGER(SPIDER,FLY)
 CPU time : 7.61
```

2.3 The Unique Variable Binding Rule

An AND gate has two input ports and one output port. If the input ports are high and the output port is low, then the AND gate is faulty. A straightforward translation of the latter rule into FOPC might be

$$\begin{aligned}\forall(g,x,y,z)[&ANDGATE(g) \land INPORT(x,g)\\ &\land INPORT(y,g) \land OUTPORT(z,g)\\ &\Rightarrow [HIGH(x) \land HIGH(y) \land LOW(z)\\ &\qquad \Rightarrow FAULTY(g)]]\end{aligned}$$

However, this rule omits the antecedent literal $x \neq y$, and so could be fired by an AND gate with a high and a low input and a low output.

I argued in [11] that this situation is so common in naturally expressed rules that we must conclude that in naturally expressed rules there is an assumption that different variables must bind to different terms. (The paper cited is actually more involved than this, but space precludes a more extensive discussion in this paper.) I therefore implemented what I call the "Unique Variable Binding Rule" (UVBR) in the current version of SNIP. The interaction below demonstrates the working of the UVBR on the AND gate example. There are three other example interactions in the cited paper.

```
: all(g, x, y, z)
     ({ANDGATE(g), INPORT(x, g),
                   INPORT(y, g), OUTPORT(z, g)}
      &=> {{HIGH(x), HIGH(y), LOW(z)}
            &=> {FAULTY(g)}}).
  all(G,X,Y,Z)
     ({ANDGATE(G),INPORT(X,G),INPORT(Y,G),
                  OUTPORT(Z,G)}
      &=> {{HIGH(X),HIGH(Y),LOW(Z)}
           &=> {FAULTY(G)}})
 CPU time : 0.37
```

The gate G1 is not faulty, but would be inferred to be faulty without UVBR.

```
: ANDGATE(G1).
  ANDGATE(G1)
 CPU time : 0.07

: INPORT(G1I1, G1).
  INPORT(G1I1,G1)
 CPU time : 0.06

: INPORT(G1I2, G1).
  INPORT(G1I2,G1)
 CPU time : 0.06

: OUTPORT(G1O, G1).
  OUTPORT(G1O,G1)
 CPU time : 0.05

: HIGH(G1I1).
  HIGH(G1I1)
 CPU time : 0.06

: LOW(G1I2).
  LOW(G1I2)
 CPU time : 0.05

: LOW(G1O).
  LOW(G1O)
 CPU time : 0.06
```

Gate G2 is faulty.

```
: ANDGATE(G2)
  ANDGATE(G2)
 CPU time : 0.07

: INPORT(G2I1, G2).
  INPORT(G2I1,G2)
 CPU time : 0.08

: INPORT(G2I2, G2).
  INPORT(G2I2,G2)
 CPU time : 0.06

: OUTPORT(G2O, G2).
  OUTPORT(G2O,G2)
 CPU time : 0.07

: HIGH(G2I1).
  HIGH(G2I1)
 CPU time : 0.06

: HIGH(G2I2).
  HIGH(G2I2)
 CPU time : 0.07

: LOW(G2O).
  LOW(G2O)
 CPU time : 0.06
```

With UVBR, when we ask for faulty gates, only gate G2 is found.

```
: FAULTY(?G)?
  FAULTY(G2)
 CPU time : 3.84
```

Although UVBR seems to be correct for modelling human reasoning, it makes SNePSLOG confusing as a Logic Programming language. For example, the Logic Programmers' favorite example of append does not work:

```
: all(l1)(List(l1) => Append(|nil|, l1, l1)).
  all(L1)(LIST(L1) => APPEND(nil,L1,L1))
 CPU time : 0.14

: all(x, l1, l2, l3)
     ({Object(x), Append(l1, l2, l3)}
      &=> {Append(Cons(x, l1), l2, Cons(x, l3))}).
  all(X,L1,L2,L3)
     ({OBJECT(X),APPEND(L1,L2,L3)}
      &=> {APPEND(CONS(X,L1),L2,CONS(X,L3))})
 CPU time : 0.26

: Object(|nil|).
  OBJECT(nil)
 CPU time : 0.04

: Object(a).
  OBJECT(A)
 CPU time : 0.14

: Object(b).
  OBJECT(B)
 CPU time : 0.07

: List(Cons(a, |nil|)).
  LIST(CONS(A,nil))
 CPU time : 0.07

: List(Cons(b, |nil|)).
  LIST(CONS(B,nil))
 CPU time : 0.09

: Append(|nil|, Cons(b, |nil|), ?l)?

