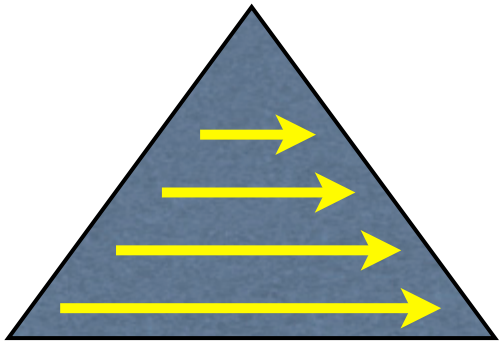
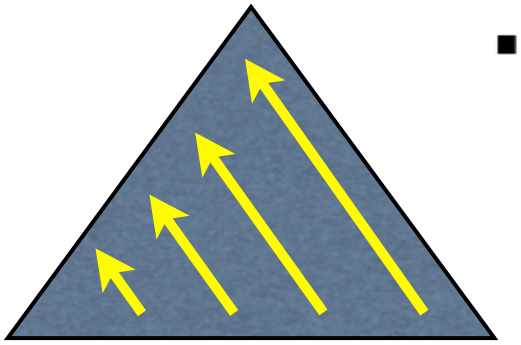
Chomsky Normal Form

- wait! how can you assume a CFG is binary-branching?
- well, we can always convert a CFG into Chomsky-Normal Form (CNF)
  - A → B C
  - A → a
- how to deal with epsilon-removal?
- how to do it with PCFG?

Any variants of CKY?



bottom-up



left-to-right

- For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k

```
score[X][i][j] = max
```

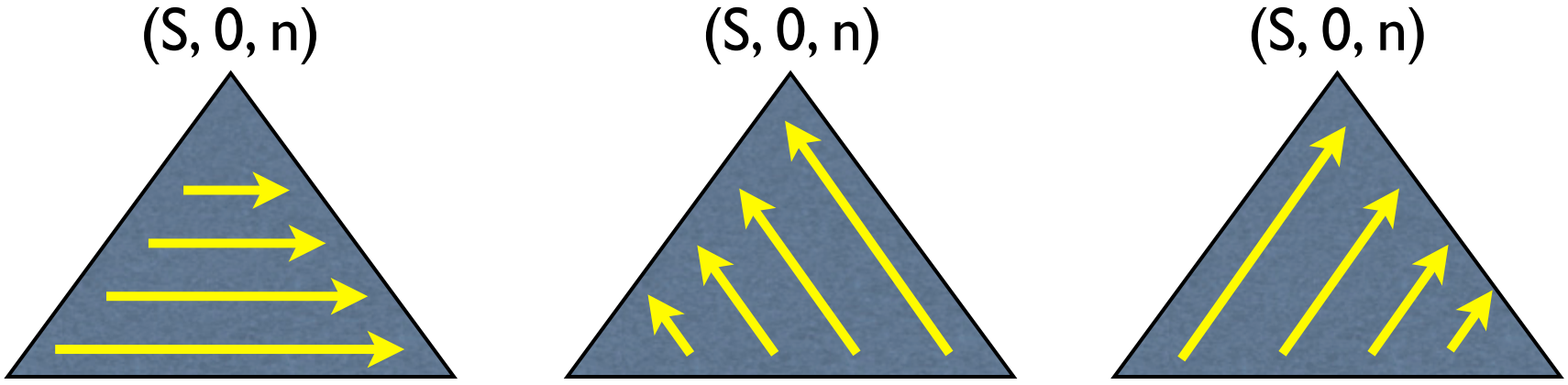
$O(n^3|P|)$

Any variants of CKY?

- For each diff (<= n)
  - For each i (<= n)
    - For each rule X → Y Z
      - For each split point k

```
score[X][i][j] = max score[X][i][j],
                     score(X->YZ) *
                     score[Y][i][k] *
                     score[Z][k][j]
```

what's the difference with shift-reduce?



bottom-up left-to-right right-to-left

$O(n^3|P|)$

Parsing as Deduction

$$\frac{(B, i, k) : a \quad (C, k, j) : b}{(A, i, j) : a \times b \times \Pr(A \rightarrow B\ C)}\ A \rightarrow B\ C$$

