11-721: Grammars and Lexicons

Grammar Writing

Teruko Mitamura

Language Technologies Institute

Carnegie Mellon University

1. How To Run The Parser

1.1 Grammar file:

- A grammar file has to have the extension **.gra** (e.g. test.gra).

1.2 How to Run the Generalized LR Parser/Compiler from Emacs from Lab machine

1. Login to one of the PC machines using andrew login/password.
2. Telnet to andrew unix machine using andrew login/password.
3. In xterm, login to **bert.speech.cs.cmu.edu**, using CS login/password.
    - Type: ssh -l <cs user name> <machine name>
4. Start Emacs in the LTI machine. Type: **emacs -nw**
5. Run Lisp inside an Emacs buffer
   Esc-x inferior-lisp
   OR
   Ctr-u Meta-x inferior-lisp
   then type: gcl <cr> for GNU common lisp on PC
6. Load Parser
   (load "/afs/cs/project/cmt-55/lti/Lab/Modules/GNL-721/parser/v8-4-rt.lisp")
7. Set the working directory
   e.g. (system:chdir "/afs/cs/user/your-name/your-grammar-directory") for GNU common lisp
   e.g. (setf (working-directory) "<your-directory>") for Lucid lisp.
8. Put Emacs into the background by typing **^z**.
9. Copy a grammar exercise file from:
   /afs/cs/project/cmt-55/lti/Lab/Modules/GNL-721/2007/
   to your own working directory.
10. Get back to Emacs by typing **fg**.
11. Edit a grammar. ^x2 will split the window (one for lisp, one for grammar editing). ^xo will move the cursor from one window to another.
    GNU Emacs manual:
    http://www.gnu.org/manual/emacs-20.3/emacs.html
12. Save the grammar that you updated: ^X^S in Emacs.
13. Compile a grammar (**Do not type the extension .gra**)
    (compgra "file-name")
    e.g. (compgra "test")
    compgra compiles a grammar and load it into lisp automatically.
14. If you enter the lisp debugger, you type **:q** to get back to the top level.

15. Character-Based Parsing
    - Make sure that the lexicon is written for character-based parsing.
    - Parsing a sentence
      (p "sentence string")
      e.g. (p "a bird flies")
    - Parsing sentences
      (p* list-of-strings)
      This function parses all the sentences in the list. It is useful when you test a grammar with a set of test sentences.
      a. Write test sentences into a file and create the test file: "test-file.lisp".):

```
e.g. (setq test-sentences '("A bird flies"
                            "Birds fly"
                            "The bird flies"))
```

      b. Load the test file:
         e.g. (load "test-file.lisp")
      c. Parse the sentences:
         e.g. (p* test-sentences)
16. Word-Based Parsing
    - Make sure that the lexicon is written for word-based parsing.
    - Parsing a sentence
      (parse-list list-of-symbols $)
      e.g. (parse-list '(a bird flies $))
17. How to Quit Lisp
    - Quit GNU Common Lisp: type Ctr-d
    - (Or quit Lisp: type (quit)
18. Make sure to logout from both CS machine and Andrew machine.
19. Make sure to logout from Windows.

1.3 Loading the Compiled Grammar

When the grammar was already compiled and it has not been changed since it was compiled, you can use the **loadgra** function to load the compiled grammar. **Do not type the extension .gra.**

- **(loadgra "file-name")**

1.4 Creating "init.lisp" File

It would be helpful to create a "init.lisp" file for loading Lisp, so that you don't need to type the parser location, etc. each time. In a file (called init.lisp), you write:

```
;; This is for loading the parser.
(load "/afs/cs.cmu.edu/project/cmt-55/lti/Lab/Modules/GNL-721/parser/v8-4-rt.lis p")
;; This is for setting the working directory.
(system:chdir "/afs/cs.cmu.edu/usr/<your login>/11-721/)
```

When you load the parser, you type **(load "init.lisp"**) and the files will be loaded into the Lisp buffer.

