人工知能特論II

二宮　崇

今日の講義の予定

- CFG構文解析
- 教科書
  - 北研二(著) 辻井潤一(編) 言語と計算4 確率的言語モデル 東大出版会
  - C. D. Manning & Hinrich Schütze “FOUNDATIONS OF STATISTICAL NATURAL LANGUAGE PROCESSING” MIT Press, 1999
  - D. Jurafsky, J. H. Martin, A. Kehler, K.V. Linden & N. Ward “**Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition**” Prentice Hall Series in Artificial Intelligence, 2000

CFGの構文解析

- ある文sが与えられた時、文法Gによって導出できる全ての構文木を導出する構文解析
- 何のために？
  - PCFG構文解析の基礎
  - 構文解析後に、確率計算を行って、最も良い構文木を選択する
  - パラメータ推定の際に構文木の候補集合が必要（学習方法によっては必要ない）

CFG構文解析

CFG構文解析のアルゴリズム

- トップダウン型
  - アーリー法 (earley parsing algorithm)
- ボトムアップ型
  - CKY法 (CKY parsing algorithm, CYK法ともいう)
  - チャート法 (chart parsing algorithm)
  - 左隅解析法 (left-corner parsing algorithm)
- 一般化LR法 (generalized LR parsing)

CKY法

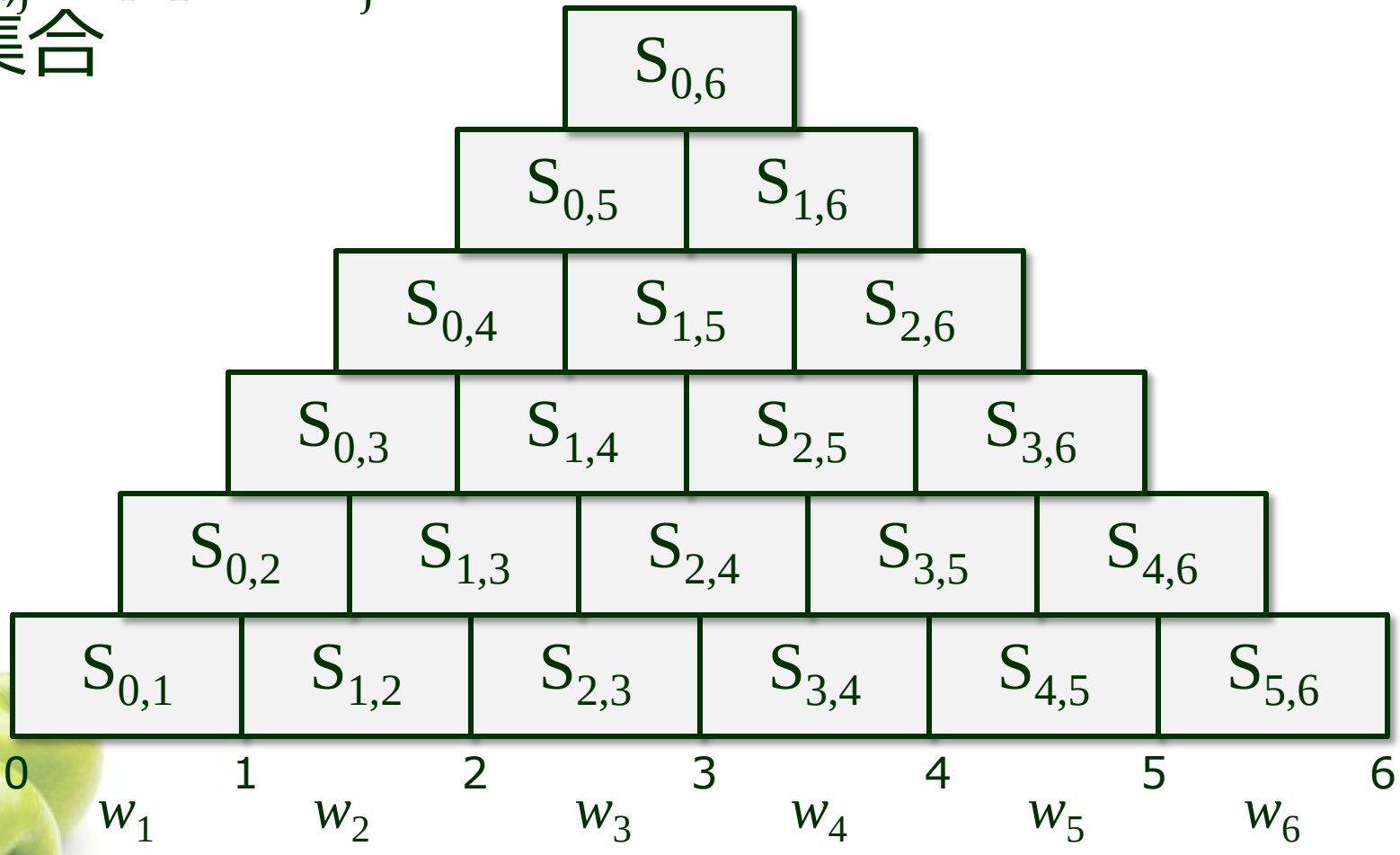
- Cocke, Kasami, Youngerにより提案され、それぞれの頭文字をとって、CKYもしくはCYK構文解析アルゴリズムと呼ばれる
- 多くのパーザーで用いられている
  - 簡単
  - 効率が良い
  - デコーディングと相性が良い
- 文法規則はバイナリルールかユーナリールールのみ
  - バイナリールール: 書換規則の右側の要素が二つしかないルール
  - ユーナリールール: 書換規則の右側の要素が一つしかないルール
  - CFGならチョムスキー標準形に変形
  - HPSG、CCGではバイナリールールを想定しているので特に問題は無い

準備: 書換規則と位置

- 書換規則は次の3つを想定
  - A → B C (バイナリールール)
  - A → B (ユーナリールール)
  - A → w (辞書ルール)
- 位置
  - 文 $w_1,w_2,\ldots,w_n$ が与えられた時、
    - 単語w_iの位置: <i-1, i>
    - 句 $w_i,\ldots,w_j$の位置: <i-1, j>