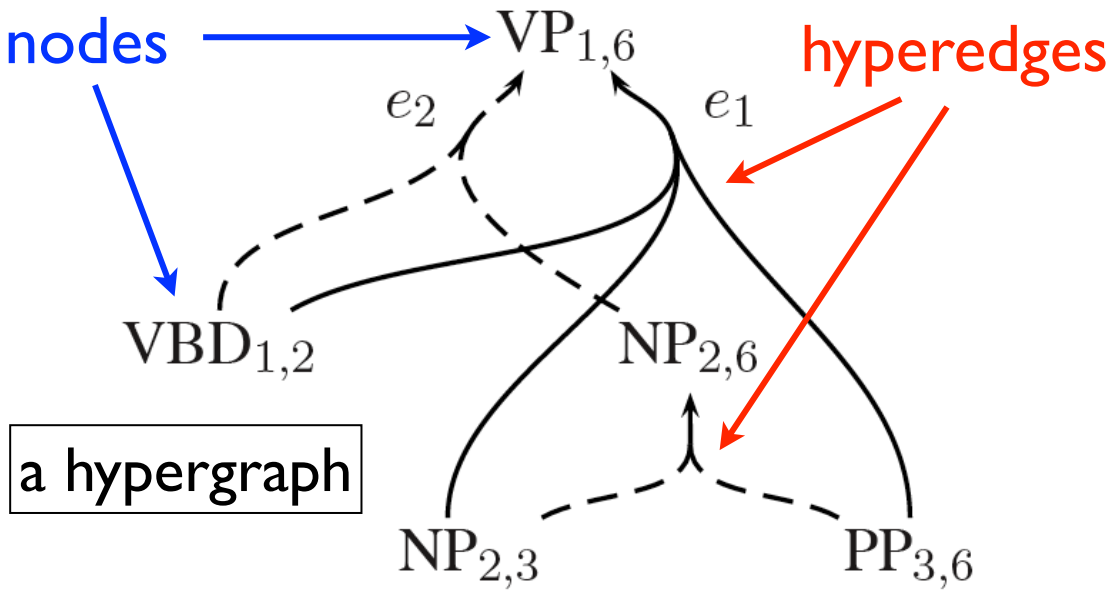
Parsing as Intersection

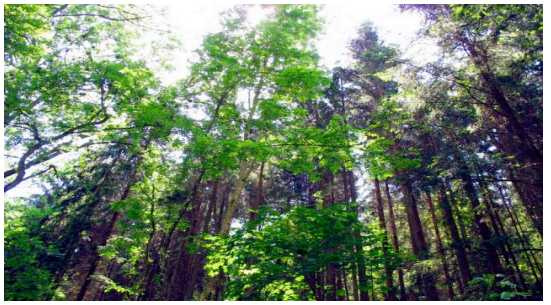
$$\frac{(B, i, k) : a \quad (C, k, j) : b}{(A, i, j) : a \times b \times \Pr(A \rightarrow B\ C)}\ A \rightarrow B\ C$$

- intersection between a CFG *G* and an FSA *D:*
  - define *L(G)* to be the set of *strings* *(i.e., yields) G* generates
  - define $L(G \cap D) = L(G) \cap L(D)$
  - what does this new language generate??
  - what does the new grammar *look like*?
- what about CFG ∩ CFG ?

Packed Forests

- a compact representation of many parses
  - by sharing common sub-derivations
  - polynomial-space encoding of exponentially large set