1.5 How to Debug the Grammar:

1. **Dmode** function shows you the rule application during the parsing process.
    - Enter **(dmode 2)** to see the rules applied as well as rules that are killed.
    - Enter **(dmode 0)** to turn off the dmode function.
2. **Trace** function provided by the LISP shows you the input and output of the function associated with a particular rule.
    - For example, enter **(trace testf-4)** to see the value passed to the function and the value returned by the function.
    - To turn off tracing, enter **(untrace)** to turn off tracing all the functions, or you can specify a particular function by entering (untrace name-of-function). For example, enter **(untrace testf-4)**.
3. Other useful functions after parsing:
    - **(disp-tree)** function displays a tree structure obtained. However, this function can only display one ambiguity.
    - If you would like to see a subtree, it accepts an optional argument **n**, where **n** is a node number. For example, enter **(disp-tree 7)**.
    - To obtain node numbers, you can enter **(disp-nodes)** or use **(dmode 2)**.
    - **disp-node-value** function displays the category, son (child) nodes, and the value of the node. For example, enter **(disp-node-value 7)**.
4. Load **tracevars.lisp** file from the directory below, if you would like to see full f-structures.
    - /afs/cs/project/cmt-55/lti/Lab/Modules/GNL-721/

1.6 How to Create the Result File

1. Type: **(dribble "file-name")**
   e.g. (dribble "test-result")
2. Run a test file.
   e.g. (p* test-sentences)
3. When the test is done, type: **(dribble)** to close the file.
   The test result file (e.g. "test-result") will be created in the working directory.

2. Writing a Grammar for Analyzer

2.1 General Format of Grammar Rules

```
( context-free phrase structure rule
   ( list of equations))

(<S> <==> (<NP> <VP>)
   (((x1 case) = nominative)
    ((x1 agr) = (x2 agr))
    ((x0 subj) = x1)
    (x0 = x2)))
```

2.2 The Starting Symbol

```
(<start> <==> (<S>)
         ((x0 = x1)))

(<start> <==> (<NP>)
         ((x0 = x1)))
```

2.3 Equations

The left hand side of an equation is a path. A path is:

- A variable (e.g. x0, x1, etc.)
- A variable followed by any number of character strings separated by spaces.
  (x1 subj), (x2 xcomp subj)
  The character strings may not include certain special characters, such as the quotation mark.

  The type of path mush be enclosed in parentheses.

The right hand side of an equation is:

- A path
- A character string (e.g. foot, head, 12), excluding some special characters, such as the quotation mark.
- A list of consisting of the word (*OR* or *EOR*), followed by any number of character strings.

  e.g. (*OR* nominative accusative)

Each equation is enclosed in parentheses. The following is a list of example equations:

```
(x0 = x1)

((x0 subj) = x1)

((x1 case) = (*OR* nom acc))

((x1 agreement) = (x2 agreement))

((x0 root) = bird)
```

2.4 Disjunctive Equations

There are two types of disjunctive equations: *OR* and *EOR*.

A disjunction consists of the word, *OR* or *EOR*, followed by any number of lists of equations. Make sure to put an extra parenthesis in each list of equations. See the example below.

```
(*OR*
   (list-of-equations)
   (list-of-equations)
   (list-of-equations)
   ...)
```

Example:

```
(*OR*
   (
    ((x2 tense) = present)
    ((x1 agr) = (x2 agr))
   )
   (
    ((x2 tense) = past)
   )
)
```

2.5 Pseudo Equations

The following operators are only for the PSEUDO mode. If you would like to learn more about the differences between pseudo unification and full unification, read the CMT technical memo:

CMU-CMT-88-MEMO
Pseudo-Unification and Full-Unification
Masaru Tomita and Kevin Knight
November 1987

2.5.1 Constraint Equations

- Constraint equations use the symbol =c in place of the plain equal sign.
- A requlare equation causes unification or assignment of a value to a function, while constraint equation only checks to make sure that the function has the intended value.
- If the function does not already have the intended value, the parse will fail.
- **((x1 case) =c nom)**

  ((x1 case) =c (*OR* nom acc))
- The following equation doesn't work.

  ((x1 agr) =c (x2 agr))

2.5.2 Negative Equations

- The word *NOT* can be used on the right hand side of an equation to check to see if the value specified in the equation does not exist.

  ((x2 subcat) = (*NOT* intrans))

2.5.3 *UNDEFINED* and *DEFINED*

- The word *UNDEFINED* and *DEFINED* can be used on the right hand side of an equation.
- *UNDEFINED* makes sure that the left hand side of the equation has no value.
- *DEFINED* makes sure that the left hand side of the equation has a value.

```
((x1 negation) = *UNDEFINED*)
```

2.5.4 Assigning Multiple Values

- Multiple values can be assigned to a feature.
- Use the grater-than sign (>) in place of the equal sign.
- If the following rule applies recursively, the pp-adjunct function will have several different values at the same time:

```
(<S> <==> (<S> <PP>)
   ((x0 = x1)
    ((x0 pp-adjunct) > x2)))
```

2.6 Commenting the Grammar

Any line that begins with a semi-colon (;) is treated as a comment.

Table of Contents