Natural Language Processing based on ATN and SNePS (Report)

Nyurguyana Petrova
CSE717 Contextual Vocabulary Acquisition Seminar
Department of Linguistics, University at Buffalo
petrova3@buffalo.edu

Fall 2007

Contents

1 Debugging *brachet.demo*

1.1 Problems

At a closer inspection of Vikranth Rao's files, specifically *Falltempgram.lisp* - a grammar based on the ATN approach, we noticed a fundamental problem in the grammar. Although the grammar successfully parses the given input from *nlpdemo.demo*, at the end it fails to define a word *'brachet'*. See the output below.

```
: define brachet.
Definition of (brachet):
Possible Actions: bay, bite buttock, belong,
Possible Properties: white,
Time (sec.): 0.13
```

As the output shows the grammar completely ignores the background knowledge and built in rules, which leads to a failure when defining a word *'brachet'*. The final output simply represents the information SNePS was being told.

1.2 Debugging

Full-forward inference Rao's file did not have full-forward inference that was present in Ehrlich's file. It was added at the beginning of the *nlpdemo.demo* file, however, it did not make any change.

Input Vik Rao's file was compared with the original *'brachet'* file by Ehrlich. First the prior knowledge was compared. Everything looks the same except couple of points:

1) added by Rao to PK (these statements are not found in the original *brachet.demo*):

```
Make member move over equiv arcs
Herbivore and carnivore are antonyms
Dogs are carnivores
Sir Gawain is a knight
```

2) However, Rao's file does not state what *carnivore* is. These problems are still minor and should not affect the definition process.

Rules were also compared. There were no problems: Rao's rules look exactly the same as Ehrlich's.

The input sentences differ structure-wise but not context or meaning-wise. Ehrlich's file explicitly states the existence of some object, Vik Rao's files don't: e.g. Ehrlich's file states - *'there is a brachet'*, but Rao's file starts with *'A white brachet is nextto the hart'*.

Building objects When the output of the parser was analyzed using trace 7, some problems were revealed. Every time the parser sees an object, it builds a new object. Therefore, there are repeated cases of building one and the same object twice or even more times. It is not consistent in building new objects over and over again, some objects were built only once (e.g. *a lady*).

```
b13, b42 - hart
b14, b45 - brachet
b19 - pavilion
b20 - lady
b18, b21, b22, b23, b24, b25, b27, b28, b35, b40 - Sir Tor
b26 - dwarf
b30 - hall
b29 - King Arthur
b37, b39 - Merlin
b38 - Sir Gawain
b41 - elder
b43 - place
```

Case Frames Case frames were also analyzed. In some parts it seems that Rao was using old case frames when, for example, dealing with propername case frames. Rao's grammar always 'builds' *lex* for *propername*. The attempt to change this old case frame brought a general confusion to the parsing process in the grammar. Fundamental changes in the grammar need to be done: fixing a single occurrence of the old case frame does not work.

Rao's case frame:

```
\emph{object/propername - lex}
```

New case frame:

```
\emph{object/propername}
```

```
(pop #! ((add object ~(getr agent)
               propername (build lex ~(getr object))
               ;;;comment by Yana Petrova
               ;;;according to a new case frame it should be:
               ;;;propername ~(getr object)
               kn_cat "story")) ; "is named" + propername
     (overlap proptype 'isnamed)
     (liftr proptype))
```

When building an *'object1 rel object2'* case frame Rao introduces dashes, which do not follow the current SNePS indentation conventions.

Rao's case frame:

```
object-1/rel/object-2
```

Current case frame:

```
object1/rel/object2
```

Add vs. Assert In the file ***grammar2.lisp*** all the 'asserts' were changed to 'adds' in **s/end** category where final 'pop' functions take place.

```
;;;modified by Yana Petrova (10/26/07)
;;;'assert' was changed to 'add'
(pop #! ((add object ~(getr agent) ; "is" + adj declar. specif.
              property (build lex ~(getr object))
                    kn_cat "story"))
         (and (overlap mood 'decl) (overlap proptype 'isadj)
              (nullr generic))
         (liftr proptype))
```

As a result, although some responses were repetitive, which suggested that the parser was running into a loop, the grammar was able to parse the sentences . For the first time the parser was able to produce an answer based on the PK and built in rules. However, it could only infer some properties of *brachet*, such as *'valuable'* and *'small'*. It was still not giving the proper definition of a word *brachet*.

```
: define brachet.
Definition of (brachet):
Possible Actions: bay, bite buttock, belong,
Possible Properties: valuable, small, white,
Time (sec.): 0.12
```

