Introduction to Grammar Writing

11-721
Grammars and Lexicons

Teruko Mitamura

teruko@cs.cmu.edu
www.cs.cmu.edu/~teruko

Outline

Part 5: Grammar Writing
- Goals of Grammar Writing Course
- Grammar Writing Project
- Schedule
- Introduction to Grammar Writing
  - Principle for Grammar Writing
  - Process of Grammar Writing
  - Design Issues
  - How to Write a Grammar Using Generalized LR Parser

Goals of Grammar Writing

- Understand principles of grammar writing
- Learn basic techniques for grammar writing
- Obtain awareness of real-world development issues through laboratory exercises (Learning by doing)
  - in class exercises
  - the grammar writing project

Requirements

- Part 5: Grammar Writing course counts as 35% of the course grade (attendance, in class exercises and the grammar writing project).
- Attendance and class participation is very important. If you miss class, you need to contact us before the class.
- It is your responsibility to obtain handouts and assignments if you miss the class.
- There will be Q/A sessions with TAs when necessary.

Schedule of Grammar Writing

Nov 5 Mon
- Schedule
- Introduction
- How to write a grammar

Before Nov 7, make sure that you can login to both Andrew and CS machines.

Nov 7 Wed: Class will meet in the Cluster: Hunt Near/Far Room
- Grammar writing project
- How to run the parser
- How to debug a grammar
- Q/A for the 1st assignment: step 1-2 and test suite for one type
- Grammar exercise (1)

Schedule of Grammar Writing (2)

Nov 12 Mon
- Finish grammar exercise (1) and hand it in at the end of class.
- Start grammar exercise (2)

Nov 14 Wed
- Finish grammar exercise (2) and hand it in at the end of class.
- Grammar Writing Project -- The 1st assignment Due

Nov 19 Mon
- Grammar exercise (3)

Nov 21 Wed No Class (Thanksgiving Break)

Schedule of Grammar Writing (3)

Nov 26 Mon
- Submit Grammar exercise (3)
- Start Grammar exercise (4)
- Feedback on the 1st assignment

Nov 28 Wed
- Submit Grammar exercise (4)
- Grammar exercise (5)

Dec 3 Mon
- Submit Grammar exercise (5)
- Q/A session

Schedule of Grammar Writing (4)

Dec 5 Wed
- Q/A session

Dec 7 Fri
- **Grammar Writing Project due at 3:00pm.**

Introduction to Grammar Writing

Principles for Grammar Writing

- Generality
- Extensibility
- Selectivity
- Simplicity

1. Generality

Capture linguistic generalization

Test for constituency
- Conjunction test
  - “I ate a hot dog and a sandwich.”
  - *“I ate a hot dog and on the stove.”
- Particles and Prepositions
  - “I **looked up** John’s phone number.”
  - “I **looked up** Mary’s chimney.”
  - *”I looked up John’s phone number and Mary’s chimney.”
  - “I looked up Mary’s chimney and in her cupboards.”

2. Extensibility

Able to extend grammar without having to rewrite a large portion of the grammar
- Additional structures
  - e.g. subordinate clauses, relative clauses
- Additional lexicons
- Free word order language (e.g. Japanese)

Japanese Examples

Nichiyoubi	ni	Ichiro	ga	hoomuran	wo	utta.
Sunday	on	Ichiro	NOM	home run	ACC	hit-PAST

“Ichiro hit a home run on Sunday.”

Nichiyoubi ni hoomuran wo Ichiro ga utta.
Ichiro ga nichiyoubi ni hoomuran wo utta.
Ichiro ga homuran wo nichiyoubi ni utta.
Hoomuran wo Ichiro ga nichiyoubi ni utta.
Hoomuran wo nichiyoubi ni Ichiro ga utta.

It’s not general or extensible to write phrase structure rules for each sentence.

3. Selectivity

- Not to over-generalize the grammar
- Ungrammatical sentences should fail

Birds fly.
*Birds flies.
*Bird fly.
A bird is flying.
*A bird are flying.