準備: CKYテーブル（チャート）

- $S_{i,j}$: w_{i+1},…, w_jに対応する句の非終端記号の集合

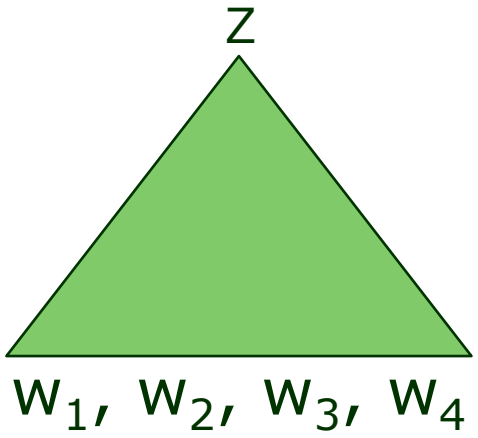


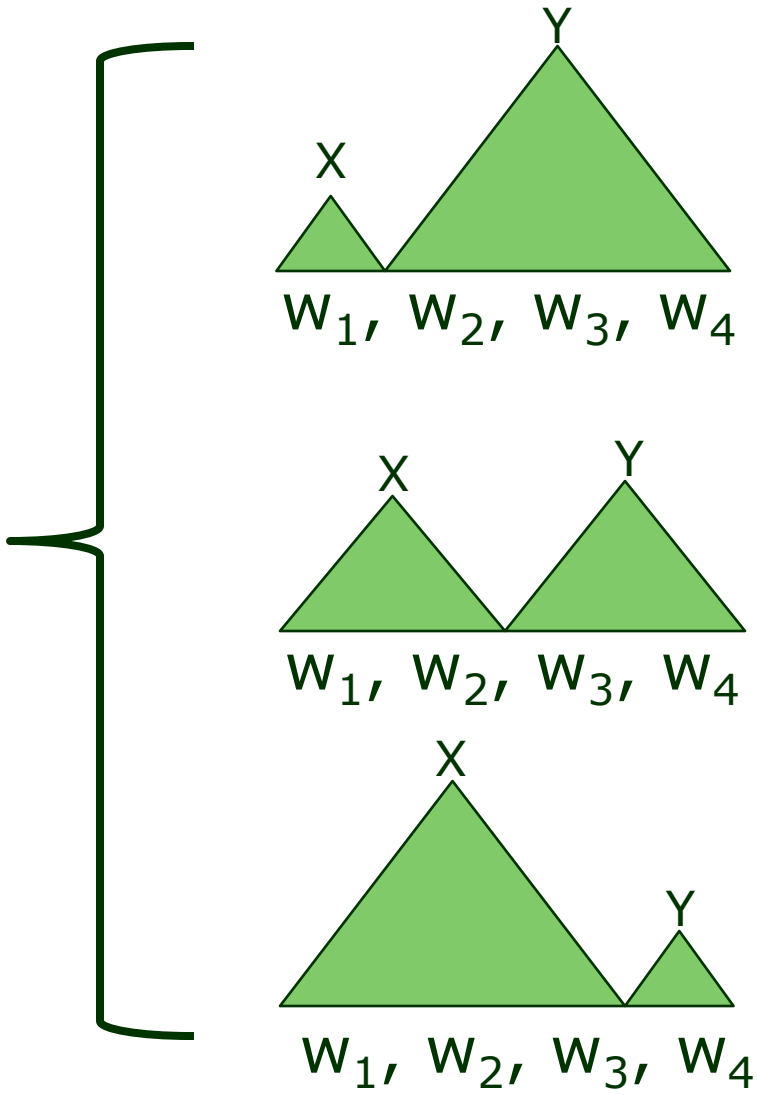
CKY法: 基本的なアイデア

- 目的: $S_{0, n}$を計算
- $S_{i,j}$は次のSから計算できる
  - $S_{i, i+1}$と$S_{i+1, j}$
  - $S_{i, i+2}$と$S_{i+2, j}$
  - ....
  - $S_{i, j-1}$と$S_{j-1, j}$

CKY法: 基本的なアイデア

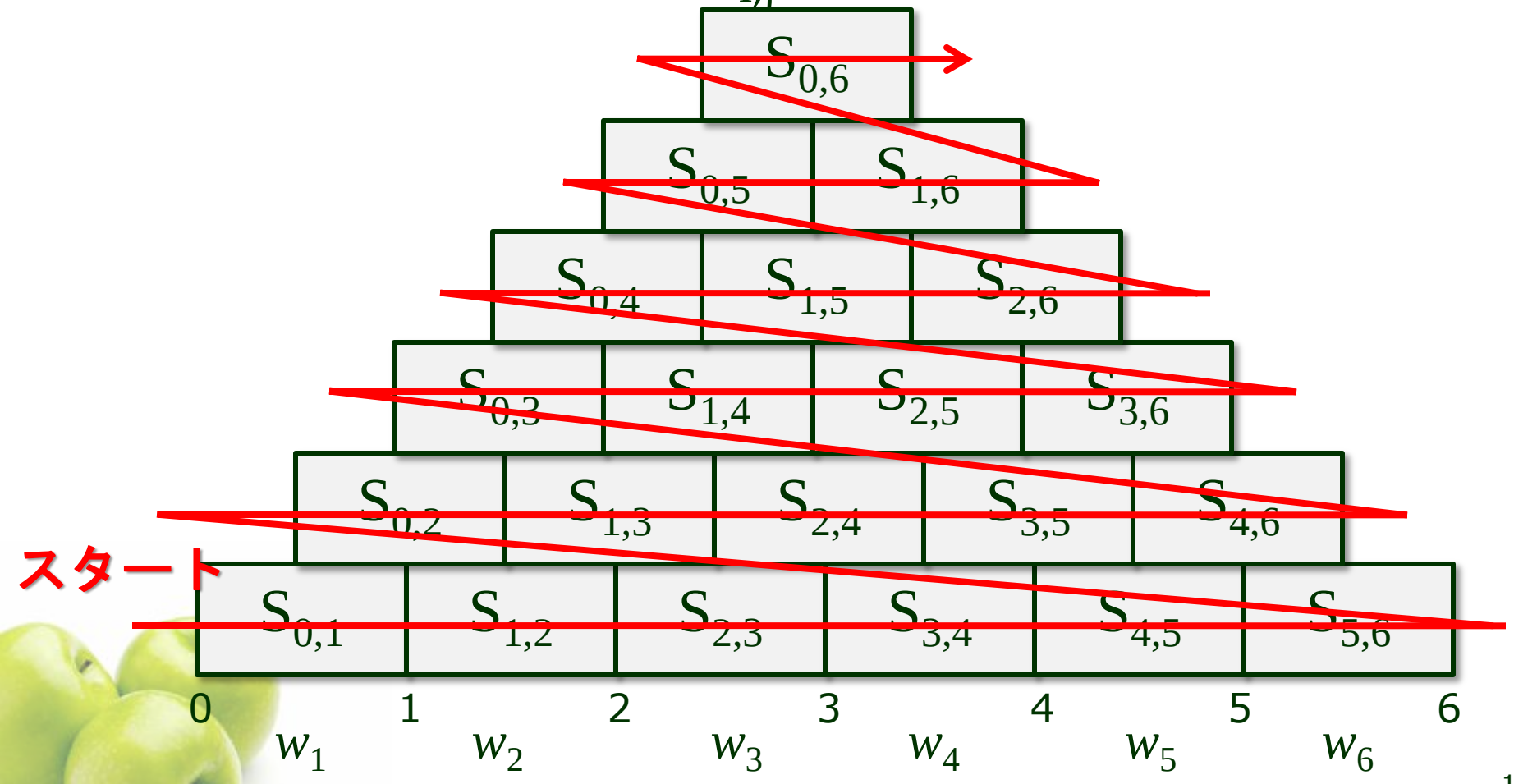
- $Z \rightarrow X\ Y$





CKY法

- 矢印の順で全ての$S_{i,j}$が求まる



ルール適用と$S_{i,j}$の求め方

- $G(X, Y) = \{Z \mid \exists p \in P.p=(Z \rightarrow X\ Y)\}$
  - $X\ Y$に対する全ての親を返す関数
    - X, Y: 非終端記号
    - P: 書換規則の集合
- $S_{i,j}$を求めるアルゴリズム