The sentence *'The knight carries the brachet'* triggered the rule *'If there is a person and that person can carry something, then the thing that can be carried has the property "small"'* to fire, which in its turn produced the following response.

```
: The knight carries the brachet.
 I understand that the small white brachet is small a knight carries
 the small white brachet a knight carries the small white brachet a
 knight carries the small white brachet a knight carries the small
 white brachet a knight carries the small white brachet .
```

The sentence *'The lady says that she wants the brachet'* invokes the other rule, which states *'If something wants something then the thing that is wanted is valuable'*. As a result the parser produces the following response.

```
: The lady says that she wants the brachet.
 I understand that the lady says that the valuable small white brachet
 is valuable the lady wants the valuable small white brachet the lady
 wants the valuable small white brachet .
```

Although the parser clearly has the generation problems by giving the repetitive answers, the inferences that it is making are correct.

In the file ***grammar.lisp*** all the 'asserts' were substituted by 'adds' in **s/end** category where final 'pops' take place, as well as in other categories, such as **npr**, **n**, **n/ref**, and **np/n**. The grammar was able to parse the sentences, although it was again giving repetitive responses in some sentences. However, when the command *'define brachet'* was entered, the parser responded with an error message:

```
 : define brachet.
Error: (brachet) cannot be coerced to a string.
  [condition type: type-error]

Restart actions (select using :continue):
 0: Return to Debug Level 1 (an "abort" restart).
 1: Return to Top Level (an "abort" restart).
 2: Abort entirely from this (lisp) process.
```

Although the parser runs into an error message and is not able to define *brachet* explicitly, it correctly infers the information. Unlike *grammar2.lisp*, *grammar.lisp* has more rules being triggered. In this grammar, a sentence *'The knight carries the brachet'* causes two essential rules to fire: the previously mentioned rule *'If there is a person and that person can carry something, then the thing that can be carried has the property "small"'*, as well as a new rule *'If a member of some class has a property that is a size, then the class that it is a member of is a subclass of 'physical object"*. Due to this sentence the parser infers not only the properties of a brachet, but also the fact that a brachet is a physical object.

```
: The knight carries the brachet.
 I understand that the small white brachet is small a knight carries
 the small white brachet a knight carries the small white brachet a
 knight carries the small white brachet a knight carries the small
 white brachet a knight carries the small white brachet phys objs are
 phys objs carnivores are phys objs dogs are phys objs vertebrates are
 phys objs animals are phys objs mammals are phys objs quadrupeds are
 phys objs hounds are phys objs brachets are phys objs .
```

The sentence *'The brachet bays at Sir Tor'* helps the parser to infer that brachets are hounds using the rule that states: *'If something bays and it is a member of some class then that class is a subclass of hound'.*

```
 : The brachet bays at Sir Tor.
 I understand that brachets are hounds the small white brachet bays the
 small white brachet bays at Sir Tor at Sir Tor .
```

The sentence *'The brachet bites a hart's buttock'* produces an output where the parser infers that brachets are animals. It is based on the rule: *'If one thing bites another and the biter is a member of some class then that class is a subclass of animal'.*

```
 : The brachet bites a hart's buttock.
 I understand that brachets are animals the small white brachet bites a hart
 buttock the small white brachet bites a hart buttock .
```

2 Parsing *importunate.demo*

Vik Rao's grammar was further developed into parsing the sentences from *importunate.demo*. The lexicon *Klexicon.lisp* was also further enhanced to recognize the new lexical items. From the file *importunate.demo* the grammar learned to parse the following sentences:

Someone is named Philip.
Someone is named MissWilkinson.
MissWilkinson makes advances towards Philip.
MissWilkinson is older than Philip.
When Philip is with MissWilkinson, then Philip is discomforted.
She is importunate.

Obviously, this list is not a complete representation of *importunate.demo*, more information from the text is needed to successfully be able to define an unknown word - *importunate*.