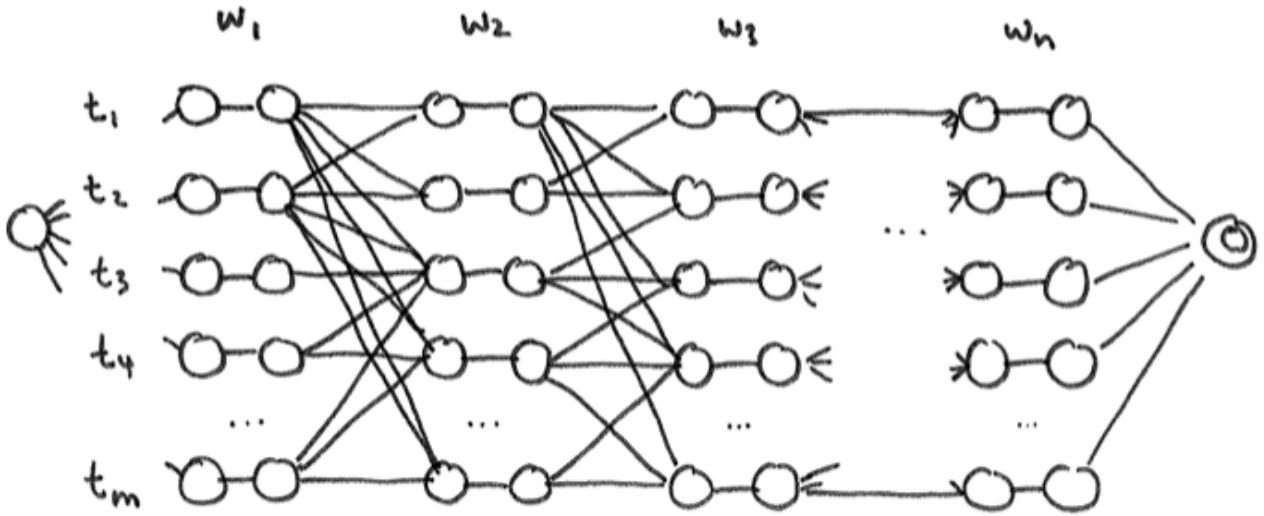


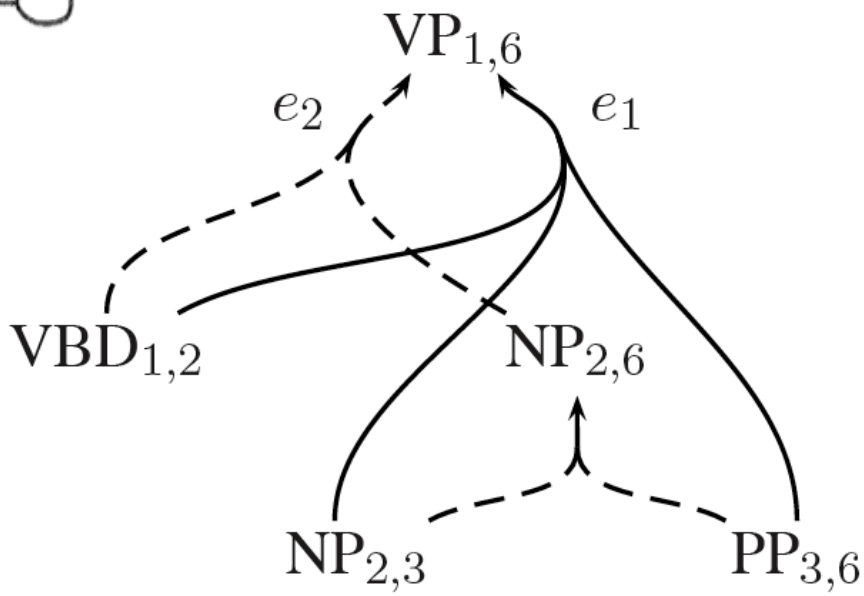
$$e_1 \quad \frac{\mathrm{VBD}_{1,2} \quad \mathrm{NP}_{2,3} \quad \mathrm{PP}_{3,6}}{\mathrm{VP}_{1,6}}$$