4. Simplicity

- Write clear, simple rules
  - Organization of rules: from top level categories to lower level rules
  - Use of general constraints rather than specific ones
  - Well-documented rules
  - Disjunctive equations within a rule VS. separate phrase structure rules

Example

```
(<s> <== (<np> <vp>)
     ((*EOR*
          (((x1 root) = "I")
           ((x2 form) = (*OR* rootform past am was)))
          (((x1 root) = (*OR* "he" "she" "it" "this" "that"))
           ((x2 form) = (*OR* present3sg past is was)))
          (((x1 root) = (*OR* "you" "we" "they" "those" "these" "there"))
           ((x2 form) = (*OR* rootform past are were)))
          (((x1 count) = +)
           ((x1 number) =c pl)
           ((x2 form) = (*OR* rootform past are were)))
          (((x1 count) = +)
           ((x1 number) = sg)
           ((x2 form) = (*OR* present3sg past is was)))
             ....
        (*OR* (((x2 form) = (*or* past was were))
               ((x2 tense) = past))
              (((x2 form) = (*or* rootform is are am present3sg))))
        (x0 = x2)
        ((x0 subj) = x1)))
```

Grammar Writing Project

- Develop a grammar for 9 types of English sentences
- Follow the process of Grammar Writing
- 1st assignment due on Nov 14 Wed in class
- The project is due on Dec 7 Friday at 3:00pm
- Late submission will be down-graded
- Work alone
- There will be no Final Exam
- More detail information in the next class

Process of Grammar Writing

7 Steps to follow:

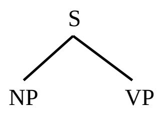
1. Planning
2. Design
3. Create test suite
4. Implement
5. Document
6. Test & Debug
7. Describe remaining issues

1. Planning the Task

- Set a goal
  - Purpose of developing a grammar
    - MT system, QA system, CALL system, etc.
  - Determine type of sentence structures
  - Determine sets of rules (e.g. S rules, NP rules)
- Make a schedule for tasks (when to do what)
- Estimate the time required for each step.

2. Grammar Design

- Decide set of structures to be covered.
  e.g. through corpus analysis
- For each type of structure:
  - Decide what the c-structure would look like.



$S \longrightarrow NP\ \ VP$

2. Grammar Design (2)

- For each type of structure:
  - Decide on the set of grammatical features.
  (e.g., person/number/gender agreement, verb class features, etc.)
  - Decide on the grammatical functions to be used
  (e.g., SUBJ, OBJ, PP, etc.)
  - Decide what the feature structure would look like.

  (cat n)
  (number sg)
  (form pastpart)

3. Create Test Suite

- Write the purpose of each test.
  (e.g. test for subj-verb agreement, etc.)
- Write each sentence type that should parse.
- Write sentences that shouldn't parse.
- Write why these sentences should fail.

4. Implement Grammar

- Organize the types of rules
  (e.g. start rules, NP rules, VP rules, PP rules, etc.)
- Write a phrase structure rule.
- Add equations to the phrase structure rule.
- Write morphology rules if necessary.
- Write lexical entries.

Grammar Rule Example

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

5. Documentation in the Grammar

- Cover page should include:
  - parser used
  - implementer's name(s) and dates
  - grammatical functions, features and values used
  - grammar change notes when changes occur

5. Documentation in the Grammar (2)

- Explain each type of rule
  - Sentence rules
  - NP rules
  - VP rules, etc.
- Write comments when necessary
  - e.g. ``This is to rule out wrong subj/verb agreement."
- Write short examples when necessary

6. Testing and Debugging

1. Create a test file from the test suite.
2. Run a test using the test file.
3. Check the result to see if you get the expected output.
4. If not, trace the grammar.
5. Debug the grammar.
6. Test the grammar again until you get it right.
7. Write the results into a file.
8. Comment on the results.
   (e.g. ambiguity, reason for failure, etc.)
9. Comment the fix in the grammar change note.