for $k = i+1$ to $j-1$

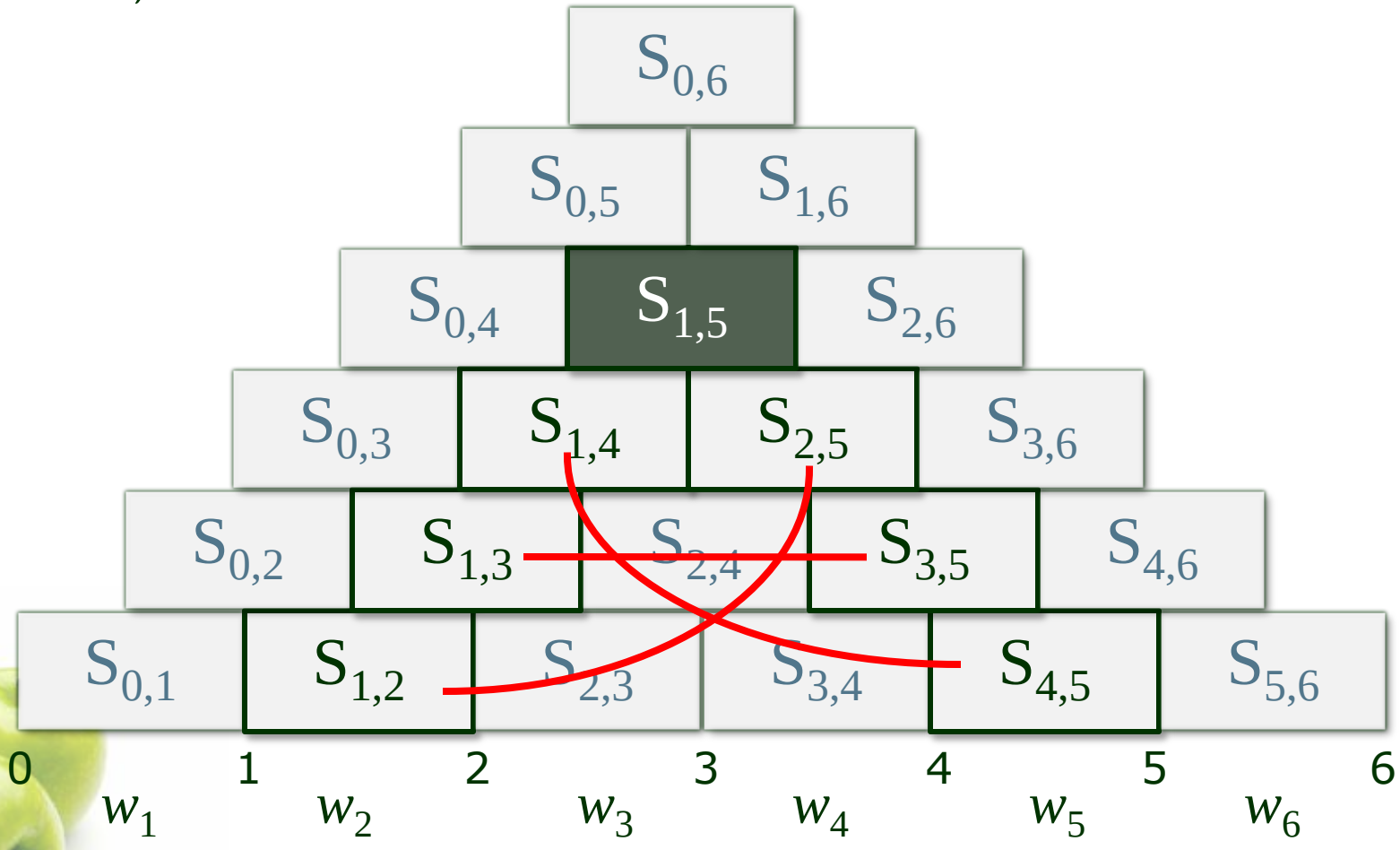
  forall $X \in S_{i,k}$

    forall $Y \in S_{k,j}$

      $S_{i,j} := S_{i,j} \cup G(X, Y)$

CKY法: $S_{i,j}$

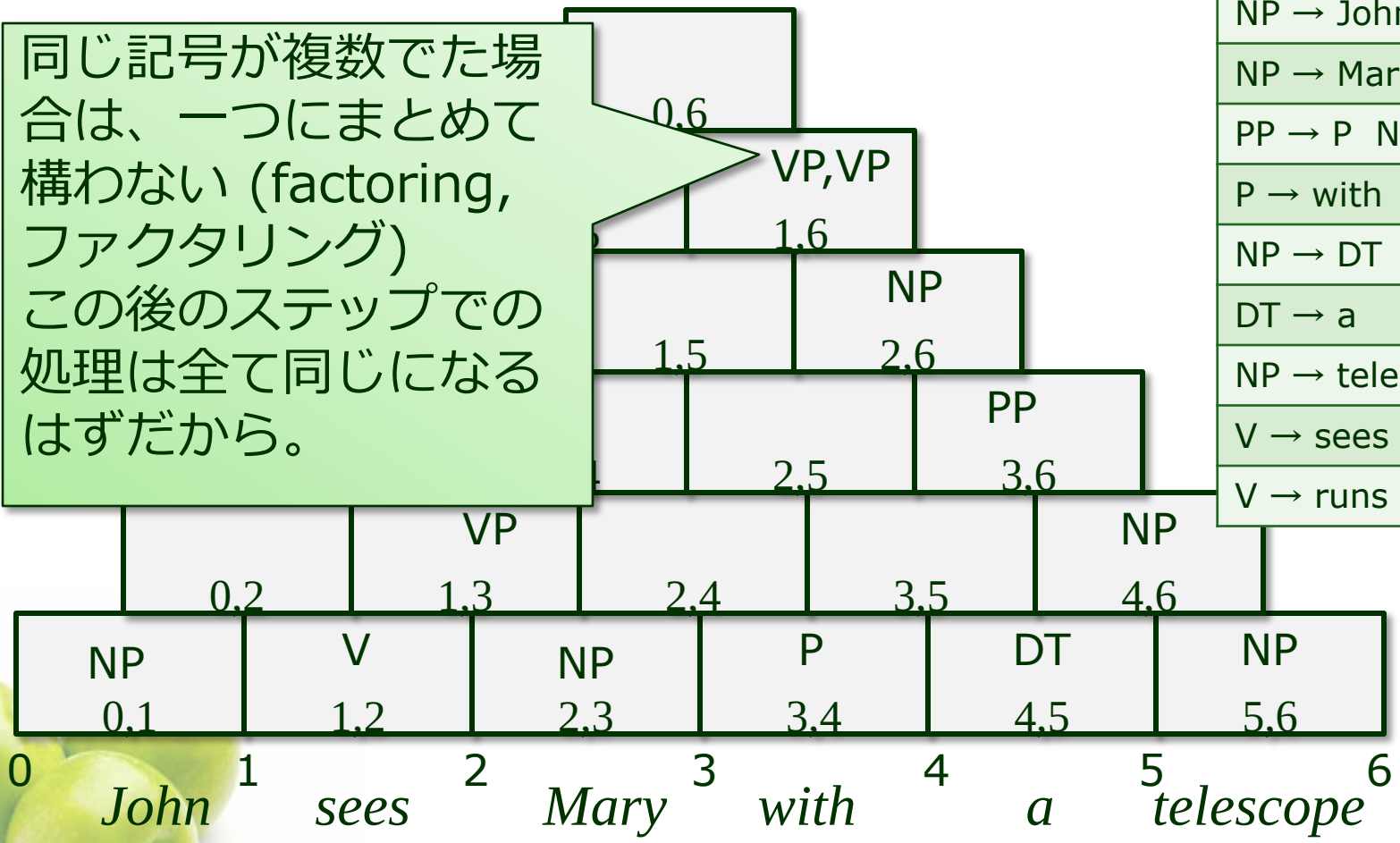
- 例: $S_{1,5}$に対しk=2,3,4



CKY法

- 例

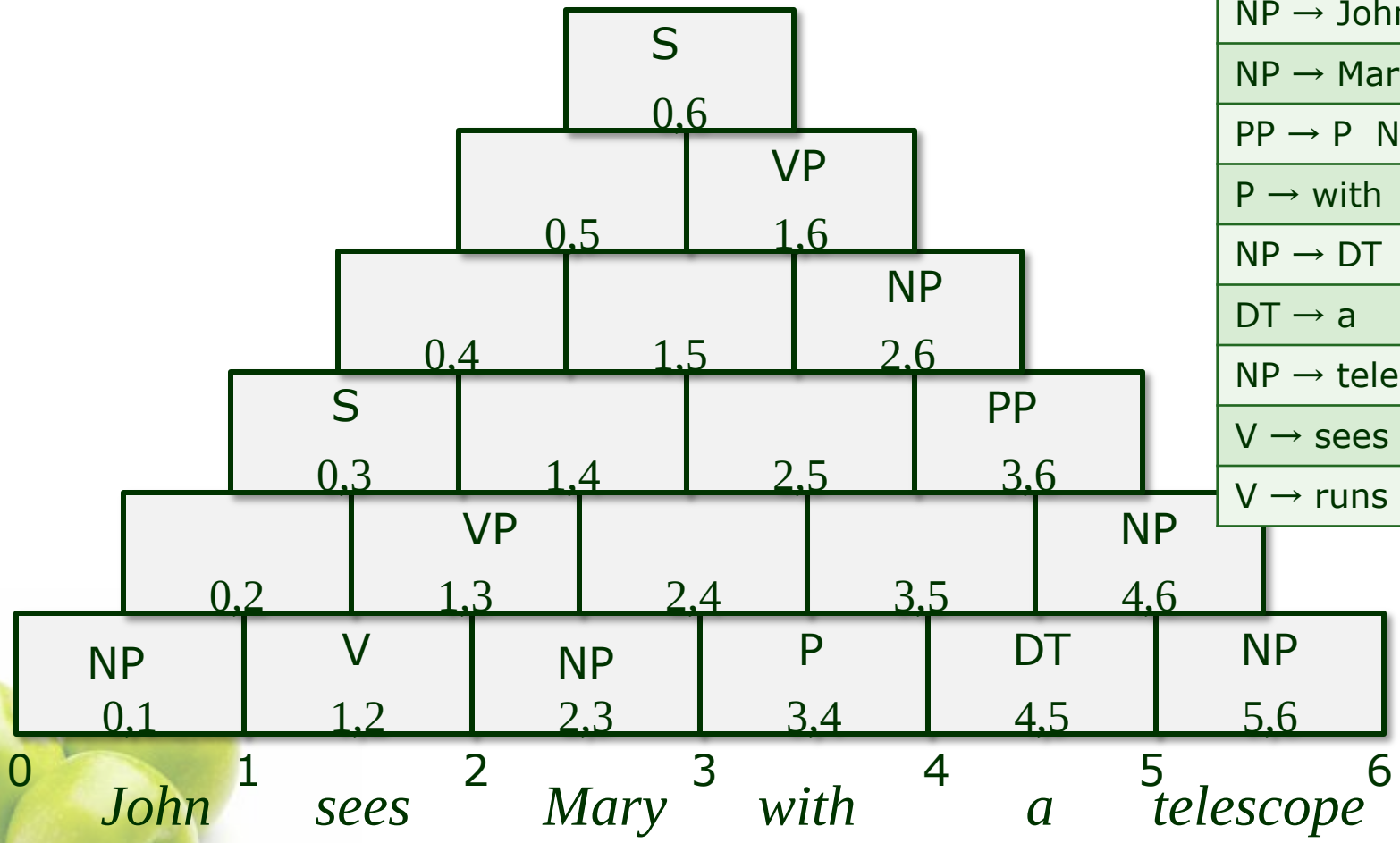
同じ記号が複数でた場合は、一つにまとめて構わない (factoring, ファクタリング)
この後のステップでの処理は全て同じになるはずだから。



文法
S → NP VP
VP → VP PP
VP → V NP
VP → V
NP → NP PP
NP → John
NP → Mary
PP → P NP
P → with
NP → DT NP
DT → a
NP → telescope
V → sees
V → runs

CKY法

- 例



文法
S → NP VP
VP → VP PP
VP → V NP
VP → V
NP → NP PP
NP → John
NP → Mary
PP → P NP
P → with
NP → DT NP
DT → a
NP → telescope
V → sees
V → runs

