

# 15-482/11-682: Homework 3

## Helpful Things To Know

# Overview

- Input:
  - Your grammar file
  - Your lexicon
  - Training sentences (test.txt)
- Algorithm
  - Submit inputs to the parser server
- Output:
  - Parsed training sentences
  - Parse quality evaluation

# Sample Grammar Rules

- (`<start>` `<-->` (`<SENT>`)  
    (`(x0 = x1)`))
- (`<SENT>` `<-->` (`<ACT-Q>`)  
    (`(x0 = x1)`))
- (`<SENT>` `<-->` (`<DIR-Q>`)  
    (`(x0 = x1)`))
- (`<DIR-Q>` `<-->` (`who directed <M-NAME>`)  
    (`((x0 director) = *query)`  
    (`(x0 title) = (x3 title)`))
- (`<M-NAME>` `<-->` (`@CAP %`)  
    (`((x0 title) <= (get-name (x2 value))`))

# Sample Lexicon Entries

- ("act" V :VALENCY (\*OR\* \*subj-obj \*subj))  
("action" N U)  
("actor" N)  
("actress" N)  
("comedy" N U C)  
("make" V :PAST "made" :PASTPART "made"  
:VALENCY (\*OR\* \*subj-obj \*subj))

# You're Not Really Writing Lisp Programs

- *But* you need to know some Lisp syntax
- Parentheses/Brackets
  - Use an editor that has automatic matching!
  - Structure your file for simplicity:
    - e.g. with one line per rule

# List of List Types...

- Simple

( X Y Z )

( X )

- List of Lists

( X ( A B ) ( Y Z ) )

( ( X Y ) )

( ( X ) )

- List of List of Lists

( ( X ( A B ) ( Y Z ) )

( ( X Y ) )

( ( X ) )

)

# Grammar File Format

- A file is a set of rules (*not* a Lisp-list):  
R1  
R2  
...  
Rm
- You can have comments between rules:  
`;; This is ignored by the parser`
- Each rule  $R_i$  is a list:  
`( <NT> <--> ( <T1> <T2> ... <Tn> ) (URULES) )`

# Sample Grammar Rules

```
;; This is a sample grammar file
```

```
(<start> <--> (<SENT>)  
              ((x0 = x1)))  
(<SENT> <--> (<ACT-Q>)  
              ((x0 = x1)))  
(<SENT> <--> (<DIR-Q>)  
              ((x0 = x1)))  
(<DIR-Q> <--> (who directed <M-NAME>)  
               (((x0 director) = *query)  
                ((x0 title) = (x3 title))))  
(<M-NAME> <--> (@CAP %)  
                (((x0 title) <= (get-name (x2 value))))))
```

# F-struct labels correspond to ordering of symbols in rule

```
(<DIR-Q> <--> (who directed <M-NAME>)  
                (((x0 director) = *query)  
                 ((x0 title) = (x3 title))))
```

x0 → <DIR-Q>  
x1 → who  
x2 → directed  
x3 → <M-NAME>

# Unification Rules

( <NT> <--> ( <T1> <T2> ... <Tn> ) ( **URULES** ) )

- Format of URULES section is a list of rules
- Some example rules:
  - ( (x0 title) = \*query )
  - ( x0 = x1 )
  - ( (x0 dict cat) =c V )
  - ( (x0 dict cat) =c "perform" )
- For clarity, use one per line!

# Unification Rules

- Combine rules using \*OR\* operator:

```
( *OR*  
  (URULES1)  
  (URULES2)  
  ...  
  (URULESn)  
)
```

- Each (URULES<sub>*i*</sub>) is a *list* of unification rules (implicit AND)

# Special Grammar Rules

- **x0**                      **x1**  
( <NT> <--> (%)  
  ((x0 dict) <= (parse-word (x1 value))))
  
- **x0**                      **x1** **x2**  
( <M-NAME> <--> (@CAP %)  
  (((x0 title) <= (get-name (x2 value))))))
  
- Asterisk '\*' used to prefix non-token values  
  - ((x0 title) = \*query)

# Sample Lexicon Entries

```
("act" V :VALENCY (*OR* *subj-obj *subj))
("action" N U)
("actor" N)
("actress" N)
("comedy" N U C)
("make" V :PAST "made" :PASTPART "made"
  :VALENCY (*OR* *subj-obj *subj))
```

Typically, your new entries will be simple, having just the part of speech:

```
("foobar" N)
```

A word can have more than one entry if it has multiple parts of speech.

```
("bank" N)
("bank" V)
```