7. Describe remaining issues

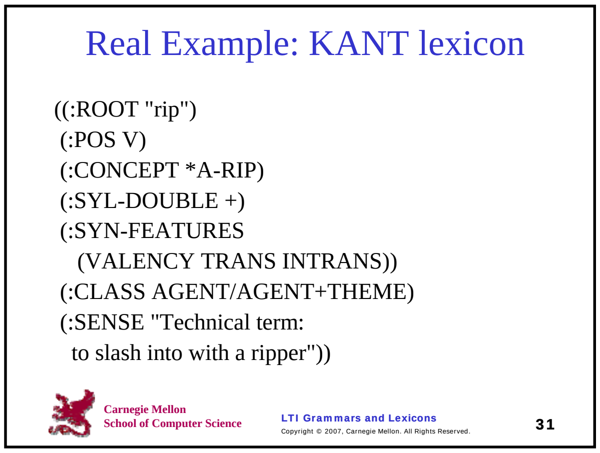
- Compare: Time estimate vs. Actual time spent
- Any unresolved problems
- Reasons for the problems
  - Ambiguity: reasons for more than one parse
  - Any limitations that you encountered
    - Platform limitations
    - Parser limitations
    - Grammar rule limitations
  - Other Reasons
- Other issues/Discussions
- Future Plan

Grammar Design Issues

- Coverage of the grammar
  - Objectives of the system
    - Machine translation
    - Language Tutoring
    - Information Retrieval
    - Question Answering
  - Type of documentation
    - e.g. general vs technical
  - Controlled vs General Language

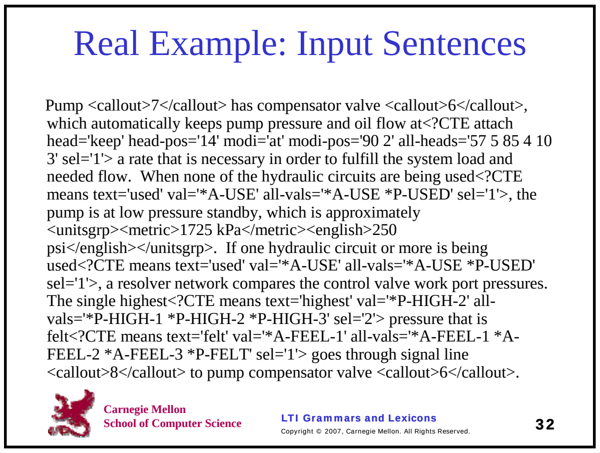
Grammar Design Issues (2)

- Linguistic Issues: Ambiguity resolution
  - Lexical ambiguity
    e.g. POS ambiguity, semantic ambiguity
  - Syntactic ambiguity
    e.g. PP attachment ambiguity
    N-N compound ambiguity
- Organization of the linguistic information
  - lexicon
  - morphology
  - syntax
  - domain semantics



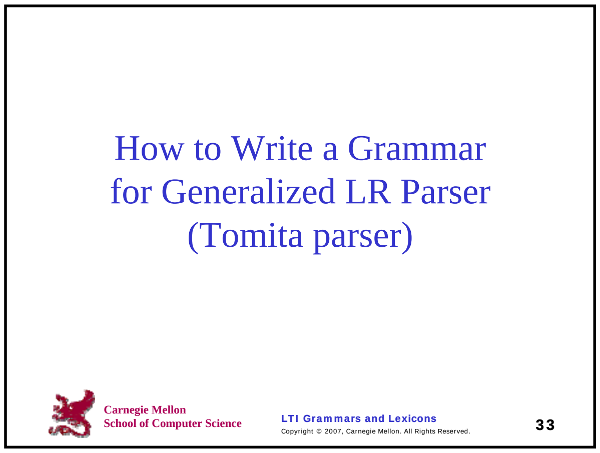
Real Example: KANT lexicon

```
((:ROOT "rip")
 (:POS V)
 (:CONCEPT *A-RIP)
 (:SYL-DOUBLE +)
 (:SYN-FEATURES
    (VALENCY TRANS INTRANS))
 (:CLASS AGENT/AGENT+THEME)
 (:SENSE "Technical term:
   to slash into with a ripper"))
```