CKY法: アルゴリズム

for j = 1 to n

$S_{j-1,j} := L(w_j)$　## Lは単語wに対する非終端記号の集合を返す関数

for l = 2 to n

for i = 0 to n – l

j := i + l;

for k = i+1 to j - 1

forall $X \in S_{i,k}$

forall $Y \in S_{k,j}$

$Si,j := Si,j \cup G(X, Y)$

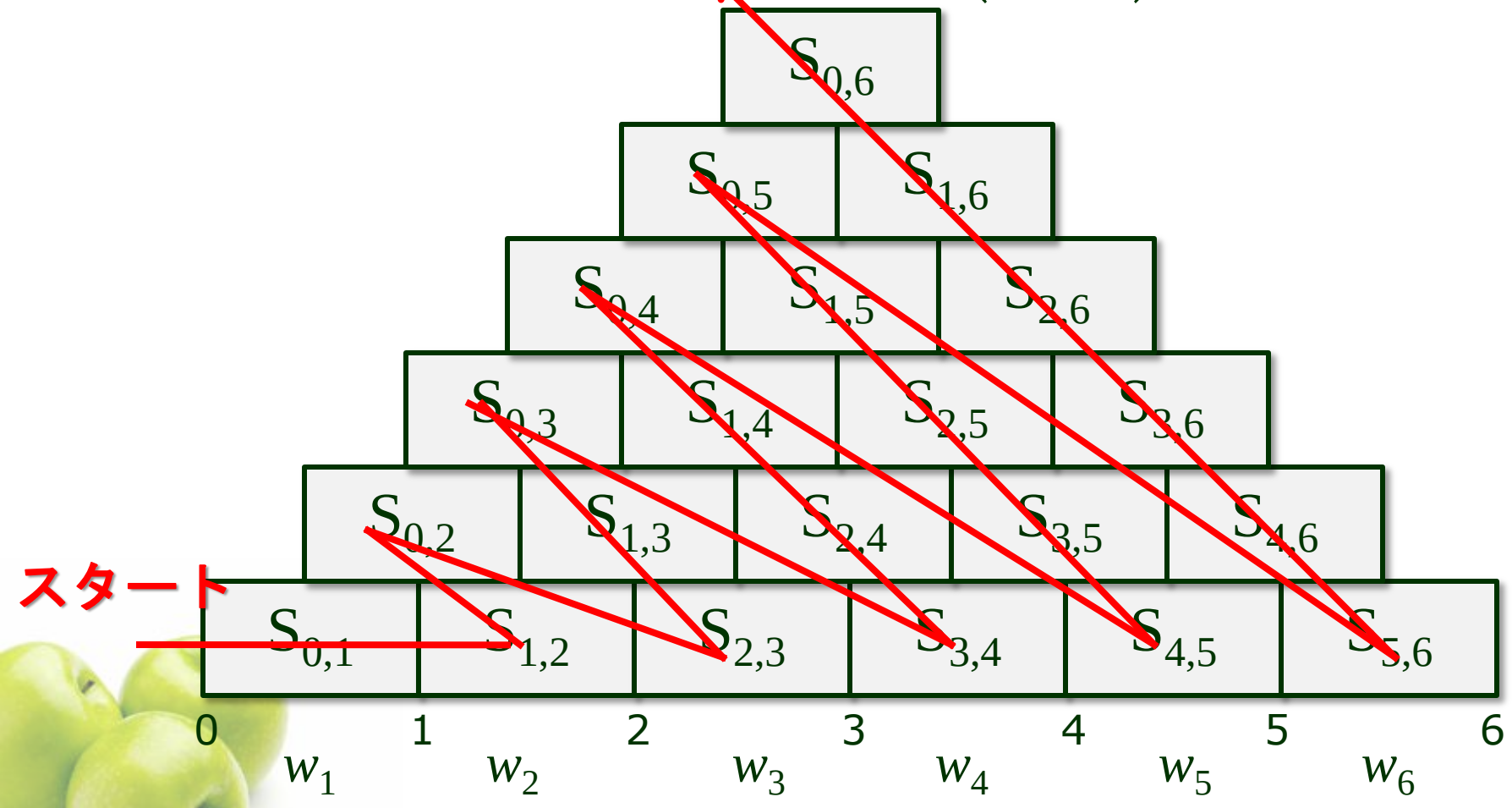
$S_{i,j} := S_{i,j} \cup U(S_{i,j})$ ## Uはユーナリールールを適用して得られる非終端記号集合

CKY法: 計算量

- 最悪時間計算量 (worst-case time complexity)
  - $O(n^3)$
    - nは文長
    - アルゴリズムより明らか
  - 非終端記号数を$|V_N|$とすると、$O(n^3|V_N|^2)$
  - ファクタリングのおかげで計算量が指数爆発していないということに注意！