2.1 Lexicon - *Klexicon.lisp*

New lexical items and categories were introduced to the lexicon from *importunate.demo*. The new lexical categories are comparartive (comp) and conjunctions (con), as well as a pseudo-category consequence (cq) to handle a word *'then'* in *ant-cq* constructions.

```
;;; Adjectives
("older"          ((ctgy . comp)))
("importunate"    ((ctgy . adj)))
("discomforted"   ((ctgy . adj)))

;;;Nouns
("advances"       ((ctgy . n)(num . plur)(basic . t)))

;;;Pronouns
("someone"        ((ctgy . pron)(num . sing)(case . nom)(gender m f)(human . t)))
("Someone"        ((ctgy . pron)(num . sing)(case . nom)(gender m f)(human . t)))

;;;Proper Nouns
("Philip"         ((ctgy . npr)(num . sing)(gender . m)(human . t)))
("MissWilkinson"  ((ctgy . npr)(num . sing)(gender . f)(human . t)))

;;;Verbs
("named"          ((ctgy . v)(root . "name")(tense . past)))
("makes"          ((ctgy . v)(root . "make")(tense . present)))

;;;Prepositions
("towards"        ((ctgy . prep)))
("than"           ((ctgy . prep)))
("with"           ((ctgy . prep)))

;;;Conjunctions
("when"           ((ctgy . con)))
("When"           ((ctgy . con)))

;;;Consequence
("then"           ((ctgy . cq)))
```

2.2 Grammar - *cq.lisp*

Parsing Some changes were introduced to the grammar in order to be able to parse the *'importunate'* sentences from *demo.sneps*.

In **ps** a new lexical category **conjunction (con)** was added. This category has to take care of a word *'when'*, which occurs at the beginning of a sentence in *ant-cq* constructions. When it sees a word *'when'* with a category **conjunction (con)**, the grammar sends it back to **ps** to continue parsing.

```
(ps (cat wh t
         (setr agent (* 'wh))
         (setr mood 'question)
         (liftr mood)
         (setr humanness (getf human))
         (to s/subj))

    (push np t
          (sendr mood 'decl)
          (sendr pron_case 'nom)
          (setr agent *)
          (setr mood 'decl) (liftr mood)
          (to s/subj))

    ;;;added by Yana Petrova
    (cat con t (to ps)))                ; takes care of "when"
```

In **s/is** a word *'named'*, that originally occurred at the bottom of **s/is**, was moved up immediately after a word *'a'*, in order to avoid a word *'named'* to be garden-pathed in a category **v** since *'named'* is also a verb.

```
  (s/is (wrd "a" t (to s/isnoun))

        ;;;modified by Yana Petrova
        ;;;"named" line was moved from bottom of s/is

        (wrd "named" t (to s/isnamed))
        (cat n (and (getr generic) (overlap (getf num) 'plur))
               (jump s/isnoun))
        (cat v t (setr action *) (setr pass t) (to vp/passv))
        (cat av t
               (setr mod *)(liftr mod)(to s/is))
        (cat adj t (jump s/isadj))

        ;;;added by Yana Petrova
        (cat comp t (jump s/iscomp)))
```

Although the grammar was able to parse the adjectives, it could not take care of comparative constructions when a comparative adjective is followed by a lexical item *'than'*. **s/iscomp** was

introduced to parse comparative adjectives. The grammar sets a comparative adjective to be object, assigns a *proptype 'iscomp*, which is different from *proptype 'isadj*, and continues on in **s/final**.

```
;;;added by Yana Petrova
;;;s/iscomp is introduced to take care of comparative constructions

(s/iscomp (cat comp t
               (setr object *)
               (setr proptype 'iscomp)
               (to s/final)))
```

In comparative sentences like *'MissWilkinson is older than Philip'* a word *'than'* acts as a conjunction, but in order to simplify our task in this grammar, we treated *'than'* as a preposition. It does not terribly violate the grammar of English since in some other constructions *'than'* is indeed considered as a preposition. A new preposition *'with'* was added to the existing set of prepositions in **pp**. *'With'* occurs in an *ant-cq* sentence *'When Philip is* with *MissWilkinson, then Philip is discomforted'*. A preposition *'towards'* was added to already existing word *'to'*, since usually they act similarly in most of the English sentences. *'Towards'* occurs in the sentence *'MissWilkinson makes advances towards Philip'*.

```
(pp   (wrd "into" t
           (setr into 'true) (liftr into)(to pp/end))

      ;;;added by Yana Petrova
      (wrd "than" t
           (setr rel *)(setr nec 'true)(liftr nec)(liftr rel)(to pp/end))
      (wrd "with" t
           (setr with *)(setr nec 'true)(liftr nec)(liftr with)(to pp/end))

      (wrd ("to" "towards") t        ;;;"towards" was added by Yana Petrova
           (setr to 'true)(liftr to)(to pp/end))
      (wrd "at" t
           (setr dir 'true)(liftr dir)(to pp/end))
      (wrd "in" t
           (setr pla 'true)(liftr pla)(to pp/end))
      (wrd "nextto" t
           (setr rel *)(setr nec 'true)(liftr nec)(liftr rel)(to pp/end))
      (wrd "behind" t
           (setr rel *)(setr nec 'true)(liftr nec)(liftr rel)(to pp/end)))
```

