

# ECE49595NL Lecture 21: Parsing—II

Jeffrey Mark Siskind

Elmore Family School of Electrical and Computer Engineering

Spring 2025



Elmore Family School of Electrical  
and Computer Engineering

© 2025 Jeffrey Mark Siskind. All rights reserved.

# Some Abstractions

```
(define rule-lhs first)
(define rule-rhs1 third)
(define rule-rhs2 fourth)
(define entry-word first)
(define entry-category second)

(define (empty? word-string) (null? word-string))

(define (singleton? word-string)
  (= (length word-string) 1))

(define (head word-string) (first word-string))

(define (tail word-string) (rest word-string))

(define (ith-word word-string i)
  (list-ref word-string i))

(define (lookup word lexicon)
  (define (lookup lexicon)
    (when (null? lexicon) (fail))
    (if (string-ci=? word (entry-word (first lexicon)))
        (either (entry-category (first lexicon))
                 (lookup (rest lexicon)))
        (lookup (rest lexicon))))
  (lookup lexicon))
```

# Top Down Recognizer

$\text{FAILIFNOTPHRASE}(w, c)$

- ▶ Base case:  $w$  contains a single word
  - ▶ fail if  $\text{CATEGORY}(w) \neq c$
- ▶ Inductive case:  $w$  contains more than one word
  - ▶ choose a rule  $A \rightarrow BC$  where  $A = c$
  - ▶ split  $w$  into  $lr$
  - ▶  $\text{FAILIFNOTPHRASE}(l, B)$
  - ▶  $\text{FAILIFNOTPHRASE}(r, C)$

# Top Down Recognizer in Lisp

```
(define (split word-string)
  (define (split left right)
    (when (null? right) (fail))
    (either (list left right)
             (split (append left (list (first right)))
                    (rest right)))))
  (when (null? word-string) (fail))
  (split (list (first word-string)) (rest word-string)))

(define (top-down:is-sentence? word-string rules lexicon)
  (define fail-if-not-phrase
    (lambda (word-string category)
      (cond
        ((singleton? word-string)
         (unless (eq? (lookup (head word-string) lexicon)
                          category)
          (fail)))
        (#t)
        (else (let ((rule (a-member-of rules)))
                  (unless (eq? (rule-lhs rule) category)
                    (fail))
                  (let ((word-strings (split word-string)))
                    (fail-if-not-phrase (first word-strings)
                                         (rule-rhs1 rule))
                    (fail-if-not-phrase (second word-strings)
                                         (rule-rhs2 rule))))))))))
  (one-value (fail-if-not-phrase word-string 's) #f))
```

# Recursive Descent Recognizer

$\text{PEEL}(w, c)$

- ▶ fail if  $w$  is empty
- ▶ either
  - ▶ base case
    - ▶ fail if first word of  $w$  not of category  $c$
    - ▶ return tail of  $w$
  - ▶ inductive case
    - ▶ choose a rule  $A \rightarrow BC$  with  $A = c$
    - ▶ let  $w' = \text{PEEL}(w, B)$
    - ▶ return  $\text{PEEL}(w', C)$

# Recursive Descent Recognizer in Lisp

```
(define (recursive-descent:is-sentence?
      word-string rules lexicon)
  (define peel
    (lambda (word-string category)
      (when (empty? word-string) (fail))
      (either
        (begin
          (unless (eq? (lookup (head word-string) lexicon)
                        category)
            (fail))
          (tail word-string))
        (let ((rule (a-member-of rules)))
          (unless (eq? (rule-lhs rule) category) (fail))
          (peel (peel word-string (rule-rhs1 rule))
                (rule-rhs2 rule))))))
    (one-value (begin (unless (null? (peel word-string 's))
                          (fail))
                      #t)
              #f))
```

# Shift Reduce Recognizer

## SHIFTREDUCE

- ▶ Termination Condition
  - ▶ fail unless buffer is empty and stack has a single entry
  - ▶ return top of stack
- ▶ Shift
  - ▶ fail if buffer is empty
  - ▶ pop off first word in buffer and push its category on the stack
  - ▶ SHIFTREDUCE
- ▶ Reduce
  - ▶ fail if stack has less than two entries
  - ▶ choose a rule  $A \rightarrow BC$  where  $B$  = next of stack and  $C$  = top of stack
  - ▶ pop off top two entries from stack
  - ▶ push  $A$  on the stack
  - ▶ SHIFTREDUCE

# Shift Reduce Recognizer in Lisp

```
(define (shift-reduce:is-sentence?
      word-string rules lexicon)
  (define shift-reduce
    (lambda (stack word-string)
      (either (begin
                (unless (and (empty? word-string)
                              (= (length stack) 1))
                  (fail))
                (first stack))
              (begin (when (empty? word-string) (fail))
                      (shift-reduce
                       (cons (lookup (head word-string)
                                     lexicon)
                             stack)
                       (tail word-string))))
            (begin
              (when (< (length stack) 2) (fail))
              (let ((rule (a-member-of rules)))
                (unless (and (eq? (rule-rhs1 rule) (second stack))
                              (eq? (rule-rhs2 rule) (first stack)))
                  (fail))
                (shift-reduce
                 (cons (rule-lhs rule) (rest (rest stack)))
                 word-string)))))))
  (one-value
   (begin (unless (eq? (shift-reduce '() word-string) 's)
                 (fail))
          #t)
   #f))
```



# Complexity of Top Down Recognizer

OBSERVATION: Halts since length of `word-string` decreases at each recursive call and can never be less than zero.