$_0$ I $_1$ saw $_2$ him $_3$ with $_4$ a $_5$ mirror $_6$

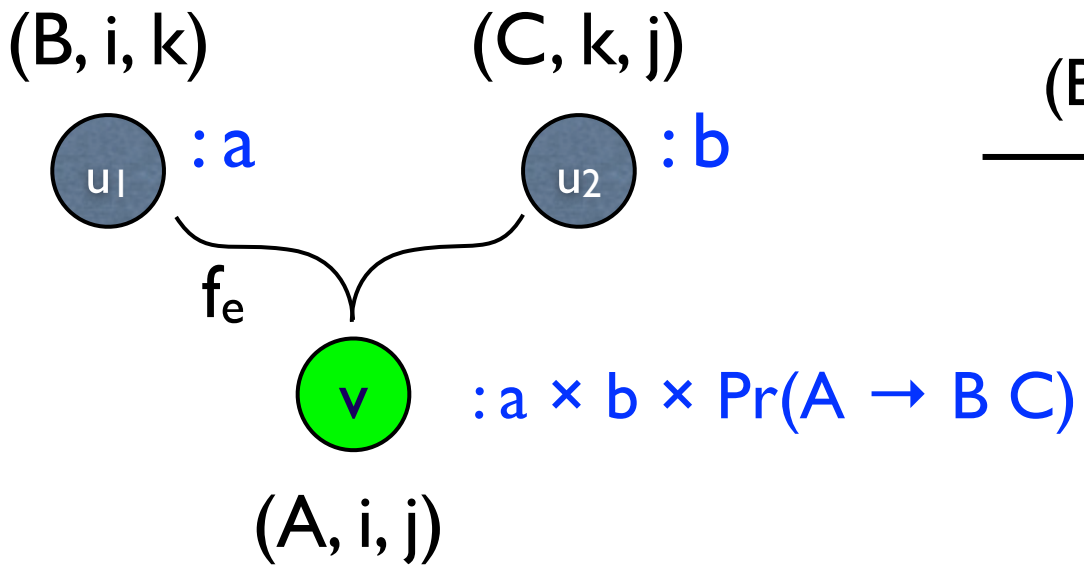
(Klein and Manning, 2001; Huang and Chiang, 2005)

