

Homework Assignment #2: Sentence Realization**Due date: March 13, 2008**

This assignment illustrates ‘older-style’ computational linguistics / NLP: representation and non-learning rule-based processing. In the first part you define a knowledge representation scheme and create instance frames representing sentences. In the remainder you build a sentence generator that ultimately can convert these frames into English sentences.

Please email all results by the due date to hovy@isi.edu.

Part 1 (35 points)

As you know, representing meaning is tricky. Design a system for representing the meanings of simple sentences. Try to be parsimonious if you can: create as few basic primitives ‘atoms’ of meaning as possible and then try to compose together these atoms into larger composites. Create representation frames/schemas for the following types of meaning:

- **Objects** (e.g., table, chair, wheel, car, human, child, cat, mountain, idea, justice, happiness). Use the relations *:type*, *:color*, *:age*, *:size*, *:price*...and the quasi-syntactic slot *:identifiable*.
- **Events and actions** (e.g., eating, playing, singing, swimming, thinking, giving, taking, selling). Use the relations *:type*, *:agent*, *:patient*, *:message*, *:desire*, *:instrument*, *:manner*, *:source*, *:destination*, *:beneficiary*, *:time*, *:location*, the pragmatic slots *:speechact* (for Assertion, Question, and Order) and *:polarity* (for Pos and Neg), and the inter-event connectives *:temp-after*, *:temp-during*, *:condition*, *:cause*...
- **Qualities** (e.g., red, happy, fast, wild, relaxed, large, small, old). Use the relations *:type*, *:entity*, *:attribute*, *:value*, *:degree*, *:polarity*, as in:
The chair is not very heavy
(C0 :type QUALITY :entity (C1 :type CHAIR) :attribute (C2 :type WEIGHT :value HEAVY :degree VERY :polarity NEG))
- **States** (e.g., having a headache, having a stomach ache, being happy, being very happy, wanting). Use the relations *:type*, *:experiencer*, *:attribute*, *:degree*, *:polarity*, as in:
John had a severe headache
(K0 :type STATE :experiencer JOHN :attribute (K1 :type HEADACHE :degree +7) :time PAST)
- **State changes** (e.g., healing from the flu, becoming happy, and then very happy, growing tall). Use the relations *:type*, *:experiencer*, *:attribute*, *:old-degree*, *:new-degree*, *:old-value*, *:new-value*, *:polarity*

An example representation frame is:

```
(X15  :type EAT
      :actor (X16
              :type MAN
              :size LARGE
              :identifiable T)
      :object (X17
```

:type PEACH
:size SMALL
:color RED
:identifiable T
:time PAST
:speechact ASSERTION
:polarity POS)

Look in the class handout for other examples, and choose how ‘deep’ (parsimonious) you want your representation primitives to be.

Create (for 10 points) on paper a small ontology of the various representational classes, and list for each class the entities/events you need to use for your sentences:

TOP

OBJECT

PHYSICALOBJECT (e.g., CAR, PEACH)

NONPHYSICALOBJECT (e.g., IDEA, DEMOCRACY)

EVENT

QUALITY

STATE

STATE-CHANGE

SPEECHACT

RELATION

EVENTROLE

OBJECTRELATION

QUALITYRELATION

STATEANDSTATECHANGERELATION

To test your ideas, represent the following sentences:

John bought an expensive book about medicine from Mary (2 points)

John told Mary he wanted to marry her (4 points: what do you do with the inner sentence?)

John and Mary will marry in July (4 points: what do you do with the “and”?)

John has a stomach ache (2 points)

John’s headache gets worse while his stomach ache gets better (2 points)

Reading the book made John feel better (2 points)

The chair is not very heavy (4 points. Note that you could logically place the *:polarity* in two places, at the clause level or inside the weight proposition itself. Does it matter? What do the two alternatives mean, and if your sentence realizer *had* to say the two differently, what would you make it do? 5 extra points if you make your sentence realizer able to say the two differently.)

The price of the book gave John a headache (5 points. Note it is not the price itself, but John’s perception of the price, that has this effect. 10 extra points if your generator is able to generate this sentence from a representation deep enough to capture this ‘hidden embedding’.)

Write up and hand in the ontology and your representation(s) of these 8 sentences.

Part 2 (15 points)

Write a random sentence generator.

As explained in class, to make a realizer, we have the system produce outputs along the way, thereby changing the RTN into an Augmented Transition Network (an implementation is described in (Simmons & Slocum, 1972)). Whenever it crosses over an arc, the system has to generate the constituent stated on the arc. If the constituent is simple (just a word), the system simply outputs the word and continues. If it is complex (that is, another constituent, which itself may be a sequence of states), the system has to temporarily suspend the current ATN, push into the new constituent's ATN, and realize that. When it has successfully completed the new one, the system pops back to the old ATN, back to the sequence that initiated the push, and continues.