In this grammar we attempted to parse the *ant-cq* sentence for the first time. So far the grammar only had to deal with the simple sentences of the pattern S -> NP . VP, but never the complex subordinate or coordinate sentences. When the grammar is parsing the *ant-cq* sentence, at the end of the *ant* clause of a sentence it comes to **s/final**. When the grammar sees a word *'then'*, which signals a start of a *cq* clause of a sentence, it continues on **s/iscq**.

```
(s/final
```

```
        (push pp t
            (setr prepo 'prepo)(setr rel *)
            (to s/finis))

        ;;;added by Yana Petrova
        ;;;takes care of "then" to follow up to cq-construction
        (cat cq t
            (setr cq *)(to s/iscq))

        (jump s/end (overlap embedded t)) ; an embedded proposition

        (wrd "." (overlap mood 'decl) (to s/end))

        (wrd "?" (overlap mood 'question) (to s/end)))
```

At **s/iscq** a noun that comes right after *'then'* is set to the register *'agent'*. The parser continues on to **cq/subj**. **cq/subj** is equivalent to **s/subj**, the only difference being that it parses a *cq*-clause. In **cq/subj** if the root of the current verb overlaps with *'be'*, it continues on to **cq/is**. In **cq/is** if the grammar sees a category *adjective*, it 'jumps' to **cq/isadj**. **cq/isadj** is equivalent with **s/isadj**, but it parses a word in category *adjective* only from *cq*-clause. In **s/isadj** the current adjective is set to *'property'*, and the overall sentence is set to *proptype 'cq*.

```
;;;added by Yana Petrova
(s/iscq
   (push np t                          ; Noun that comes after 'then' is set to agent
          (sendr pron_case 'nom)
          (setr agent *)
          (to cq/subj)))

(cq/subj (to (cq/is) (overlap (getf root) 'be)))

(cq/is (cat adj t (jump cq/isadj)))

(cq/isadj (cat adj (nullr generic)
              (setr property *)
              (setr proptype 'cq)   ; sets to proptype 'cq that is recognized in popping
              (to s/final)))
```

At **s/end** new 'pop' functions were added. A new 'pop' was based on the case frame object1/rel/object2 (in Rao's grammar this case frame was misrepresented as object-1/rel/object-2) to build the sentences with *proptype 'isnoun*. New 'pop' function was introduced to build the comparative constructions with *proptype 'iscomp*.

```
;;;added by Yana Petrova
;;;case frame: obj1 rel obj2
(pop #! ((add object1 ~(getr agent)
              rel (build lex ~(getr object))
              object2 ~(getr indobject)
                    kn_cat "story"))
         (and (overlap mood 'decl) (overlap proptype 'iscomp)
```

```
            (nullr generic))
        (liftr proptype))
```

Another 'pop' function was added to the grammar to build the representation of *ant-cq* constructions in SNePS. The test of this 'pop' function consists of the *declarative mood* and a *proptype 'cq*.

```
;;;added by Yana Petrova
(pop #! ((add ant (build object1 ~(getr agent)
                  rel ~(getr with)
                  object2 ~(getr object))
              cq  (build object ~(getr agent)
                  property (build lex ~(getr property)))))
        (and (overlap mood 'decl)(overlap proptype 'cq)))
```

A new word *'someone'* was added to **np/art** in order to parse the sentences *'Someone is named Philip'* and *'Someone is named MissWilkinson'*. This word cannot be treated as a pronoun since in the grammar pronouns look for the previous mentioned referents and assume the objects to be the same. This assumption does not work when parsing the sentences *'Someone is named Philip'* and *'Someone is named MissWilkinson'*, because the referents are two different entities. Treating *'someone'* as a noun is also problematic, because the grammar simply overrides the first statement with the second one. As a result SNePS will only build either *'Philip'* or *'MissWilkinson'*, depending on who got introduced first. Thus, the easiest solution was whenever the grammar sees a word *'someone'* it asserts a new member of a class *'person'*, which becomes a head of a sentence.

```
(np/art (cat adj t (addr props (build lex ( ^(getr *))))
            (to np/art))

        ;;;modified by Yana Petrova (10/29/07)
        ;;;added wrd "someone"
        (wrd ("someone" "Someone") t
             (assert member #someone class (build lex "person"))
             (setr head (* 'someone))
             (setr num (getf num))
             (setr gender (getf gender))
             (to np/end))
```

At the end of **np/art** the else-statement was introduced. It is added to take care of *'than'* in comparative constructions.

```
   ;;;modified by Yana Petrova
        ;;;to take care of "than"
        (jump pp t))
```

Generation Although the grammar parses the sentences correctly and builds the right nodes, there is still a lot of work needs to be done in the generation part of the grammar. At this point, in most of the cases the grammar does not generate a response sentence, but shows a built node. For example, as a response to the parsed sentence *'Someone is named Philip'*, the grammar prints the default response phrase *'I understand that'* followed by the new built node.