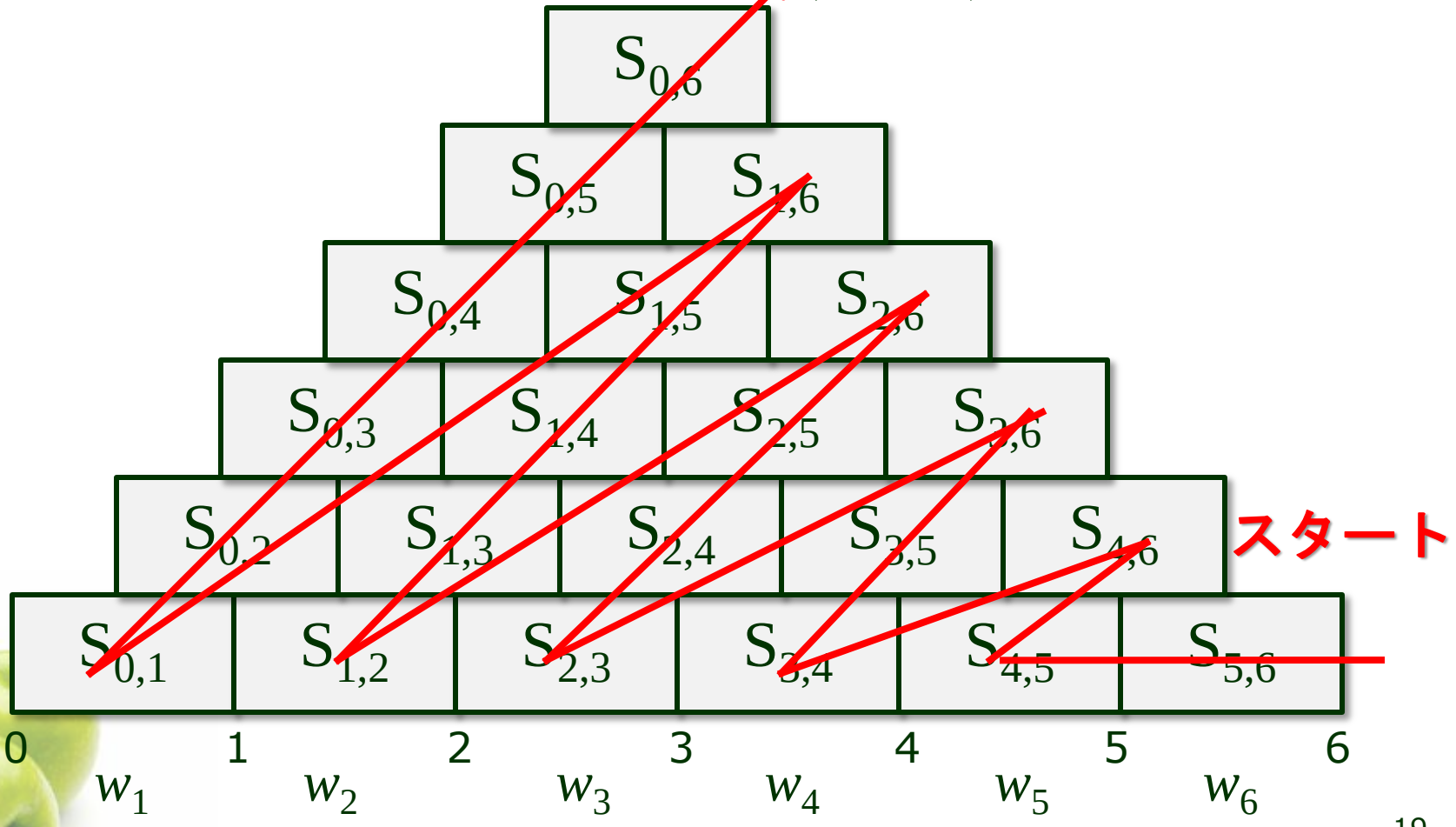
CKY法: 計算順序

- 次の順番で計算してもok (左隅)



CKY法: 計算順序

- 次の順番で計算してもok (右隅)



CKY法: データ構造

- 各CKYセル$S_{i,j}$の内容はエッジの集合
- エッジ
  - エッジID
  - 非終端記号
  - リンクの集合
    - リンク: このエッジがどのエッジから生成されたか記録したデータ構造
      - バイナリールールならエッジIDのペア
      - ユーナリールールならエッジID
      - 辞書ルールなら単語ID

チャート法

- n分岐の書換規則を扱える最も一般的な考え方のボトムアップ型パージングアルゴリズム
  - CKYは2分岐の書換規則のみ

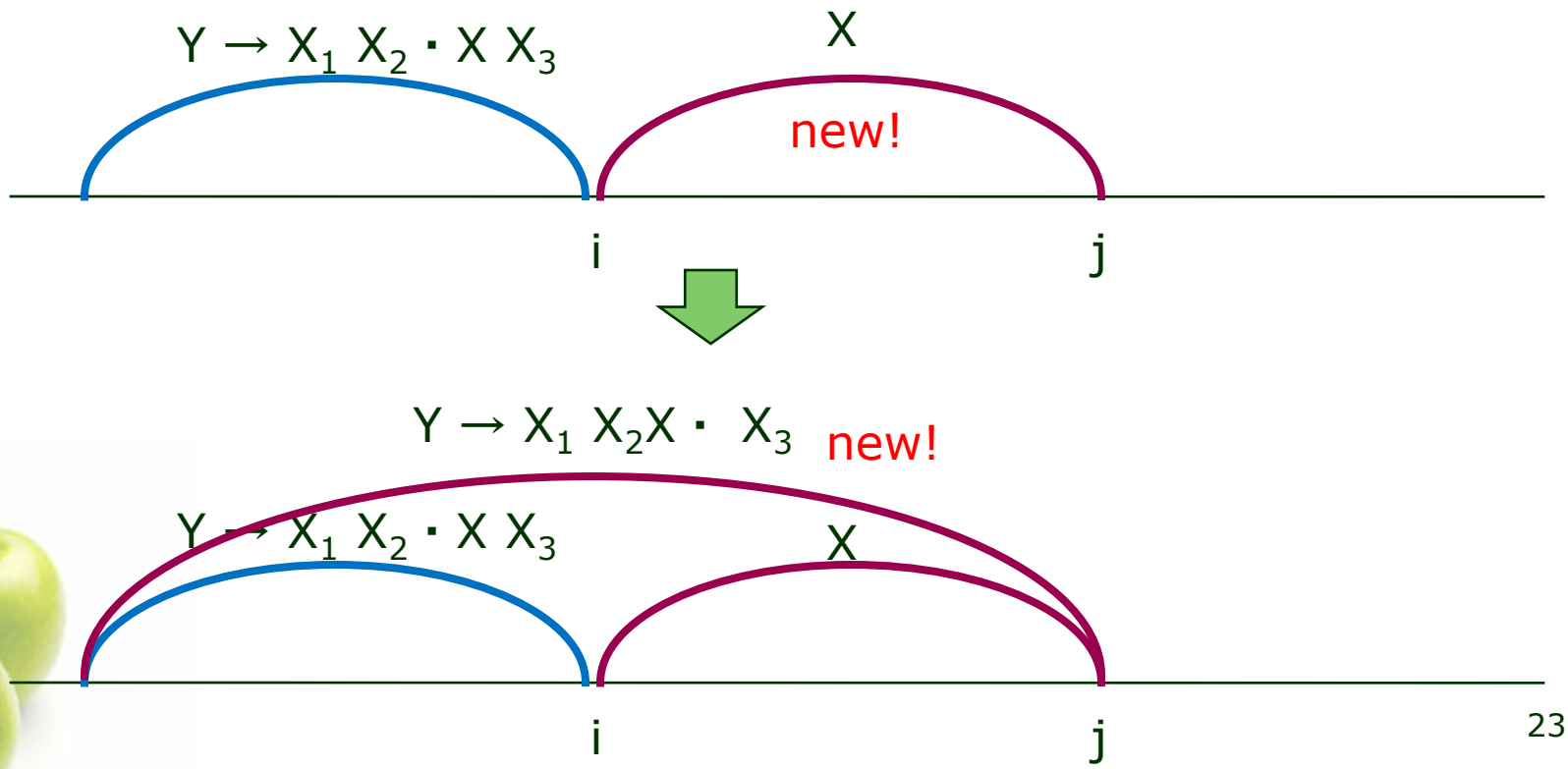
チャート法: データ構造

- エッジ
  - 活性エッジ <i, j, $Y \rightarrow X_1 \ldots X_k \cdot X_{k+1} \ldots X_n$>
    - 書換規則の途中にドットをいれたもの
      - $X_1 \ldots X_k$が解析済みということを意味する
    - エッジの左側の位置 (i)と右側の位置 (j)
      - 右側の位置はドットまでの位置のこと
  - 不活性エッジ <i, j, Y>
    - エッジの左の位置 (i)と右の位置 (j)
    - 非終端記号