 CPU time : 0.27

: Append(Cons(a, |nil|), Cons(b, |nil|), ?l)?

 CPU time : 0.65
```

Neither of those two queries worked, because they required Cons(b, nil) and a variable to be bound to the same variable, l1, in the first rule. However, if we ask this question as a True/False question, it can be answered:

```
: Append(|nil|, Cons(b, |nil|), Cons(b, |nil|))?
  APPEND(nil,CONS(B,nil),CONS(B,nil))
 CPU time : 0.34
```

Also, if we add a rule that explicitly allows the second and third argument of Append to be the same, the recursive question can be answered:

```
: all(x, l1, l2)
     ({Object(x), Append(l1, l2, l2)}
      &=> {Append(Cons(x, l1), l2, Cons(x, l2))}).
  all(L2,X,L1)({OBJECT(X),APPEND(L1,L2,L2)}
            &=> {APPEND(CONS(X,L1),L2,CONS(X,L2))})
 CPU time : 0.26

: Append(Cons(a, |nil|), Cons(b, |nil|), ?l)?
  APPEND(CONS(A,nil),CONS(B,nil),
                   CONS(A,CONS(B,nil)))
 CPU time : 1.93
```

2.4 Discussing Sentences and Propositions

The interaction below shows SNePS with a particular chioce of arc labels, and a particular parsing/generation grammar for modelling a cognitive agent, CASSIE (see [12, 15]), discussing a sentence and a proposition. This makes crucial use of the ability of SNePS nodes to represent propositions and any other entities about which we can have beliefs. Here, we also use a SNePS node to represent a sentence. This also shows that it is important to retain a distinction between propositions and sentences. Soon Ae Chun and Naicong Li wrote the sections of the CASSIE grammar illustrated in this conversation, (extracted from [12], which contains several other CASSIE case studies).

In this interaction, sentences are labelled before the prompt for reference in the discussion that follows.

```
1: Bill is Lucy's brother.
 I understand that Bill is Lucy's brother

2: He is a professor.
 I understand that Bill is a professor

3: Mary is his favorite student.
 I understand that Mary is Bill's favorite student

4: Her dog is named Rover.
 I understand that Rover is Mary's dog

5: John dislikes her dog.
 I understand that John dislikes Rover

6: He said "her dog is ugly".
 I understand that John said " her dog is ugly ",
 meaning Rover is ugly

7: That John is narrowminded
   is Bill's favorite proposition.
 I understand that that John is narrowminded
 is Bill's favorite proposition