Let  $p(n)$  be the number of recursive calls to `fail-if-not-phrase` needed to process a `word-string` of length  $n$ .

$$\begin{aligned} p(1) &= 1 \\ p(n) &= 1 + \sum_{i=1}^{n-1} p(i)p(n-i) \end{aligned}$$

Exponential in  $n$ .

# Recognition vs. Parsing

- ▶ Recognizer returns TRUE/FALSE
- ▶ Parser returns a parse tree
- ▶ Any recognizer can be turned into a parser independent of strategy, memoization, partial evaluation, . . .

# Top Down Recognizer $\Rightarrow$ Parser

FAILIFNOTPHRASE( $w, c$ )

- ▶ Base case:  $w$  contains a single word
  - ▶ fail if  $\text{CATEGORY}(w) \neq c$
- ▶ Inductive case:  $w$  contains more than one word
  - ▶ choose a rule  $A \rightarrow BC$  where  $A = c$
  - ▶ split  $w$  into  $lr$
  - ▶ FAILIFNOTPHRASE( $l, B$ )
  - ▶ FAILIFNOTPHRASE( $r, C$ )

$\Downarrow$

APARSEOF( $w, c$ )

- ▶ Base case:  $w$  contains a single word
  - ▶ fail if  $\text{CATEGORY}(w) \neq c$
  - ▶ otherwise return
- ▶ Inductive case:  $w$  contains more than one word
  - ▶ choose a rule  $A \rightarrow BC$  where  $A = c$
  - ▶ split  $w$  into  $lr$
  - ▶ let  $t_1$  be APARSEOF( $l, B$ )
  - ▶ let  $t_2$  be APARSEOF( $r, C$ )
  - ▶ return

$c$   
|  
 $w$

$A$   
└─┬─  
 $t_1$   $t_2$

# Recursive Descent Recognizer $\Rightarrow$ Parser

PEEL( $w, c$ )

- ▶ fail if  $w$  is empty
- ▶ either
  - ▶ base case
    - ▶ fail if first word of  $w$  not of category  $c$
    - ▶ return tail of  $w$
  - ▶ inductive case
    - ▶ choose a rule  $A \rightarrow BC$  with  $A = c$
    - ▶ let  $w' = \text{PEEL}(w, B)$
    - ▶ return  $\text{PEEL}(w', C)$

$\Downarrow$

PEEL( $w, c$ )

- ▶ fail if  $w$  is empty
- ▶ either
  - ▶ base case
    - ▶ fail if first word of  $w$  not of category  $c$
    - ▶ return  $\langle \text{TAIL}(w), t \rangle$  where  $t$  is
  - ▶ inductive case
    - ▶ choose a rule  $A \rightarrow BC$  with  $A = c$
    - ▶ let  $\langle w', t_1 \rangle = \text{PEEL}(w, B)$
    - ▶ let  $\langle w'', t_2 \rangle = \text{PEEL}(w', C)$
    - ▶ return  $\langle w'', t \rangle$  where  $t$  is

$c$   
|  
 $w$

$A$   
└─┬─  
 $t_1$   $t_2$

# Shift Reduce Recognizer $\Rightarrow$ Parser

## SHIFTREDUCE

- ▶ Termination Condition
  - ▶ fail unless buffer is empty and stack has a single entry
  - ▶ return top of stack
- ▶ Shift
  - ▶ fail if buffer is empty
  - ▶ pop off first word in buffer and push its category on the stack
  - ▶ SHIFTREDUCE
- ▶ Reduce
  - ▶ fail if stack has less than two entries
  - ▶ choose a rule  $A \rightarrow BC$  where  $B$  = next of stack and  $C$  = top of stack
  - ▶ pop off top two entries from stack
  - ▶ push  $A$  on stack
  - ▶ SHIFTREDUCE



## SHIFTREDUCE

- ▶ Termination Condition
  - ▶ fail unless buffer is empty and stack has a single entry
  - ▶ return top of stack
- ▶ Shift
  - ▶ fail if buffer is empty
  - ▶ pop off first word  $w$  in buffer and push  $\langle c, t \rangle$  on the stack where  $c$  is the category of  $w$  and  $t$  is

$$\begin{array}{c} \mathcal{C} \\ | \\ \mathcal{W} \end{array}$$

- ▶ SHIFTREDUCE
- ▶ Reduce
  - ▶ fail if stack has less than two entries
  - ▶ pop  $\langle c, t_2 \rangle$  off the stack
  - ▶ pop  $\langle b, t_1 \rangle$  off the stack
  - ▶ choose a rule  $A \rightarrow BC$  where  $B = b$  and  $C = c$
  - ▶ push  $\langle A, t \rangle$  on the stack where  $t$  is

$$\begin{array}{c} A \\ \wedge \\ t_1 \quad t_2 \end{array}$$

- ▶
- SHIFTREDUCE**