チャート法: 基本的な考え方

Shift-1: 新しい不活性エッジ<i, j, X>が生成された時、

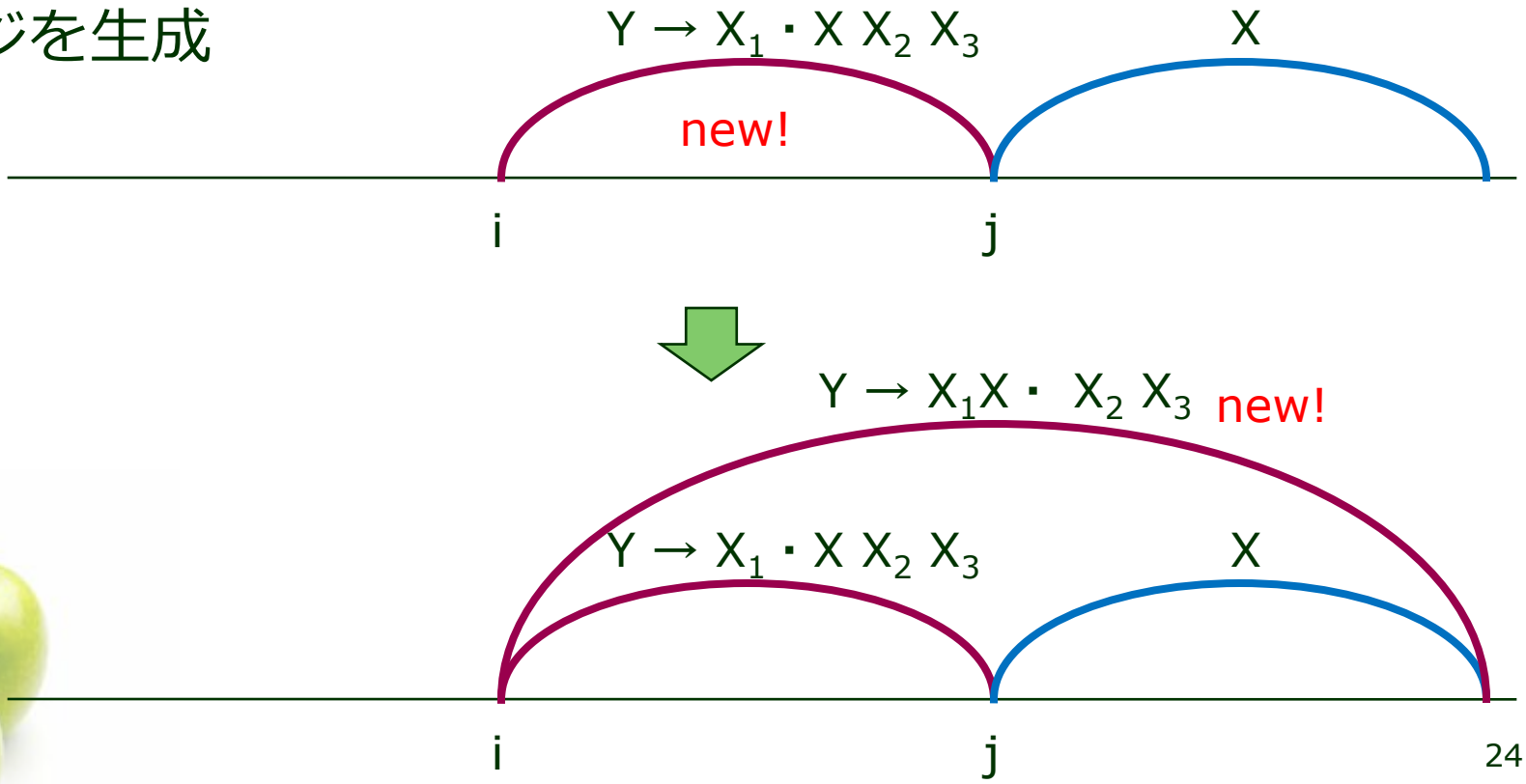
- 左側にY→…・X…の形の活性エッジがあればY→…X・…の活性エッジを生成



チャート法: 基本的な考え方

Shift-2: 新しい活性エッジ<i, j, Y→... ・ X...>が生成された時,

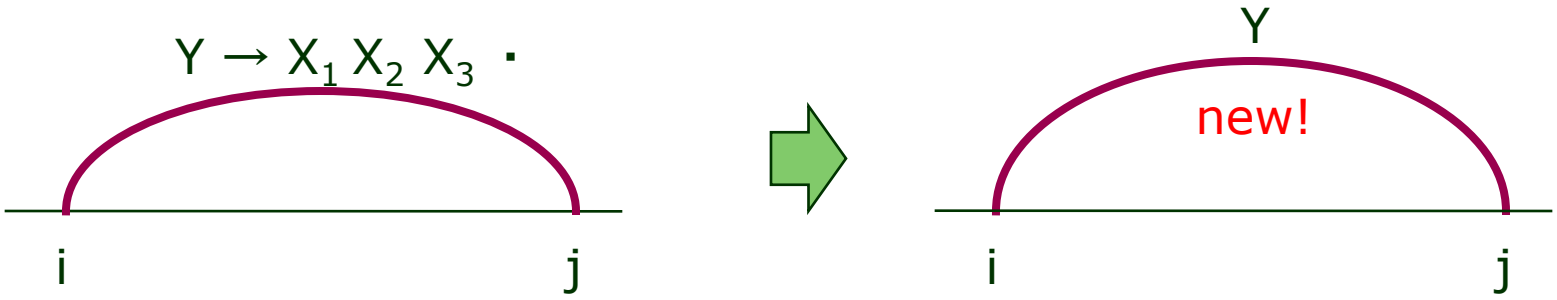
- 右側に接する全ての不活性エッジXに対し、活性エッジを生成



チャート法: 基本的な考え方

Reduce: Shift-1, Shift-2の結果新しい活性エッジ<i, j, Y→...X・>が生成された場合

- 不活性エッジ<i, j, Y>に置き換える



チャート法: アルゴリズム

```
for j = 1 to n
   Queue := Queue ∪ L(w_j)  ## 不活性エッジ<j-1, j, wjに対する非終端記号>の集合
   Chart := Chart ∪ L(w_j) ∪_{Y→β∈P}<j-1, j-1, Y→ ・ β>
while(Queue is not empty)
  E := shift(Queue)  ##EはQueueの先頭
  edges := {}; reduced_edges := {}
  if(E is 不活性エッジ<i, j, X>)
      forall F∈Chart s.t. F=<h, i, Y→ ... ・ X...>
            edges := edges ∪ <h, j, Y→ ...X ・ ...>
  if(E is 活性エッジ<i, j, Y→... ・ X...>)
      forall F∈Chart s.t. F=<j, k, X>
            edges := edges ∪ <i, k, Y→ ...X ・ ...>
  forall E' ∈ edges
      if(E' is <x, y, Y→β ・ >)
            reduced_edges := reduced_edges∪<x, y, Y>
      else
            reduced_edges := reduced_edges ∪ E'
  Queue := Queue ∪ reduced_edges; Chart := Chart ∪ reduced_edges
```