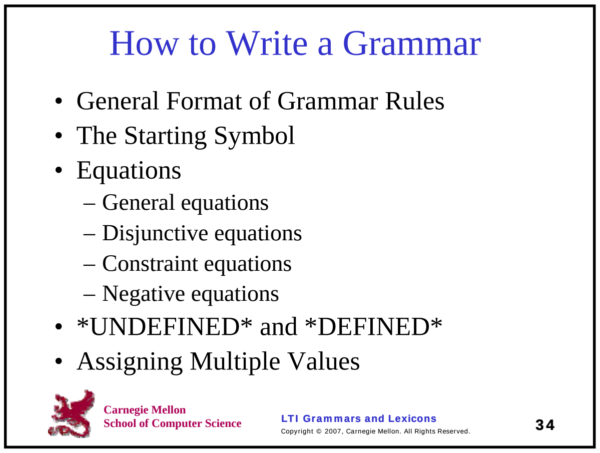


Real Example: Input Sentences

Pump <callout>7</callout> has compensator valve <callout>6</callout>, which automatically keeps pump pressure and oil flow at<?CTE attach head='keep' head-pos='14' modi='at' modi-pos='90 2' all-heads='57 5 85 4 10 3' sel='1'> a rate that is necessary in order to fulfill the system load and needed flow. When none of the hydraulic circuits are being used<?CTE means text='used' val='*A-USE' all-vals='*A-USE *P-USED' sel='1'>, the pump is at low pressure standby, which is approximately <unitsgrp><metric>1725 kPa</metric><english>250 psi</english></unitsgrp>. If one hydraulic circuit or more is being used<?CTE means text='used' val='*A-USE' all-vals='*A-USE *P-USED' sel='1'>, a resolver network compares the control valve work port pressures. The single highest<?CTE means text='highest' val='*P-HIGH-2' all-vals='*P-HIGH-1 *P-HIGH-2 *P-HIGH-3' sel='2'> pressure that is felt<?CTE means text='felt' val='*A-FEEL-1' all-vals='*A-FEEL-1 *A-FEEL-2 *A-FEEL-3 *P-FELT' sel='1'> goes through signal line <callout>8</callout> to pump compensator valve <callout>6</callout>.

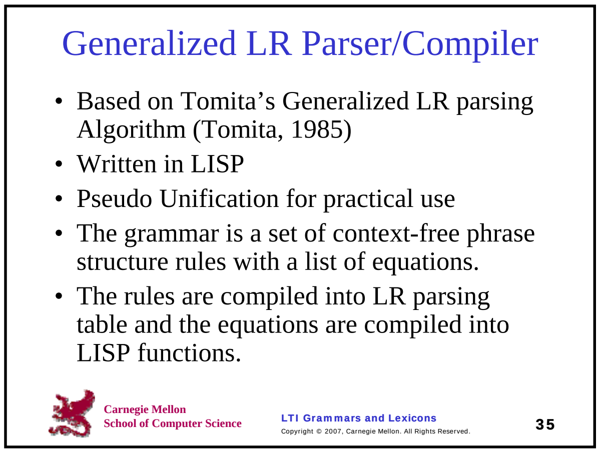


How to Write a Grammar for Generalized LR Parser (Tomita parser)



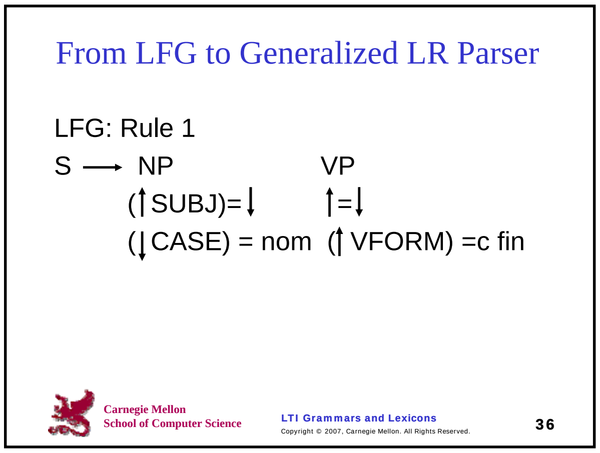
How to Write a Grammar

- General Format of Grammar Rules
- The Starting Symbol
- Equations
  - General equations
  - Disjunctive equations
  - Constraint equations
  - Negative equations
- *UNDEFINED* and *DEFINED*
- Assigning Multiple Values



Generalized LR Parser/Compiler

- Based on Tomita’s Generalized LR parsing Algorithm (Tomita, 1985)
- Written in LISP
- Pseudo Unification for practical use
- The grammar is a set of context-free phrase structure rules with a list of equations.
- The rules are compiled into LR parsing table and the equations are compiled into LISP functions.