Algorithm: Use a stack to hold unprocessed material.

START: Place "S" on the stack

GEN:

1. pop the next item on the stack; call it I
2. if I is a lexical item, print it out
 else call EXPAND with I and push the results onto the top of the stack
3. unless the stack is empty, go to 1

EXPAND (X):

1. if X is a word class in the lexicon, return a word randomly selected from that class
2. if X is the head of a grammar rule, return an expansion randomly chosen from that rule
3. else there's an error

Doing this, you can generate grammatical English sentences, starting by loading "S" onto the stack, or noun phrases, starting by loading "NP" on the stack.

Create a grammar (with at least 10 rules) and a lexicon (with at least 5 word classes, each non-closed-class containing at least 10 words), and implement an algorithm to generate syntactically correct but possibly meaningless sentences. For this:

- lexicon: 3 points
- grammar: 3 points
- algorithm: 5 points
- run of system (with at least 2 traces showing interesting intermediate processing points): 4 points

Hand in this generator: mail the code and the traces.

Part 3 (50 points)

Extend your generator to accept the shallow semantic input representations you created in part 1, and to generate the appropriate sentences. As discussed in class, you will have to augment the grammar rule representation to specify which portion of the input representation it addresses. You might also have to build a selection mechanism to decide which rule applies to the input, or else a backtracking mechanism to recover from incompletely expressed inputs.

Take as input a single sentence at a time.

In order to generate meaningful sentences, you can still use ATN traversal system outlined in class and developed in Part 2, but it has to be changed a little. While you still follow the arcs and output words, two things change:

- arc choice is no longer random,
- lexical choice is no longer random.

Both these decisions must somehow be guided by the input. Each arc will express some portion of the input. As you travel ‘down’ through the grammar, the portions of the input you consume get smaller and smaller, bottoming out in words. To do this, extend the grammar’s rules with links to the input:

```
1. S[head]      →   NP[head:agent] VP[head]
                →   AUX[head:aux] NP[head:agent] VP[head]
                →   NP[head:agent] AUX[head:aux] VP[head]
```

and so on, for the other rules. Also extend the lexicon itself:

```
N[man]         →   singular: man
                →   plural: men
```

Now extend the basic algorithm:

Input: above shallow or deep semantics

Grammar: above rules, as extended

Lexicon: as above

Algorithm:

1. get the top of the stack, bind its head to the head of the input, and push the result onto stack
- 2a. pop the top of the stack
 - b. if it is a lexical item, print it
 - c. else: find an appropriate expanded phrase for the top from the phrasal grammar
(if there are alternatives, execute some selection function)
for each portion of this phrase, find the approp. portion of the input, as specified
in the portion (this may be a lexeme) and bind it into the phrase
perform any side effects (number agreement)
push the list of bound portions back onto the stack
3. unless stack is empty, go to 2a

You will note that some things that look simple are quite difficult to generate. It’s ok to generate a paraphrase—a sentence that means the same but says it in a different way—but please explain what the trouble was in such cases.

Hand in (by sending email) the following:

- the realizer code (20 points),
- the new grammar (5 points),
- the lexicon (4 points),
- the outputs generated for these inputs (and more, if you wish) (2 points each for the 8 sentences),
- a trace of the system on at least two sentences, showing interesting processing stages (5 points),
- a discussion of what you found problematic and what not, and why (extra credit).

Extra credit (up to 15 points) if your generator can handle the last two sentences appropriately as mentioned in Part 1).

You may note that your system could say the same input in various ways; at the very least, you might have different words for the same thing, like *say* and *state* or *car* and *auto*. Why would you choose one over the other? How would you represent the information required to make the choice? You might also be worried by that fact that it will be hard to make the grammar more powerful without building in some kind of backtracking. For example, if your grammar had the two NP rules

NP → PRONOUN

NP → DET ADJ* N

Then every :agent will always become “he” or “she” or “it”, and you will never be able to say a full NP! Ideally, you’d build a backtracking mechanism into the loop, which would make sure that the whole input frame was actually said, or if not, would cause another grammar rule to be chosen as appropriate. If you can write your grammar simply enough to avoid this problem, good for you. But it’s nicer (and you get even more extra credit, up to another 10 points) if you do implement some kind of backtracking or lookahead capability.

This assignment has two main ideas: to teach you how to write a generator, including the grammar and lexicon, and to prompt you to *be creative!* The more you think about the questions, and the more you describe what you did and why you did it, the better.