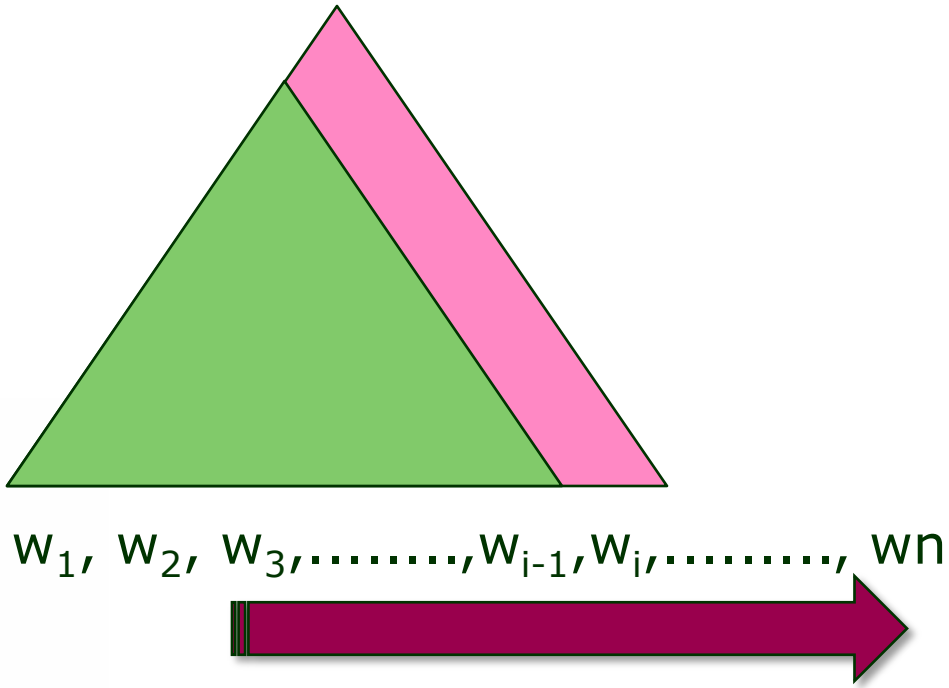
チャートとキューにエッジを格納するときにファクタリングをする

左隅解析法

- チャート法をより効率的にしたアルゴリズム
  - 活性エッジをチャートに残さなくてもok
  - 右側にエッジがないので、左側のみ解析の対象とすれば良い(アルゴリズムが簡単)

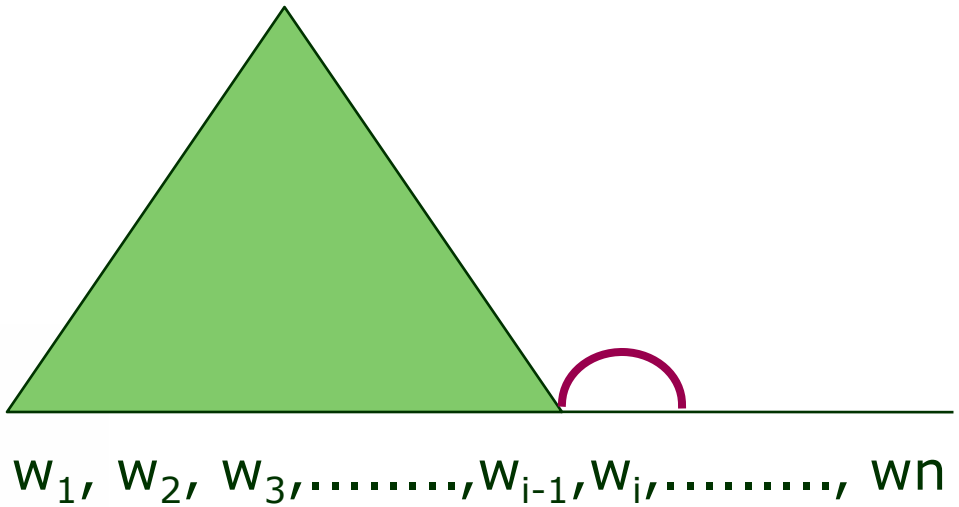
左隅解析法: 基本的な考え方

- left-to-right
  - バイナリールールに限れば、CKYの左隅解析と同じ



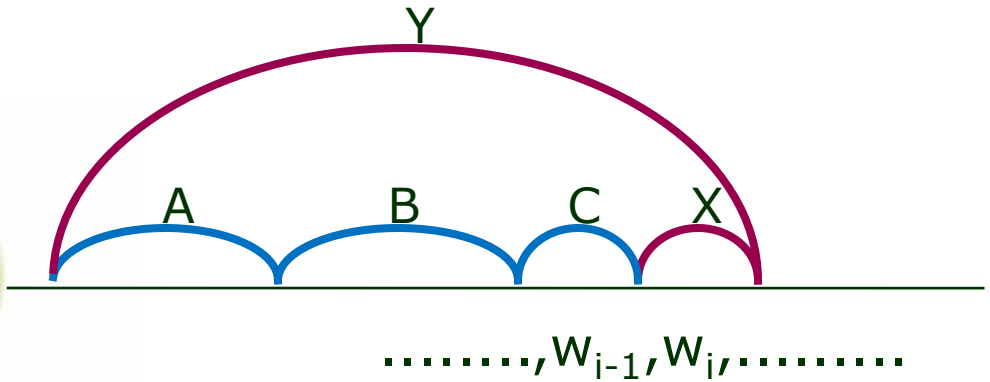
左隅解析法: 基本的な考え方 (1)

- $w_1,..,w_{i-1}$までは解析済みで、不活性エッジしか存在しないと考える
- <i-1, i, l∈L(w_i)>を新しい不活性エッジとして加える



左隅解析法: 基本的な考え方 (2)

- 新しく出来た不活性エッジ<_, i, X>に対し、
  - $Y \rightarrow X_1 \ldots X_k X$という形の全ての規則に対し、Xと連接する$X_k, .. X_1$のエッジを左に向かって探す
  - 見つかったら新しく不活性エッジ<_, i, Y>を生成
    - 新しく出来たエッジの右端は常にiなので、右側にエッジは存在しない⇒右側を気にしなくてもよい



Y→A B C X

左隅解析法: アルゴリズム

search-left(Y, β(=X_1...X_k), i,j)
    if(β is empty)
        edges := edges ∪ <i, j, Y>
    forall <h,i,X_k> ∈ Chart
        search-left(Y, X_1...X_{k-1}, h, j)

left-corner-parsing(w_1,...,w_n)
  for j = 1 to n
    Queue := L(w_j) ## <j-1, j, w_jの非終端記号>
    while(Queue is not empty)
        <i, j, X> := shift(Queue)
        forall (Y→X_1 ... X_k X) ∈ P
            edges := {}
            search-left(Y, X_1...X_k, i, j)
            Chart := Chart ∪ edges; Queue := Queue ∪ edges

チャートにエッジを格納するときにファクタリングをする

まとめ

- CFG構文解析
  - 動的計画法(dynamic programming)
    - チャートに部分構文木を残しているため、一度計算された部分構文木は二度計算されない
    - 同じ位置の同じ非終端記号を一つにまとめる(ファクタリング)
      - 同じ計算を2回以上しないようにするため
  - 出力は畳みこまれた構文木集合 (packed forest)
    - AND, ORで表現されるグラフ構造
  - CKY法、チャート法、左隅解析法
- 資料

  http://aiweb.cs.ehime-u.ac.jp/~ninomiya/ai2/