From LFG to Generalized LR Parser

LFG: Rule 1

$$\begin{array}{lll} S \longrightarrow & NP & VP \\ & (\uparrow \text{SUBJ}) = \downarrow & \uparrow = \downarrow \\ & (\downarrow \text{CASE}) = \text{nom} & (\uparrow \text{VFORM}) =_c \text{fin} \end{array}$$

From LFG to Generalized LR Parser (2)

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

```
  x0          x1               x2
(<S> <==>  (< NP >              <VP >)
     (
        ((x1 CASE) = nom)  ((x0 VFORM) =c fin)
        ((x0 SUBJ) =  x1)     (x0 = x2)
     ))
```

From LFG to Generalized LR Parser (3)

LFG Rule 2:

VP --> V

$\uparrow = \downarrow$

GLR

```
( <VP> < == > (<V>)
        (
          (x0 = x1)
        ))
```

From LFG to Generalized LR Parser (4)

LFG Rule 3:

```
VP --> V        NP
       ↑ = ↓   (↑ OBJ) = ↓
               (↓ CASE) = acc
```

From LFG to Generalized LR Parser (5)

GLR

```
  (<VP> < == > (<V> <NP>)
      (
        ((x2 case) = acc)
        ((x0 obj) = x2)
        (x0 = x1)
      ))
```

General Format of Grammar Rules

```
  x0          x1    x2
  ↓           ↓     ↓
(<S> <==> (<NP> <VP>)
   (((x1 case) = nom)           | TEST
    ((x1 agr) = (x2 agr))       |
    ((x0 subj) = x1)            | ACTION
    (x0 = x2)))                 |
```

The Starting Symbol

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

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

Equations (1)

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 must be enclosed in parentheses.

Equations (2)

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)

Example Equations

Each equation is enclosed in parentheses:

(x0 = x1)

((x0 subj) = x1)

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

((x1 agreement) = (x2 agreement))

((x0 root) = bird)

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.

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

Example of Disjunctive Equations

Note that each disjunctive equation needs to be enclosed in parentheses.

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

Constraint Equations

- Constraint equations use the symbol **=c** in place of the plain equal sign.
- A regular 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.

Examples of Constraint Equations

((x1 case) =c nom)

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

This equation doesn't work.

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

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))

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*)

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)))
```

Commenting the Grammar

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

```
; <This is a comment.>
; (<start> <==> (<NP>)
;        ((x0 = x1)))
```

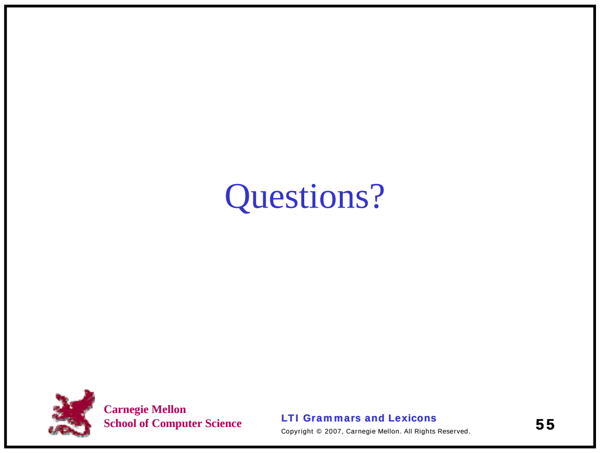
Schedule

Nov 7 Wed:

Class will meet in the Hunt Near/Far Cluster room

Before Nov 7, make sure that you can login to both Andrew and CS machines.

- Grammar writing project
- How to run the parser
- How to debug a grammar
- Q/A for the 1st assignment: step 1-2 and test suite for one type
- Start Grammar exercise (1)



Questions?