However, in example *'MissWilkinson makes advances towards Philip'* the grammar responds with a sentence. There is another problem in generation: besides the fact that the grammar does not print the proper names of the entities it generates the *lex* nodes (e.g. *m12* and *m9*) as the representatives of the objects, instead of *b2* for *'Philip'* and *b3* for *'MissWilkinson'*.

```
 : Someone is named Philip.
 I understand that
(m10! (kn_cat story) (object b2) (propername (m9 (lex Philip))))

 Time (sec.): 0.0

 : Someone is named MissWilkinson.
 I understand that
(m13! (kn_cat story) (object b3) (propername (m12 (lex MissWilkinson))))

 Time (sec.): 0.0

 : MissWilkinson makes advances towards Philip.
 I understand that m12 makes advances to m9 .
 Time (sec.): 0.01

 : MissWilkinson is older than Philip.
 I understand that
(m21! (kn_cat story) (object1 b3) (object2 b2) (rel (m20 (lex older))))

 Time (sec.): 0.01

 : When Philip is with MissWilkinson then Philip is discomforted.
 I understand that m9 is discomforted .
 Time (sec.): 0.02

 : She is importunate.

 I understand that m12 is importunate .
 Time (sec.): 0.0
```

The following generation output had the same input sentences as the previous output above. The only difference in this input was an *ant-cq* sentence, which was represented in pronouns rather than proper nouns. Specifically, the *ant-cq* sentence *'When Philip is with MissWilkinson then Philip is discomforted'* was represented as *'When he is with her then he is discomforted'*. The grammar generates a response that is based on the built *ant-cq* node, but not only *cq*-clause as in the previous output. As a result, the next input sentence *'She is importunate'* gets an output, which is also represented in a built node but not in a sentence *'I understand that m12 is importunate'* as above.

```
 : Someone is named Philip.
 I understand that
(m10! (kn_cat story) (object b2) (propername (m9 (lex Philip))))
```

```
 Time (sec.): 0.01

 : Someone is named MissWilkinson.
 I understand that
(m13! (kn_cat story) (object b3) (propername (m12 (lex MissWilkinson))))

 Time (sec.): 0.01

 : MissWilkinson makes advances towards Philip.
 I understand that m12 makes advances to m9 .
 Time (sec.): 0.0

 : MissWilkinson is older than Philip.
 I understand that
(m21! (kn_cat story) (object1 b3) (object2 b2) (rel (m20 (lex older))))

 Time (sec.): 0.0

 : ;;;When Philip is with MissWilkinson then Philip is discomforted.
When he is with her then he is discomforted.
 I understand that
(m25! (ant (m22 (object1 b2) (object2 b3) (rel with)))
 (cq (m24 (object b2) (property (m23 (lex discomforted))))))

 Time (sec.): 0.01

 : She is importunate.

 I understand that
(m27! (kn_cat story) (object b3) (property (m26 (lex importunate))))

 Time (sec.): 0.0
```

SNePS network The output of the parsed sentences is represented below.

```
* (describe *nodes)

(m27! (kn_cat story) (object b3) (property (m26 (lex importunate))))
(m25! (ant (m22 (object1 b2) (object2 b3) (rel with)))
 (cq (m24 (object b2) (property (m23 (lex discomforted))))))
(m21! (kn_cat story) (object1 b3) (object2 b2) (rel (m20 (lex
older)))) (m17!
 (act (m16 (action (m15 (lex make))) (object (m14 (lex advances)))))
 (agent b3) (kn_cat story) (to b2))
(m13! (kn_cat story) (object b3) (propername (m12 (lex
MissWilkinson)))) (m11! (class (m7 (lex person))) (member b3)) (m10!
(kn_cat story) (object b2) (propername (m9 (lex Philip)))) (m8!
(class (m7)) (member b2)) (m6! (do (m5 (action (m1 (lex defnoun)))
```

```
(object1 brachet)))
 (whenever (m4 (act (m3 (action (m2 (lex definingnoun))))))))

(m27! m26 importunate m25! m24 m23 discomforted m22 with m21! m20
older
 m17! m16 m15 make m14 advances m13! m12 MissWilkinson m11! b3 m10!
 story m9 Philip m8! m7 person b2 m6! m5 brachet m4 m3 m2 definingnoun
 m1 defnoun)

 CPU time : 0.01
```

