Massachusetts Institute of Technology

6.038 Artificial Intelligence in Practice

Assignment 5: Thursday, March 10, 1997

1 Objective

The objective of this assignment is to give you experience in understanding and developing grammars for Natural Language Understanding Systems. The code you'll be using was developed by James Allen of the University of Rochester in support of his book *Natuaral Language Understanding* published by Benjamin Cummings.

2 Files

You can find the Lisp files for the parser in `/mit/6.038/parser`. The parser runs in Lucid Lisp (I recommend that you run the following commands in a shell under Emacs). Lucid Lisp only runs on Athena Suns.

```
add lucidcl
lisp
(lcl:cd "/mit/6.038/parser")
(load "loader")
(load "/mit/6.038/assign-5/grammar")
(load "/mit/6.038/assign-5/tests")
```

Now you're ready to call the parser.

The other files mentioned below can be found in `/mit/6.038/assign-5/`.

3 Some tips on using the parser

You should read `/mit/6.038/parser/README` for some general information on using the parser.

The basic parser call is `(bu-parse '(Jack saw Jill))`. I have also provided you with the small function `(run-test '(Jack saw Jill))` which calls the parser and pretty-prints the semantics so that it is more readable. The function `(run-all-tests)` runs through a fixed set of test sentences defined in the grammar file. During initial tests, you might want to minimize the amount of trace information printed by doing `(traceoff)`.

To minimize the size of the lexicon, the grammar contains a set of rules for forming the tensed forms of regular verbs and the plural of regular nouns. This is done by including suffixes in the input (beginning with +). For example, `(Jill was see +en by Jack)` or `(The dog +s cry +ed)` or `(Jack decide +s on the car)` or `(Jack +ss sister walk +ed)`. In general, any natural language system would have a morphology front end that would process unknown words and break them down into a root word and the relevant suffixes.

Note that the printout from running the parser is (by default) all interpretations of the input. When there is no complete interpretation that spans the whole input, the results are some of the pieces that were found. In general, we are interested in getting sentence level interpretations, these print as: Snnn: <S (... (feature value) ...)>.

You should make a copy of the grammar file ("/mit/6.038/assign-5/grammar"), make any changes you need and reload it when you are ready to try parsing.

4 Assignment

The tests file has a list of example sentences that work (more or less) correctly. You can run these by executing (run-all-tests). Study these carefully in connection with the grammar so that you understand how the grammar works, both in producing a reasonable syntax and reasonable logical form (semantics). You might want to try a few sample sentences of your own.

4.1 Part 1 (easier)

Explain any ambiguous interpretations when running (run-all-tests). That is, explain how the grammar produces the multiple interpretations. Indicate whether you think these ambiguities are inherent or are deficiencies in the grammar.

4.2 Part 2 (harder)

That file also includes a number of examples that do not work at all — (run-all-tests :tests *assign-5-tests*). You should modify the grammar and the lexicon to get the examples to work; the last two are optional. In some cases, it may simply be a missing entry in the lexicon. In other cases, you will need to modify some rules or define new grammar rules. Please try to make the new rules as general as possible, that is, so that they will work for a class of examples not just the specific ones given. Keep in mind when adding rules that you don't only want to accept the correct sentences, you want to reject incorrect sentences.

Hand in only your modified rules and lexical entries as well as the result of running (run-all-tests :tests *assign-5-tests*) (with tracing off).

Explain why the last two examples in *assign-5-tests* are harder than the similar ones just above.