Lattice vs. Forest



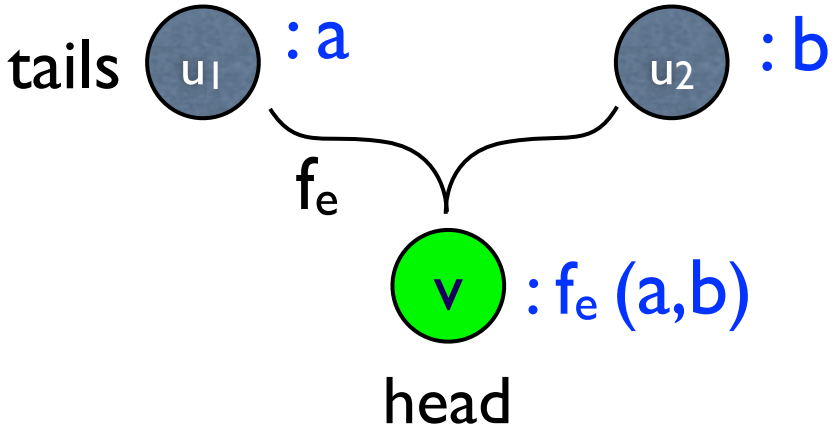


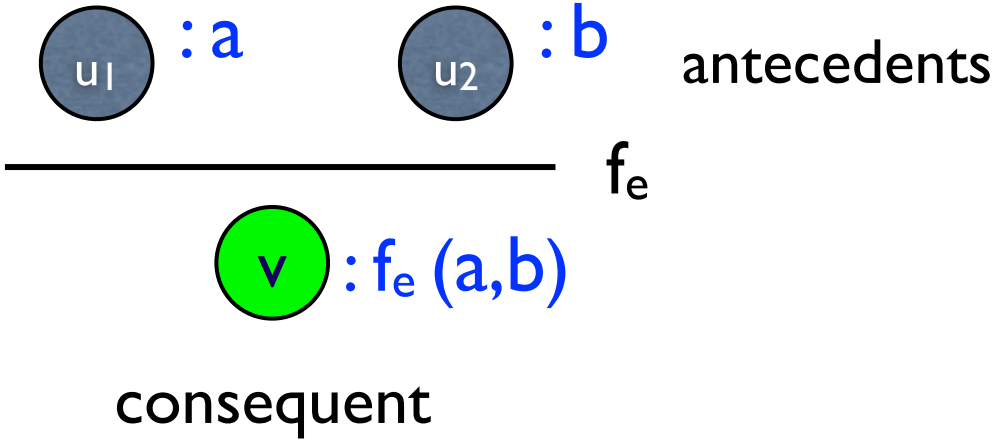
Forest and Deduction



$$\frac{(B, i, k) \qquad (C, k, j)}{(A, i, j)} \; A \rightarrow B\ C$$

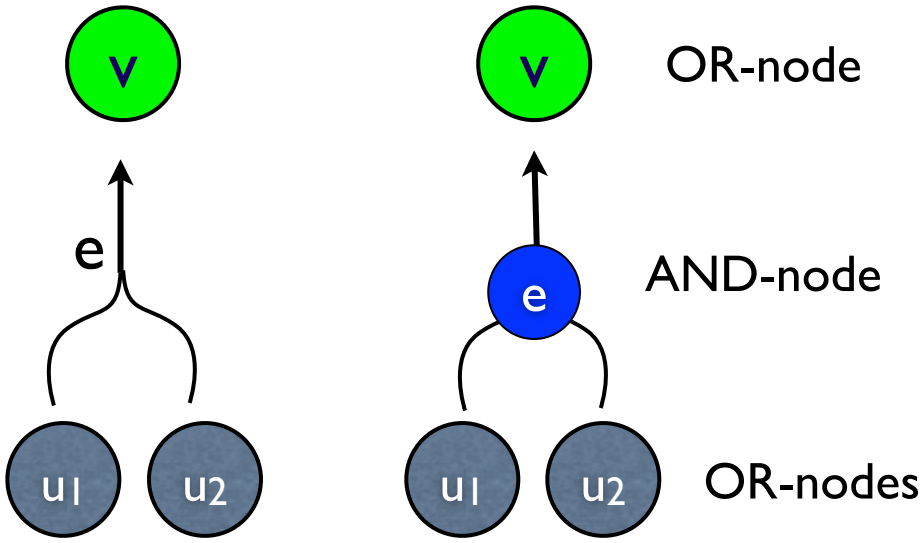
(Nederhof, 2003)





Related Formalisms

hypergraph	AND/OR graph	context-free grammar	deductive system
vertex	OR-node	symbol	item
source-vertex	leaf OR-node	terminal	axiom
target-vertex	root OR-node	start symbol	goal item
hyperedge	AND-node	production	instantiated deduction
$(\{u_1, u_2\}, v, f)$		$v \xrightarrow{f} u_1\ u_2$	$\frac{u_1 : a \quad u_2 : b}{v : f(a, b)}$



Earley Algorithm

- no binarization is needed (like CKY w/ dotted rules)
- left-to-right parsing with top-down filtering
  - in CKY, some nodes will never be used (from the top)
  - in Earley, try to build nodes when needed (from the left)

Predict:

$$\frac{(A \rightarrow \alpha.B\beta,\ i, j)}{(B \rightarrow .\gamma,\ j, j)}\ B \rightarrow \gamma \in P$$

only try to build B when needed by A!

Scan:

$$\frac{(A \rightarrow \alpha.a\beta,\ i, j)}{(A \rightarrow \alpha a.\beta,\ i, j+1)}\ w_j = a$$

Complete:

$$\frac{(A \rightarrow \alpha.B\beta,\ i, j) \qquad (B \rightarrow \gamma.,\ j, k)}{(A \rightarrow \alpha B.\beta,\ i, k)}$$

Probabilistic Earley

- probabilistic Earley due to Stolcke (1995)
- predicted item carries only the rule prob
  - different items could predict the same predicted item!
- probabilities accumulated in the complete step

Predict:

$$\frac{(A \rightarrow \alpha.B\beta,\ i,j) : p}{(B \rightarrow .\gamma,\ j,j) : \Pr(B \rightarrow \gamma)}\ B \rightarrow \gamma \in P$$

Scan:

$$\frac{(A \rightarrow \alpha.a\beta,\ i,j) : p}{(A \rightarrow \alpha a.\beta,\ i,j+1) : p}\ w_j = a$$

Complete:

$$\frac{(A \rightarrow \alpha.B\beta,\ i,j) : p \qquad (B \rightarrow \gamma.,\ j,k) : q}{(A \rightarrow \alpha B.\beta,\ i,k) : pq}$$

Implementation of Earley

- Earley implementation is much trickier than CKY
- still 3 loops, but in different order (like left-to-right cky)
  - outmost loop: right boundary *j* (left to right)
  - order of actions is important: complete > predict > scan

```
for j = 0 to n
  // first do completion, from short to long
  for i = j-1 downto 0
    // look for candidate items for completion
    for each item x = (i, j, B -> γ.) in span [i, j]
      for k = i downto 0
        for each item y = (k, i, A -> α.Bβ) in span [k, i]
          combine x and y to be (k, j, A -> αB.β) in span [k, j]
  // then do predictions and scanning
  for i = 0 to j
    for each item (i, j, A -> α.Bβ) do (repeated) prediction
    for each item (i, j, A -> α.wⱼβ) do scanning
```