3 Future Study

brachet.demo Although the major problem of the grammar *'Falltempgrammar.lisp'* is identified, there is still more work to be done in the generation part. Due to the substitutions of 'asserts' by 'adds' the grammar is able to make a correct inference based on the PK and the rules, however, it fails when asked to define the word *brachet* (see section 1.2 *Debugging (Add vs. Assert)* for details). The thorough analysis of the function defining a word *brachet*, as well as the generation part is necessary. Misrepresented or old case frames should be substituted by the correct current ones (see section 1.2 *Debugging (Case Frames)* for details). One also needs to figure out why the grammar builds one and the same object several times (e.g. in case of *'Sir Tor'*, which is built at least ten times)(see section 1.2 *Debugging (Building objects)* for details).

importunate.demo The grammar *cq.lisp* is still at the initial stage of development. Although it is able to recognize and parse several sentences from *demo.sneps*, still not all sentences are parsed yet. After parsing the input sentences, the parser builds correct network, however, it still has a problem generating response sentences. The grammar does not have a PK and rules.

Acknowledgments

I would like to thank Dr. Rapaport for his guidance and advice throughout the process of this project.

Appendix I

This is the running output demo of the file *nlpdemo.sneps* based on the grammar *grammar2.lisp* output.

```
^^( --> parse )

 ATN parser initialization...

 Trace level = 0.

 Beginning at state 's'.


 Input sentences in normal English orthographic convention.
 Sentences may go beyond a line by having a space followed by a <CR>
 To exit the parser, write ^end.


 : A hart runs.
 I understand that the hart runs the hart runs .
 Time (sec.): 0.02

 : A white brachet is nextto the hart.
 I understand that the white brachet is nextto the hart .
 Time (sec.): 0.02

 : A black hound runs.
 I understand that the black hound runs the black hound runs .
 Time (sec.): 0.27

 : The hound is behind the hart.
 I understand that the black hound is behind the hart .
 Time (sec.): 0.03

 : The hart is nextto Round Table.
 I understand that the hart is nextto Round Table .
 Time (sec.): 0.01

 : A knight arises.
 I understand that a knight arises .
 Time (sec.): 0.06

 : The knight picksup the brachet.
 I understand that a knight picksups the white brachet .
 Time (sec.): 0.04

 : The knight carries the brachet.
 I understand that the small white brachet is small a knight carries
 the small white brachet a knight carries the small white brachet a
```

```
knight carries the small white brachet a knight carries the small
white brachet a knight carries the small white brachet .
Time (sec.): 0.05

: Sir Tor goes to a pavilion.
I understand that Sir Tor goes to the pavilion .
Time (sec.): 0.01

: A lady says that the knight takes the brachet.
I understand that the lady says that a knight takes the
small white brachet .
Time (sec.): 0.03

: The lady sleeps in the pavilion.
I understand that the lady sleeps in the pavilion .
Time (sec.): 0.04

: Sir Tor finds the lady in the pavilion.
I understand that Sir Tor finds the lady in the pavilion .
Time (sec.): 0.02

: Sir Tor finds the brachet in the pavilion.
I understand that Sir Tor finds the small white brachet
in the pavilion .
Time (sec.): 0.02

: The brachet bays at Sir Tor.
I understand that the small white brachet bays the small white
brachet bays at Sir Tor .
Time (sec.): 0.01

: Sir Tor spies the brachet.
I understand that Sir Tor spies the small white brachet .
Time (sec.): 0.02

: Sir Tor takes the brachet.
I understand that Sir Tor takes the small white brachet .
Time (sec.): 0.01

: A dwarf says that he knows that Sir Tor seeks the knight.
I understand that the dwarf says that the dwarf knows that Sir Tor
seeks a knight .
Time (sec.): 0.04

: Sir Tor gives the brachet to the dwarf.
I understand that Sir Tor gives the small white brachet to the dwarf .
Time (sec.): 0.03

: The hart runs into King Arthur's hall.
```

```
I understand that the hart runs into King Arthur's hall into
King Arthur's hall into King Arthur's hall .
Time (sec.): 0.02

: The brachet bites a hart's buttock.
I understand that the small white brachet bites a hart buttock
the small white brachet bites a hart buttock .
Time (sec.): 0.04

: The knight mounts a horse.
I understand that a knight mounts the horse .
Time (sec.): 0.07

: The knight rides the horse.
I understand that a knight rides the horse .
Time (sec.): 0.03

: Sir Tor mounts a horse.
I understand that Sir Tor mounts a horse .
Time (sec.): 0.03

: The lady says that the brachet belongs to her.
I understand that the lady says that the small white brachet
belongs to the lady .
Time (sec.): 0.02

: The lady says that she wants the brachet.
I understand that the lady says that the valuable small white brachet
is valuable the lady wants the valuable small white brachet the lady
wants the valuable small white brachet .
Time (sec.): 0.04

: Merlin says that Sir Gawain must bring the hart to the hall.
I understand that Merlin says that Sir Gawain must bring a hart to
King Arthur's hall .
Time (sec.): 0.05

: Merlin says that Sir Tor must bring the brachet to the hall.
I understand that Merlin says that Sir Tor must bring the valuable
small white brachet to King Arthur's hall .
Time (sec.): 0.03

: An elder says that a white hart comes to a place.
I understand that the elder says that a white hart comes to the place .
Time (sec.): 0.05

: The elder says that a hound chases the hart.
I understand that the elder says that a hound runs .
Time (sec.): 0.1
```

```
 : The elder says that a white brachet is nextto the hart.
 I understand that the elder says .
 Time (sec.): 0.02

 :

End of /home/lingrad/petrova3/CSE717/nlpdemo.sneps demonstration.

 : define brachet.
 Definition of (brachet):
 Possible Actions: bay, bite buttock, belong,
 Possible Properties: valuable, small, white,
 Time (sec.): 0.12
```

