ECE49595NL Lecture 17: Partial Evaluation

Jeffrey Mark Siskind

Elmore Family School of Electrical and Computer Engineering

Spring 2024

The CKY Algorithm

Cocke, Kasami (1965), Younger (1967)

Bottom-Up

$_0$ *Some* $_1$ *pawn* $_2$ *is* $_3$ *on* $_4$ *some* $_5$ *square* $_6$

Chart, Well-Formed Substring Table

Each entry of $M[i,j]$ will store a subset of the set of categories that can occur in the LHS of some rule

$c \in M[i,j]$ iff the substring from i to j can be a phrase of type c

Initialize $M[i,i+1]$ to the category of word i.

$$M[i,j] \leftarrow \{A | A \Rightarrow B\ C \wedge i < k < j \wedge B \in M[i,k] \wedge C \in M[k,j]\}$$

CKY as Matrix Multiplication

1. Boolean matrices for each category: S, NP, ...
2. Initialize all matrices to zero.
3. Set $c[i, i+1]$ to one if word i can be of category c.
4. For each rule of the form $A \Rightarrow B\ C$ compute $A \leftarrow A + BC$
   Where multiplication is Boolean AND and addition is Boolean OR
5. Repeat step 4 until no matrix changes.

Partial Evaluation—I

```
(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))
```

Partial Evaluation—II

```
(define (recursive-descent:is-sentence? word-string)
 (define peel
  (lambda (word-string category)
   (when (empty? word-string) (fail))
   (either
    (begin
     (unless (eq? (lookup (head word-string)
                          '(("square" n)
                            ("piece" n)
                            ("pawn" n)
                            ("bishop" n)
                            ("knight" n)
                            ("rook" n)
                            ("queen" n)
                            ("king" n)
                            ("rank" n)
                            ("file" n)
                            ("edge" n)
                            ("corner" n)
                            ("black" a)
                            ("white" a)
                            ("every" d)
                            ("some" d)
                            ("the" d)
                            ("is" v)
                            ("of" p)
                            ("on" p)))
                  category)
      (fail))
     (tail word-string))
    (let ((rule (a-member-of '((s ==> np vp)
                               (np ==> d nb)
                               (np ==> d n)
                               (nb ==> a nb)
                               (nb ==> a n)
                               (nb ==> n pp)
                               (pp ==> p np)
                               (vp ==> v np)
                               (vp ==> v pp)))))
     (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))
```

Partial Evaluation—III

```
(define (recursive-descent:is-sentence? word-string)
 (define peel-n
  (lambda (word-string)
   ...))
 (define peel-n
  (lambda (word-string)
   ...))
 (define peel-n
  (lambda (word-string)
   ...))
 (define peel-n
  (lambda (word-string)
   ...))
 (define peel-n
  (lambda (word-string)
   ...))
 (define peel-s
  (lambda (word-string)
   ...))
 (define peel-np
  (lambda (word-string)
   ...))
 (define peel-nb
  (lambda (word-string)
   ...))
 (define peel-pp
  (lambda (word-string)
   ...))
 (define peel-vp
  (lambda (word-string)
   ...))
 (one-value (begin (unless (null? (peel-s word-string))
                     (fail))
                   #t)
            #f))
```

Partial Evaluation—IV

```
(define peel-n
 (lambda (word-string category)
  (when (empty? word-string) (fail))
  (either
   (begin
    (unless (eq? (lookup (head word-string)
                         '(("square" n)
                           ("piece" n)
                           ("pawn" n)
                           ("bishop" n)
                           ("knight" n)
                           ("rook" n)
                           ("queen" n)
                           ("king" n)
                           ("rank" n)
                           ("file" n)
                           ("edge" n)
                           ("corner" n)
                           ("black" a)
                           ("white" a)
                           ("every" d)
                           ("some" d)
                           ("the" d)
                           ("is" v)
                           ("of" p)
                           ("on" p)))
                 category)
     (fail))
    (tail word-string))
   (let ((rule (a-member-of '((s ==> np vp)
                              (np ==> d nb)
                              (np ==> d n)
                              (nb ==> a nb)
                              (nb ==> a n)
                              (nb ==> n pp)
                              (pp ==> p np)
                              (vp ==> v np)
                              (vp ==> v pp)))))
    (unless (eq? (rule-lhs rule) category) (fail))
    (peel (peel word-string (rule-rhs1 rule))
          (rule-rhs2 rule))))))
```

Partial Evaluation—V

```
(define peel-n
 (lambda (word-string)
  (when (empty? word-string) (fail))
  (unless (eq? (lookup (head word-string)
                       '(("square" n)
                         ("piece" n)
                         ("pawn" n)
                         ("bishop" n)
                         ("knight" n)
                         ("rook" n)
                         ("queen" n)
                         ("king" n)
                         ("rank" n)
                         ("file" n)
                         ("edge" n)
                         ("corner" n)
                         ("black" a)
                         ("white" a)
                         ("every" d)
                         ("some" d)
                         ("the" d)
                         ("is" v)
                         ("of" p)
                         ("on" p)))
               'n)
   (fail))
  (tail word-string)))
```