8: Mary believes Bill's favorite proposition.
 I understand that Mary believes of John
 that he is narrowminded
```

Sentences (1)–(5) introduce the characters Bill, Mary, Rover, and John. In input (6), CASSIE is informed that John uttered a particular sentence, namely "`her dog is ugly`." Just as people do, CASSIE recognizes the occurrence of a sentence being mentioned by the surrounding quote marks. The sentence is represented in SNePS as a sequential (cons-cell–like) structure of word nodes, as was described in [6, 7]. A sentence for CASSIE is just another kind of entity, so there is nothing peculiar about believing that it is the object of an act (*i.e.,* John said it). However, CASSIE does more with sentences. After representing the sentence and the proposition that John said it, CASSIE analyzes the sentence as if it had just been uttered to her. The main proposition that CASSIE understands the sentence to be expressing is also stored in SNePS, along with the proposition that the sentence expresses that proposition. All this is then output, as illustrated in output (6).

Of course, it is incorrect, in general, to analyze a mentioned sentence as if it had just been used. It should be analyzed in the context in which it was actually used, if it was used at all. This requires further work. We have constructed CASSIE to do this in order to point out a direction for further research, and to illustrate the difference between beliefs about sentences and beliefs about propositions.

Inputs (7) and (8) state beliefs about a proposition, namely that John is narrowminded. Sentence (7) gives a property of the proposition—that it is Bill's favorite proposition. Sentence (8) asserts that Mary believes it, but refers to the proposition indirectly, by the property it was given in sentence (7). It was claimed in [1] that the interaction pair (7), (8) could not be carried out in SNePS in a semantically consistent way; however, this interaction shows that it can (Cf. [16]).

References

[1] J. A. Barnden. A viewpoint distinction in the representation of propositional attitudes. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 411–415. Morgan Kaufmann, San Mateo, CA, 1986.

[2] J. P. Martins and S. C. Shapiro. A model for belief revision. *Artificial Intelligence*, 35:25–79, 1988.

[3] P. A. Matos and J. P. Martins. SNePSLOG—a logic interface to SNePS. Technical Report GIA 89/03, Grupo de Inteligencia Artificial, Instituto Superior Tecnico, Lisbon, Portugal, 1989.

[4] D. P. Mckay and J. Martins. SNePSLOG user's manual. SNeRG Technical Note 4, Department of Computer Science, SUNY at Buffalo, 1981.

[5] D. P. McKay and S. C. Shapiro. Using active connection graphs for reasoning with recursive rules. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 368–374. Morgan Kaufmann, San Mateo, CA, 1981.

[6] J. G. Neal and S. C. Shapiro. Knowledge-based parsing. In L. Bolc, editor, *Natural Language Parsing Systems*, pages 49–92. Springer-Verlag, Berlin, 1987.

[7] J. G. Neal and S. C. Shapiro. Knowledge representation for reasoning about language. *The Role of Language in Problem Solving 2*, pages 27–46, 1987.

[8] S. C. Shapiro. Numerical quantifiers and their use in reasoning with negative information. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, pages 791–796. Morgan Kaufmann, San Mateo, CA, 1979.

[9] S. C. Shapiro. The SNePS semantic network processing system. In N. V. Findler, editor, *Associative Networks: The Representation and Use of Knowledge by Computers*, pages 179–203. Academic Press, New York, 1979.

[10] S. C. Shapiro. Using non-standard connectives and quantifiers for representing deduction rules in a semantic network, 1979. Invited paper presented at Current Aspects of AI Research, a seminar held at the Electrotechnical Laboratory, Tokyo.

[11] S. C. Shapiro. Symmetric relations, intensional individuals, and variable binding. *Proceedings of the IEEE*, 74(10):1354–1363, October 1986.

[12] S. C. Shapiro. The CASSIE projects: An approach to natural language competence. In J. P. Martins and E. M. Morgado, editors, *EPIA 89: 4th Portugese Conference on Artificial Intelligence Proceedings, Lecture Notes in Artificial Intelligence 390*, pages 362–380. Springer-Verlag, Berlin, 1989.

[13] S. C. Shapiro and D. P. McKay. Inference with recursive rules. In *Proceedings of the First Annual National Conference on Artificial Intelligence*, pages 151–153. Morgan Kaufmann, San Mateo, CA, 1980.

[14] S. C. Shapiro, D. P. McKay, J. Martins, and E. Morgado. SNePSLOG: A "higher order" logic programming language. SNeRG Technical Note 8, Department of Computer Science, SUNY at Buffalo, 1981. Presented at the Workshop on Logic Programming for Intelligent Systems, R.M.S. Queen Mary, Long Beach, CA, 1981.

[15] S. C. Shapiro and W. J. Rapaport. SNePS considered as a fully intensional propositional semantic network. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier*, pages 263–315. Springer-Verlag, New York, 1987.

[16] S. C. Shapiro and W. J. Rapaport. Models and minds: Knowledge representation for natural-language competence. In R. Cummins and J. Pollock, editors, *Philosophical AI: Computational Approaches to Reasoning*. MIT Press, Cambridge, MA, 1991.

[17] S. C. Shapiro and The SNePS Implementation Group. SNePS-2.1 user's manual. SNeRG Technical Note 4, Department of Computer Science, SUNY at Buffalo, Buffalo, NY, 1989.

[18] L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning: Introduction and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1984.