Appendix II

This is the running output demo of the file *nlpdemo.sneps* based on the grammar *grammar.lisp*.

```
^^( --> parse )

 ATN parser initialization...

 Trace level = 0.

 Beginning at state 's'.


 Input sentences in normal English orthographic convention.
 Sentences may go beyond a line by having a space followed by a <CR>
 To exit the parser, write ^end.


 : A hart runs.
 I understand that the hart runs the hart runs .
 Time (sec.): 0.02

 : A white brachet is nextto the hart.
 I understand that the white brachet is nextto the hart .
 Time (sec.): 0.05

 : A black hound runs.
 I understand that the black hound runs the black hound runs .
 Time (sec.): 0.56

 : The hound is behind the hart.
 I understand that the black hound is behind the hart .
 Time (sec.): 0.03

 : The hart is nextto Round Table.
 I understand that the hart is nextto Round Table .
 Time (sec.): 0.02

 : A knight arises.
 I understand that a knight arises .
 Time (sec.): 0.1

 : The knight picksup the brachet.
 I understand that a knight picksups the white brachet .
 Time (sec.): 0.06

 : The knight carries the brachet.
 I understand that the small white brachet is small a knight carries
 the small white brachet a knight carries the small white brachet a
 knight carries the small white brachet a knight carries the small
```

```
white brachet a knight carries the small white brachet phys objs are
phys objs carnivores are phys objs dogs are phys objs vertebrates are
phys objs animals are phys objs mammals are phys objs quadrupeds are
phys objs hounds are phys objs brachets are phys objs .
Time (sec.): 0.08

: Sir Tor goes to a pavilion.
I understand that Sir Tor goes to the pavilion .
Time (sec.): 0.03

: A lady says that the knight takes the brachet.
I understand that the lady says that a knight takes the small white brachet .
Time (sec.): 0.04

: The lady sleeps in the pavilion.
I understand that the lady sleeps in the pavilion .
Time (sec.): 0.05

: Sir Tor finds the lady in the pavilion.
I understand that Sir Tor finds the lady in the pavilion .
Time (sec.): 0.03

: Sir Tor finds the brachet in the pavilion.
I understand that Sir Tor finds the small white brachet in the pavilion .
Time (sec.): 0.03

: The brachet bays at Sir Tor.
I understand that brachets are hounds the small white brachet bays the
small white brachet bays at Sir Tor at Sir Tor .
Time (sec.): 0.3

: Sir Tor spies the brachet.
I understand that Sir Tor spies the small white brachet .
Time (sec.): 0.03

: Sir Tor takes the brachet.
I understand that Sir Tor takes the small white brachet .
Time (sec.): 0.02

: A dwarf says that he knows that Sir Tor seeks the knight.
I understand that the dwarf says that the dwarf knows that Sir Tor seeks
a knight .
Time (sec.): 0.06

: Sir Tor gives the brachet to the dwarf.
I understand that Sir Tor gives the small white brachet to the dwarf .
Time (sec.): 0.06

: The hart runs into King Arthur's hall.
```

```
I understand that the hart runs into King Arthur's hall into King Arthur's
hall into King Arthur's hall .
Time (sec.): 0.04

: The brachet bites a hart's buttock.
I understand that brachets are animals the small white brachet bites a hart
buttock the small white brachet bites a hart buttock .
Time (sec.): 0.1

: The knight mounts a horse.
I understand that a knight mounts the horse .
Time (sec.): 1.01

: The knight rides the horse.
I understand that a knight rides the horse .
Time (sec.): 0.06

: Sir Tor mounts a horse.
I understand that Sir Tor mounts a horse .
Time (sec.): 0.69

: The lady says that the brachet belongs to her.
I understand that the lady says that the small white brachet belongs
to the lady .
Time (sec.): 0.04

: The lady says that she wants the brachet.
I understand that the lady says that the valuable small white brachet
is valuable the lady wants the valuable small white brachet the lady
wants the valuable small white brachet .
Time (sec.): 0.07

: Merlin says that Sir Gawain must bring the hart to the hall.
I understand that Merlin says that Sir Gawain must bring a hart to
King Arthur's hall .
Time (sec.): 0.1

: Merlin says that Sir Tor must bring the brachet to the hall.
I understand that Merlin says that Sir Tor must bring the valuable
small white brachet to King Arthur's hall .
Time (sec.): 0.06

: An elder says that a white hart comes to a place.
I understand that the old elder says that a white hart comes to the place .
Time (sec.): 0.81

: The elder says that a hound chases the hart.
I understand that the old elder says .
Time (sec.): 0.95
```

```
 : The elder says that a white brachet is nextto the hart.
 I understand that the old elder says .
 Time (sec.): 1.16

 :

End of /home/lingrad/petrova3/CSE717/nlpdemo.sneps demonstration.

 : define brachet.
Error: (brachet) cannot be coerced to a string.
  [condition type: type-error]

Restart actions (select using :continue):
 0: Return to Debug Level 1 (an "abort" restart).
 1: Return to Top Level (an "abort" restart).
 2: Abort entirely from this (lisp) process.
```