Partial Evaluation—VI

```
(define peel-n
 (lambda (word-string)
  (when (empty? word-string) (fail))
  (unless (eq? (lookup (head word-string)
                       '(("square" n)
                         ("piece" n)
                         ("pawn" n)
                         ("bishop" n)
                         ("knight" n)
                         ("rook" n)
                         ("queen" n)
                         ("king" n)
                         ("rank" n)
                         ("file" n)
                         ("edge" n)
                         ("corner" n)))
               'n)
   (fail))
  (tail word-string)))
```

Partial Evaluation—VII

```
(define peel-n
 (lambda (word-string)
  (when (empty? word-string) (fail))
  (unless (or (string-ci=? (head word-string) "square")
              (string-ci=? (head word-string) "piece")
              (string-ci=? (head word-string) "pawn")
              (string-ci=? (head word-string) "bishop")
              (string-ci=? (head word-string) "knight")
              (string-ci=? (head word-string) "rook")
              (string-ci=? (head word-string) "queen")
              (string-ci=? (head word-string) "king")
              (string-ci=? (head word-string) "rank")
              (string-ci=? (head word-string) "file")
              (string-ci=? (head word-string) "edge")
              (string-ci=? (head word-string) "corner"))
   (fail))
  (tail word-string)))
```

Partial Evaluation—VII

```
(define peel-nb
 (lambda (word-string)
  (when (empty? word-string) (fail))
  (let ((rule (a-member-of '((s ==> np vp)
                             (np ==> d nb)
                             (np ==> d n)
                             (nb ==> a nb)
                             (nb ==> a n)
                             (nb ==> n pp)
                             (pp ==> p np)
                             (vp ==> v np)
                             (vp ==> v pp)))))
   (unless (eq? (rule-lhs rule) 'nb) (fail))
   (peel (peel word-string (rule-rhs1 rule))
         (rule-rhs2 rule)))))
```

Partial Evaluation—IX

```
(define peel-nb
 (lambda (word-string)
  (when (empty? word-string) (fail))
  (let ((rule (a-member-of '((nb ==> a nb)
                             (nb ==> a n)
                             (nb ==> n pp)))))
   (unless (eq? (rule-lhs rule) 'nb) (fail))
   (peel (peel word-string (rule-rhs1 rule))
         (rule-rhs2 rule)))))
```

Partial Evaluation—X

```
(define peel-nb
 (lambda (word-string)
  (when (empty? word-string) (fail))
  (either (peel-nb (peel-a word-string))
          (peel-n (peel-a word-string))
          (peel-pp (peel-n word-string)))))
```

Partial Evaluation—XI

```
(define (recursive-descent:is-sentence? word-string)
 (define peel-n
  (lambda (word-string)
   (when (empty? word-string) (fail))
   (unless (or (string-ci=? (head word-string) "square")
               (string-ci=? (head word-string) "piece")
               (string-ci=? (head word-string) "pawn")
               (string-ci=? (head word-string) "bishop")
               (string-ci=? (head word-string) "knight")
               (string-ci=? (head word-string) "rook")
               (string-ci=? (head word-string) "queen")
               (string-ci=? (head word-string) "king")
               (string-ci=? (head word-string) "rank")
               (string-ci=? (head word-string) "file")
               (string-ci=? (head word-string) "edge")
               (string-ci=? (head word-string) "corner"))
    (fail))
   (tail word-string)))
 (define peel-a
  (lambda (word-string)
   (when (empty? word-string) (fail))
   (unless (or (string-ci=? (head word-string) "black")
               (string-ci=? (head word-string) "white"))
    (fail))
   (tail word-string)))
 (define peel-d
  (lambda (word-string)
   (when (empty? word-string) (fail))
   (unless (or (string-ci=? (head word-string) "every")
               (string-ci=? (head word-string) "some")
               (string-ci=? (head word-string) "the"))
    (fail))
   (tail word-string)))
 (define peel-v
  (lambda (word-string)
   (when (empty? word-string) (fail))
   (unless (string-ci=? (head word-string) "is")
    (fail))
   (tail word-string)))
 (define peel-p
  (lambda (word-string)
   (when (empty? word-string) (fail))
   (unless (or (string-ci=? (head word-string) "of")
               (string-ci=? (head word-string) "on"))
    (fail))
   (tail word-string)))
```

Partial Evaluation—XII

```
(define peel-s
 (lambda (word-string)
  (when (empty? word-string) (fail))
  (peel-vp (peel-np word-string))))
(define peel-np
 (lambda (word-string)
  (when (empty? word-string) (fail))
  (either (peel-nb (peel-d word-string))
          (peel-n (peel-d word-string)))))
(define peel-nb
 (lambda (word-string)
  (when (empty? word-string) (fail))
  (either (peel-nb (peel-a word-string))
          (peel-n (peel-a word-string))
          (peel-pp (peel-n word-string)))))
(define peel-pp
 (lambda (word-string)
  (when (empty? word-string) (fail))
  (peel-np (peel-p word-string))))
(define peel-vp
 (lambda (word-string)
  (when (empty? word-string) (fail))
  (either (peel-np (peel-v word-string))
          (peel-vp (peel-v word-string)))))
(one-value (begin (unless (null? (peel-s word-string))
                    (fail))
                  #t)
           #f))
```