Appendix III

This is the running output demo of the file *demo.sneps* based on the grammar *cq.lisp*.

```
 : Someone is named Philip.
 I understand that
(m10! (kn_cat story) (object b2) (propername (m9 (lex Philip))))

 Time (sec.): 0.01

 : Someone is named MissWilkinson.
 I understand that
(m13! (kn_cat story) (object b3) (propername (m12 (lex
MissWilkinson))))

 Time (sec.): 0.01

 : MissWilkinson makes advances towards Philip.
 I understand that m12 makes advances to m9 .
 Time (sec.): 0.0

 : MissWilkinson is older than Philip.
 I understand that
(m21! (kn_cat story) (object1 b3) (object2 b2) (rel (m20 (lex
older))))

 Time (sec.): 0.0

 : ;;;When Philip is with MissWilkinson then Philip is discomforted.
When he is with her then he is discomforted.
 I understand that
(m25! (ant (m22 (object1 b2) (object2 b3) (rel with)))
 (cq (m24 (object b2) (property (m23 (lex discomforted))))))

 Time (sec.): 0.01

 : She is importunate.

 I understand that
(m27! (kn_cat story) (object b3) (property (m26 (lex importunate))))

 Time (sec.): 0.0

* (describe *nodes)

(m27! (kn_cat story) (object b3) (property (m26 (lex importunate))))
(m25! (ant (m22 (object1 b2) (object2 b3) (rel with)))
 (cq (m24 (object b2) (property (m23 (lex discomforted))))))
(m21! (kn_cat story) (object1 b3) (object2 b2) (rel (m20 (lex
older)))) (m17!
```

```
 (act (m16 (action (m15 (lex make))) (object (m14 (lex advances)))))
 (agent b3) (kn_cat story) (to b2))
(m13! (kn_cat story) (object b3) (propername (m12 (lex
MissWilkinson)))) (m11! (class (m7 (lex person))) (member b3)) (m10!
(kn_cat story) (object b2) (propername (m9 (lex Philip)))) (m8!
(class (m7)) (member b2)) (m6! (do (m5 (action (m1 (lex defnoun)))
(object1 brachet)))
 (whenever (m4 (act (m3 (action (m2 (lex defininggnoun))))))))

(m27! m26 importunate m25! m24 m23 discomforted m22 with m21! m20
older
 m17! m16 m15 make m14 advances m13! m12 MissWilkinson m11! b3 m10!
 story m9 Philip m8! m7 person b2 m6! m5 brachet m4 m3 m2 defininggnoun
 m1 defnoun)

 CPU time : 0.01
```